EPFL

Thèse n° 9008

Efficient and Accurate Physically-Based Differentiable Rendering

Présentée le 25 novembre 2022

à la Faculté informatique et communications
Laboratoire d'informatique graphique réaliste
Programme doctoral en informatique et communications

pour l'obtention du grade de Docteur ès Sciences

par

Delio Aleardo VICINI

Acceptée sur proposition du jury

Prof. N. H. B. Flammarion, président du jury
Prof. W. A. Jakob, directeur de thèse
Prof. T.-M. Li, rapporteur
Prof. I. Gkioulekas, rapporteur
Prof. M. Pauly, rapporteur

École
polytechnique
fédérale
de Lausanne

2022

Abstract

Physically-based rendering algorithms generate photorealistic images of virtual scenes. By simulating light paths in a scene, complex physical effects such as shadows, reflections and volumetric scattering can be reproduced. Over the last decade, physically-based rendering methods have become efficient enough for widespread use. They are used to synthesize realistic imagery for visual effects, animated movies and games, as well as architectural, product and scientific visualization.

We investigate the use of physically-based rendering for inverse problems. For example, given a set of images (e.g., photographs of a real scene), we would like to reconstruct scene geometry, material properties and lighting conditions that when rendered reproduce the provided reference images. Such a task can be formalized as minimizing the difference between reference and rendered images over the space of scene parameter. The resulting non-linear objective functions can be minimized by using a differentiable renderer and gradient descent. However, the complexity of physically-based light transport algorithms makes it infeasible to compute parameter gradients by naïvely using off-the-shelf automatic differentiation (AD) tools. In this thesis, we present several novel algorithms that efficiently and accurately compute gradients of a physically-based renderer.

First, conventional AD cannot scale to the complexity that is due to long light paths. For example, differentiable rendering of participating media requires differentiating light paths with potentially hundreds of interactions. We introduce path replay backpropagation, an unbiased method that enables differentiation of multiple-scattering light transport at a computational and storage complexity similar to the original simulation. Leveraging the invertibility of local Jacobians, our method efficiently differentiates even perfectly specular scattering and unbiased volume rendering using delta tracking. Path replay backpropagation is the first unbiased differentiable rendering method that scales to an arbitrary number of differentiated variables and an unbounded number of scattering events.

Second, differentiating a rendering algorithm requires handling parameter-dependent discontinuities due to occlusion. This is essential for using such methods to reconstruct the geometry of objects. We propose a method that accurately differentiates renderings of surfaces represented by signed distance functions (SDFs). We leverage their spatial structure to construct a reparameterization of the integration domain. This reparameterization accounts for gradients due to occlusion changes and enables image-based reconstruction of objects of arbitrary topology, without requiring strong priors or

knowledge about object silhouettes.

Lastly, inverse rendering has led to renewed interest in developing non-standard scene representations that are amenable to optimization. In the last part of the thesis, we introduce a novel transmittance model for participating media. This model allows representing scenes containing opaque surfaces as a scattering volume. This unified representation can be used to compress complex scenes into a sparse volumetric data structure. The compressed representation is visually almost identical to the original high-resolution scene. Our new model further benefits inverse rendering, recovering relightable volumetric representations that more faithfully capture opaque surfaces than prior models.

Keywords: differentiable rendering, inverse rendering, differentiable Monte Carlo, path replay backpropagation, delta tracking, signed distance functions, scene reconstruction, non-exponential media, level of detail, scene representation

Zusammenfassung

Physikalisch basierte Renderingalgorithmen erzeugen realistische Bilder virtueller Szenen. Durch die Simulation von Lichtpfaden können komplexe physikalische Effekte wie Schatten, Reflexionen und volumetrische Streuung reproduziert werden. Im Laufe des letzten Jahrzehnts wurden physikalisch basierte Renderingmethoden effizient genug für eine breite Verwendung. Sie werden eingesetzt, um realistische Bilder für visuelle Effekte, animierte Filme und Videospiele sowie Architektur-, Produkt- und wissenschaftliche Visualisierungen zu generieren.

Wir untersuchen die Verwendung von physikalisch basiertem Rendering für inverse Probleme. Beispielsweise möchten wir ausgehend von Referenzbildern (z.B. Fotografien einer realen Szene) die Szenengeometrie, Materialeigenschaften und Lichtverhältnisse rekonstruieren, die beim Rendern die Referenzbilder reproduzieren. Das Ziel ist, den Unterschied zwischen Referenz- und gerenderten Bildern über den Szenenparameterraum zu minimieren. Diese Kostenfunktion kann mithilfe eines differenzierbaren Renderers und dem Gradientenverfahren minimiert werden. Die Komplexität der Lichtsimulationen erschwert die Anwendung herkömmlicher automatischer Differenzierung (AD). Wir stellen neue Algorithmen vor, die effizient und genau Ableitungen einer Lichtsimulation berechnen.

Konventionelle AD kann nicht mit der Komplexität von langen Lichtpfaden umgehen. Volumetrische Streuung erfordert zum Beispiel die Ableitung von Lichtpfaden mit möglicherweise Hunderten von Interaktionen. Wir führen Path Replay Backpropagation ein, eine erwartungstreue Methode, welche die Ableitung des Lichttransports bei einer Rechen- und Speicherkomplexität ähnlich der ursprünglichen Simulation ermöglicht. Unter Ausnutzung der Umkehrbarkeit lokaler Jakobimatrizen differenziert unser Verfahren effizient sogar perfekte Spiegelstreuung und erwartungstreues Volumenrendering. Path Replay Backpropagation ist die erste erwartungstreue differenzierbare Renderingmethode, die auf eine beliebige Anzahl Parameter und eine unbegrenzte Anzahl von Lichtinteraktionen skaliert.

Weiterhin erfordert die Differenzierung die Berücksichtigung parameterabhängiger Unstetigkeiten aufgrund von Okklusionen. Dies ist notwendig, um mit solchen Methoden die Geometrie von Objekten zu rekonstruieren. Wir schlagen eine Methode vor, die Renderings von Oberflächen, die durch vorzeichenbehaftete Abstandsfunktionen dargestellt werden, akkurat differenziert. Wir nutzen die räumliche Struktur dieser Funktionen, um eine Reparametrisierung der Integrationsdomäne zu konstruieren. Diese berücksichtigt Gradienten aufgrund von Okklusionsänderungen und ermöglicht die Re-

konstruktion von Objekten beliebiger Topologie, ohne die Kenntnis von a-priori Informationen oder Silhouetten.

Inverse Probleme haben zudem das Interesse an der Entwicklung von neuen, zur Optimierung geeigneten, Szenenrepräsentationen wiedererweckt. Im letzten Teil der Arbeit stellen wir ein neuartiges Transmissionsmodell für Volumen vor. Dies ermöglicht die Darstellung von Szenen mit opaken Oberflächen als Volumen. Diese neue Repräsentation kann verwendet werden, um komplexe Szenen in ein komprimiertes Volumen umzuwandeln. Unsere Methode erstellt eine volumetrische Darstellung, die visuell fast identisch mit der ursprünglichen Szene ist. Unser neues Modell bietet weiterhin Vorteile für das inverse Rendering, indem es Volumen rekonstruiert, die opake Oberflächen besser modellieren als frühere Modelle.

Schlüsselwörter: differenzierbares Rendering, inverses Rendering, differenzierbare Monte-Carlo-Integration, Path Replay Backpropagation, Delta Tracking, vorzeichenbehaftete Abstandsfunktionen, Szenenrekonstruktion, nicht-exponentielle Volumen, Detaillierungsgrad, Szenenrepräsentation

Acknowledgements

I would like to express my sincere gratitude towards my advisor Wenzel Jakob for his support and guidance during my time at EPFL. He was closely involved with my research projects and worked tirelessly to ensure their success. Without his work on the Mitsuba renderer and underlying compiler, much of our research on differentiable rendering would not have been possible. I would also like to thank Ioannis Gkioulekas, Tzu-Mao Li and Mark Pauly for serving on the jury of my thesis and providing valuable feedback. I thank Nicolas Flammarion for his service as the president of the jury.

The research presented in this thesis has been carried out with the support of many collaborators. I would like to express my gratitude to Sébastien Speierer for his help on my last two projects and for working closely with me on incorporating some of our research into the Mitsuba renderer. I would also like to thank Merlin Nimier-David and Tizian Zeltner for their work on the Mitsuba renderer and associated paper, as well as many useful discussions and support over the last five years. I thank Anton Kaplanyan for advising me during my internship at Meta, which led to one of the publications presented in this thesis. I also thank Vladlen Koltun for his collaboration on the first project of my Ph.D.

I would also like to thank the other members of the Realistic Graphics Lab. I had many interesting and fun discussions with Baptiste Nicolet, Guillaume Loubet, Mandy Xia, Mariia Soroka, Miguel Crespo, Nicolas Roussel and Ziyi Zhang. I would also like to thank Pauline Raffestine for all her help with administrative matters.

I also want to thank all the students I got to supervise and work with: Damien Martin, Ekrem Fatih Yilmazer, Guirec Maloisel, Hugues Saltini, Nathan Greslin, Valentin Borgeaud and Vincent Tournier. It was inspiring and motivating to be able to work with so many great students.

I am thankful for all the great friends I made while living in Lausanne. The Ph.D. would have been much less enjoyable without the company of Anna Vybornova, Bharath Narayanan, Bruno Schmitt, Edoardo Remelli, Erik Frank, Fabian Latorre, Julian Englert, Lola Martini, Ron Rabi, Sandra Siby, Silvio Müller, Thijs Vogels and Vivek Jayaram.

Special thanks also go to my girlfriend Elisa Melo for being extremely supportive of me and my work, in particular during intense SIGGRAPH deadlines and stressful phases. I could not wish for a more loving and caring partner.

Finally, I would like to thank my parents, Sandro and Sabine, as well as my brother Anteo. I would not have completed this Ph.D. without their unconditional love, support and encouragement over all these years.

Third-party materials. The LaTeX template used for this thesis was provided by Tizian Zeltner. Several figures are based on third-party images and 3D objects:

- The scene in Figure 1.1 was created by *Architectural Topics* on YouTube.
- Many figures use 3D models from the *Stanford 3D scanning repository*.
- Figure 3.7 features a cloud image by *Free Nature Stock* (on `www.stocksnap.io`), a photo of an ear by *Burst* (on www.stocksnap.io) and a photo of a candle by *Pixabay* (on www.pexels.com).
- The *Wheatfield with Cypresses* painting is by Vincent Van Gogh (Figure 4.8).
- The Living Room scene was created by Blendswap user *Wig42* and converted to Mitsuba by Benedikt Bitterli (Figure 5.5).
- The smoke plume volume is from `www.openvdb.org` (Figure 5.8).
- Several 3D models are from the *Virtual Museums of Małopolska* (Figure 6.1, Boar in Figure 6.9).
- The Hotdog scene was created by Blendswap user *erickfree* (Figure 6.9).
- The Cranium mesh was provided by the Smithsonian museum (Figure 6.10).
- The Statue mesh was provided by the Statens Museum for Kunst (Figure 6.11).
- The Chair mesh was created by Blendswap user *1DInc* (Figures 6.9, 6.10).
- The Head mesh was created by Lee Perry-Smith and downloaded from Morgan McGuire's computer graphics archive (Figure 6.9).
- The Lego scene was created by Håvard Dalen (Figures 6.15, 7.14).
- Several scenes were created by Matthew Chapman at Meta Reality Labs (Building, City, Trees in Figure 7.12).
- The Armchair mesh was created by Blendswap user *Kilt2007* (Figures 7.3, 7.6).
- The Fractal object was created by Blendswap user *3dphotosystems* (Figure 7.12).
- The Tree Stump model was created by Blendswap user *rubberduck* (Figure 7.5).
- The Drums scene was created by Blendswap user *bryanajones* (Figure 7.14).
- We further used several HDR images by *Poly Haven*.

Funding The research in this thesis was partially supported by the Swiss National Science Foundation (SNSF) as part of grant 200021_184629.

To Sabine and Sandro

Table of Contents

List of Figures

List of Algorithms

1 | Introduction

1.1 Motivation

As humans, one of the primary ways in which we perceive the world is by using our sense of vision. We obtain a tremendous amount of information by leveraging various visual cues. Vision provides an evolutionary advantage for essential tasks such as obtaining food and evading danger. Not only can we recognize objects and people, but we also use our stereo vision to accurately judge the distance of objects. We further intuitively infer useful information from the complex ways that light interacts with objects. For example, shading and cast shadows help determine the 3D shape of objects or the ground that we walk on. We can distinguish between metal and glass objects by observing how they reflect or refract incident illumination [1]. When assessing the color of a surface, our brain implicitly considers the illumination conditions [2]. In this thesis, our goal is to develop algorithms that infer various physical properties of the real world from images. Different from human perception, algorithms can directly leverage our mathematical understanding of the underlying light transport.

Concretely, we build on physically-based rendering algorithms [3, 4], which explicitly simulate the interaction of light with objects. These algorithms synthesize photorealistic images of virtual scenes. A virtual scene consists of a description of object geometries, surface material parameters, light sources and one or more virtual cameras. Figure 1.1 shows a rendering of an example scene. Physically-based rendering algorithms then explicitly simulate *light paths* to produce images containing accurate reflections, shadows, volumetric scattering and indirect illumination. This is a computationally expensive process, but the development of efficient algorithms and fast computer hardware has enabled the widespread adoption of physically-based rendering in movie production, games, architecture and product visualization. Physically-based methods have largely replaced heuristics and approximate algorithms [5]. The reason is simple: accurate rendering models are easier to control and lead to more predictable results with fewer rendering artifacts. While certain approximate algorithms are still necessary for real-time rendering applications (e.g., games), their use is expected to diminish over time in favor of fully physically-based methods [6].

In this thesis, we are primarily concerned with the associated *inverse problem*. Given a set of images or visual observations, we want to infer the parameters of a corresponding virtual scene. Similar to previous and concurrent work, we approach this problem as an

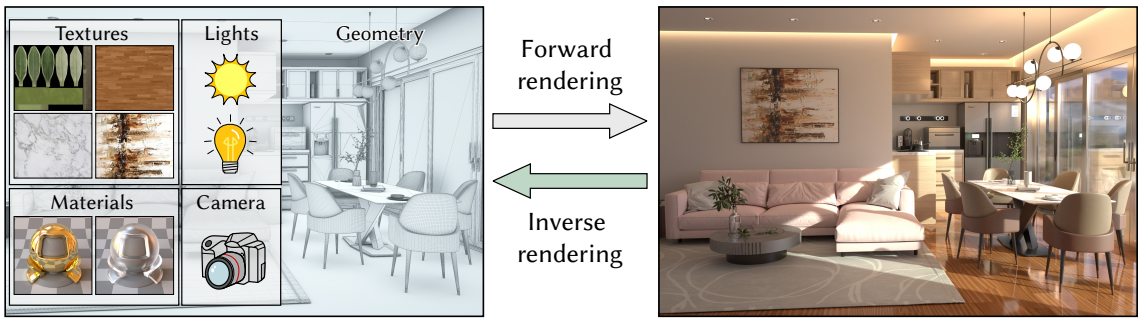


Figure 1.1: A scene description usually consists of geometry, surface textures, material models, light sources and a virtual camera. Rendering produces a photorealistic image of such a virtual scene. We will often refer to this process as *forward* rendering. This is in contrast to *inverse* rendering, which attempts to reconstruct a scene description given one or more images.

iterative optimization. Starting from some initial guess of scene parameters, we render our virtual scene and (numerically) compare the result to the reference images. In each iteration, we then update our parameters to gradually more closely approximate the reference observations.

Reconstructing the real world from images is generally considered to be a topic of *computer vision*. However, there is a mismatch between the generality and accuracy of forward rendering algorithms and the lighting models used in most computer vision research. Many existing 3D reconstruction algorithms for example ignore the effect of shadows and interreflections. Our goal is to apply modern rendering algorithms to inverse problems, thereby reducing the gap between the light transport models used in computer vision and graphics. Similar to forward rendering, we believe that physically-based algorithms will reduce the need for application-specific heuristics and allow to robustly solve both current and future problems. Our goal is to develop algorithms that can solve general inverse problems specified in terms of light transport. The use of these methods is not limited to 3D scene reconstruction, as we will see when discussing related work.

Historically, the understanding of light transport and its use for inverse problems were often intertwined. For example, Ibn-Sahl (940–1000 AD) is credited for discovering the law of refraction and using his insights to design better refractive lenses [7]. Johannes Kepler (1571-1630 AD) and Isaac Newton (1643-1727 AD) used their understanding of light to design and build telescopes. In the twentieth century, the theory of neutron scattering was used to design and improve radiation shielding and nuclear reactors. Both in astronomy and atmospheric sciences solving inverse problems under

physical constraints is common. All these problems are highly related, and we hope to provide some of the building blocks that can help tackle general inverse problems related to light and radiation scattering.

1.2 Differentiable Monte Carlo rendering

In this section, we will briefly introduce the general framework used throughout the thesis. We will provide detailed definitions of the fundamentals of light transport and rendering algorithms at a later point.

Mathematically, the intensity of light incident at a pixel of a virtual sensor can be written as a high-dimensional integral over light paths. Physically-based rendering algorithms estimate pixel colors using *Monte Carlo integration* [3, 8]. Practically, this means that they trace many light paths through the scene to estimate the contribution of light sources to the sensor of a virtual camera. When light paths encounter an object, they either scatter or get absorbed. To render even a moderately complex scene, millions of light paths have to be simulated to estimate the color of each pixel of the output image. The physical analogy of this process is how photon particles scatter on objects in a real scene.

While costly, Monte Carlo methods have the key advantage that they are *unbiased*. The only error in the rendered image is due to *variance*, which manifests in the form of per-pixel noise (similar to noise in nighttime photography). Monte Carlo methods further scale well to high-dimensional problems (e.g., integration over long light paths) and, unlike deterministic quadrature methods, do not suffer from the *curse of dimensionality*. They work using the original scene representation and do not require a specialized discretization of the scene. This is an advantage compared to finite element methods, which require adequate discretization to obtain plausible results. While forward rendering is not a solved problem, Monte Carlo methods are effective at simulating a large range of physical effects. The mathematical framework is very flexible and can easily be extended by specialized techniques for cases not well handled by a general method. Tailored rendering methods have been developed for effects such as scattering volumes [9, 10], human skin [11, 12, 13], hair [14], glints [15, 16] and layered materials [17].

We can leverage these favorable properties of Monte Carlo integration for inverse problems by developing *differentiable* Monte Carlo rendering algorithms. By differentiating the output of a rendering algorithm, we can solve complex optimization problems using gradient descent. Differentiating a physically-based forward model is not a new

idea and follows a general trend. The widespread use of gradient descent to train artificial neural networks and other machine learning models has inspired a shift toward using *differentiable programming* for various inverse problems. Conceptually this is a very simple, general approach, although in practice there are some challenges. Machine learning frameworks providing automatic differentiation (AD) methods can be used to implement general differentiable computations. The extensive adoption of these AD frameworks has enabled the rapid exploration of differentiable computation, eliminating the need for tedious manual implementation of derivative code. Automatic differentiation is not a new idea either, but its use has increased dramatically thanks to the availability of easy-to-use tools (primarily in the form of Python libraries).

This dissertation focuses on differentiating Monte Carlo rendering algorithms to solve inverse problems related to light transport. We aim to develop general, principled methods, without introducing major approximations or ignoring relevant physical effects. Ultimately, we expect differentiable Monte Carlo rendering algorithms to benefit a large range of inverse problems. Similar to forward rendering, handling the full complexity of light transport allows accounting for many important physical effects in a single unified framework. This reduces the need for application-specific methods and heuristics. Many prior approaches for inverse light transport problems ignore multiple scattering or only consider simple special cases (e.g., flat geometry). With the development of differentiable Monte Carlo rendering, there is no fundamental reason for inverse methods to be significantly less physically-based than their forward counterparts. This thesis proposes several methods that increase the applicability of differentiable rendering for important problems. We show that accurate gradients can often be obtained at a similar complexity as forward rendering.

While our methods are physically-based, we do restrict ourselves to considering geometric optics and ignore wave effects such as interference and diffraction. We assume steady-state light transport and do not consider temporal effects such as light moving with the speed of light or materials heating up and glowing. These assumptions are commonly made in forward rendering, and going beyond them is out of scope for this thesis. Additionally, differentiable rendering is not a silver bullet: practical applications oftentimes do require some amount of problem-specific initialization or parameterization of the scene. Nevertheless, we believe differentiable Monte Carlo rendering will be an important baseline and building block for many inverse problems.

1.3 Inverse rendering and related problems

In this section, we provide a high-level overview of important inverse rendering and related problems. The thesis is focused on techniques relevant to image-based reconstruction. However, (differentiable) Monte Carlo estimators also have applications in other fields. We believe that some of the methods developed in this thesis could potentially also be adapted to these problems. This section serves to further motivate the work presented in this thesis but is not a comprehensive overview of the vast field of inverse problems. Chapter 4 provides a focused review of more closely related prior and concurrent work from the computer graphics and vision communities.

Reconstruction. As mentioned above, a primary application of differentiable rendering algorithms is 3D reconstruction. Given a set of reference images, we aim to recover physically plausible scene parameters. Such methods are useful for any type of application where a digital copy of real-world objects is needed (e.g., for visual effects or cultural preservation). By considering physically-based rendering in the optimization loop, we aim to recover editable and relightable scene representations. Differentiable rendering is particularly useful for scenes with complex light scattering. For example, to recover the material properties of objects exhibiting subsurface scattering [18]. We will review related work on 3D reconstruction in Chapter 4.

Computational design. The term *computational design* describes the use of optimization methods to (semi-)automatically design physical items. The idea is to use optimization methods to design objects satisfying complex constraints, for which it is impossible to come up with a suitable design by hand. Computational design is often associated with satisfying certain mechanical constraints. For example, one can optimize the shape of objects to make them balance [19] or design spinning tops of unusual shapes [20].

However, interesting problems can also be specified in terms of light transport. For these applications, we can use differentiable rendering to optimize the shape or material of objects. Such problems include the fabrication of translucent objects of a desired appearance [21, 22] and *end-to-end* optimization of optical systems and camera lenses [23]. Another interesting problem is optimizing combustion chambers and gas turbines considering heat transfer. Heat transfers by convection, conduction and radiation. While convection and conduction are commonly solved using finite element methods, the radiative transfer can be simulated by Monte Carlo integration. Such heat transport simulations have for example been proposed to simulate gas turbines [24]. Differentiable

Monte Carlo methods have also been used to analyze and improve combustion chambers of complex geometry [25].

Neutron transport. Monte Carlo methods were originally invented to simulate neutron scattering in nuclear reactors and weapons [8, 26, 27]. The problem of neutron scattering is mathematically very similar to light transport, and many techniques from neutron transport have been imported into rendering. Given its origins in nuclear engineering, it is not surprising that neutron transport simulations have been used for inverse problems. Unlike many other fields, differentiating Monte Carlo estimators is common in the neutron transport literature. Already in 1967, Mikhailov [28] showed how to differentiate Monte Carlo neutron transport estimators. In the same year, Brainina et al. [29] proposed to use differentiable Monte Carlo estimators to optimize the shape and composition of radiation shielding. Sidorenko and Khisamutdinov [30] optimize a radiation shielding consisting of two layers and mention the issue of discontinuous integrands, which is also relevant in differentiable rendering. Differentiable Monte Carlo methods have further been used for *sensitivity analysis* of the criticality of a nuclear reactor [31, 32].

Atmospheric science. Directly measuring gas concentrations, temperature and pressure in the whole atmosphere is infeasible. However, all these important quantities affect the way both visible and non-visible light is scattered. It is therefore common to recover the physical parameters of the atmosphere by solving inverse problems. For example, already in the 1930s Götz et al. [33] inferred Ozone concentration from the scattered sunlight observed from the ground. There is a large body of work on using various combinations of satellite and aerial measurements to infer temperature and gas concentrations. Different from typical rendering problems, atmospheric observation oftentimes leverages measurements of radiation outside the visible range (e.g., infrared radiation). However, many existing methods do not account for multiple scattering of light in three dimensions [34]. We believe that physically-based differentiable rendering could further improve these methods and related problems such as cloud reconstruction [35].

Computational imaging. The field of computational imaging encompasses image acquisition techniques that produce images by solving inverse problems. Specifically, such techniques are common in microscopy, medicine, geophysics and astronomy. In many cases, specialized techniques build on domain-specific heuristics and assume heavily simplified physical models. The use of such approximate models oftentimes leads to

suboptimal results. For example, both positron emission tomography (PET) and computerized tomography (CT) scans suffer from artifacts due to reconstruction algorithms not considering multiple scattering of radiation [36, 37, 38]. In many applications, it could be possible to improve the results of existing approximate methods by using a differentiable light transport simulation to further refine the solution. In particular, accounting for multiple scattering could be beneficial in many contexts.

Partial differential equations. Partial differential equations (PDEs) describe a range of important natural phenomena such as heat transfer or diffusion processes. These problems are commonly approached by partitioning the domain into triangles or tetrahedra and then solving a discretized differential equation. Such finite element methods have also found early use in rendering, but have since largely been replaced by Monte Carlo methods. The reason is that discretizing the domain is expensive, and the resulting solutions often suffer from artifacts. On the other hand, Monte Carlo methods handle complex geometry without significant preprocessing.

It turns out that for certain PDEs we can formulate practical Monte Carlo estimators. The *walk on spheres* method [39, 40, 41] estimates the solution to PDEs by sampling random walks on the domain. The structure of these algorithms closely resembles Monte Carlo rendering. Recently, these methods have been imported into computer graphics [42, 43]. Similar to rendering, the generality and simplicity of Monte Carlo methods enable solving complex problems without discretization artifacts. Differentiating such solvers could allow addressing inverse problems. We briefly touch on this topic in Chapter 5, as our proposed differentiable rendering method can be applied to PDEs [44].

Machine learning. Another use of differentiable rendering algorithms is their combination with machine learning methods. For example, one could envision a generative model that generates scene parameters that could then be rendered using a physically-based renderer. By using a differentiable renderer, such a system could be trained end-to-end, thus potentially opening up text-based 3D scene or model generation.

1.4 Summary of original contributions

In this section, we briefly summarize the main contributions presented in this thesis.

Path replay backpropagation [45]. We propose a novel method to efficiently differentiate multiple scattering light transport. The complexity of multiple scattering makes it infeasible to use conventional automatic differentiation. When using AD, gigabytes of memory are required to hold information related to the derivative computation. This is problematic, because for many important optimization problems, the optimized scene itself might already use a large amount of memory. We can simply not afford for the differentiation process to tie up all the available memory. We present a simple, yet general method that differentiates light paths at a computational and memory complexity that is comparable to forward rendering. We call our method *path replay backpropagation*, as it relies on re-tracing light paths to estimate derivatives. Up to numerical differences, it computes the same derivatives we would get from AD, but without the extreme storage overheads. Our method is well-suited for the highly dynamic nature of light transport algorithms. For example, it for the first time makes it feasible to differentiate unbiased volume rendering methods.

Differentiable signed distance function rendering [46]. Another important problem in differentiable rendering is the reconstruction of 3D shapes. In forward rendering, most shapes are represented as triangle meshes, which can be rendered very efficiently. However, triangle meshes are cumbersome to use for inverse problems, as their topology is generally fixed. A common solution to this problem is to instead use an *implicit* surface representation, where the surface is represented as the zero-level set of a function $f\colon \mathbb{R}^3 \to \mathbb{R}$. A particularly well-structured class of implicit representations are *signed distance functions* (SDFs). A signed distance function measures the signed distance to the surface that is defined to be the zero-level set. We present a new method for differentiable rendering of SDF surfaces. This enables the reconstruction of shapes of arbitrary topology from a set of reference images.

Non-exponential media for scene representation [47]. The third contribution is related to the scene representation itself. An interesting alternative to surface-based scene representation are participating media. The theory of participating media has been derived to render effects such as smoke or clouds. However, participating media are also an appealing model for more general scenes containing surfaces. The motiva-

tion for this is two-fold: in rendering, *level of detail* methods describe techniques that simplify objects that are far away from the camera. For example, a 3D model of a tree might consist of millions of triangles. But if that tree is only visible on a few pixels on the screen, it is inefficient to load and render the complex original asset. On the other hand, differentiable rendering might be used to reconstruct scenes containing complex geometry. Again, a good example of this is vegetation, where directly reconstructing leaves as surfaces is highly non-convex.

For both level of detail and differentiable rendering, an approximation of the scene as a participating medium can be efficient. However, conventional participating media struggle to represent opaque surfaces. By default, we would get a representation that is prone to *light leaking* artifacts. We introduce a new volumetric appearance model based on *non-exponential* media, that can accurately represent fully opaque surfaces. We show that this improves both level of detail and differentiable rendering.

Systems [48, 49]. Differentiable rendering is not only a question of algorithms but also of the underlying system used for implementation. The author of this thesis was involved in the design and implementation of the Mitsuba 2 [48] differentiable renderer. The work on the path replay algorithm has further been tightly coupled to the development of Dr.Jit [49], a just-in-time compiler for differentiable rendering. This compiler is a key component of Mitsuba 3 [50], a recently released new version of the Mitsuba renderer. This thesis will not go into the implementation details of these systems, but all our methods have been implemented on top of these systems. We will touch on some of the general insights on differentiable rendering systems in Chapter 4.

1.5 Publication list

This thesis is primarily based on the following three publications [45, 46, 47]

- Delio Vicini, Sébastien Speierer, and Wenzel Jakob. 2021. Path Replay Backpropagation: Differentiating Light Paths using Constant Memory and Linear Time. In Transactions on Graphics (Proceedings of SIGGRAPH) 40(4).

- Delio Vicini, Sébastien Speierer, and Wenzel Jakob. 2022, Differentiable Signed Distance Function Rendering. In Transactions on Graphics (Proceedings of SIGGRAPH) 41(4).

- Delio Vicini, Wenzel Jakob, and Anton Kaplanyan. 2021. A Non-Exponential Transmittance Model for Volumetric Scene Representations. In Transactions on Graphics (Proceedings of SIGGRAPH) 40(4).

Each of these publications is covered in a dedicated chapter, and Chapter 4 partially unifies their background and related work sections. Over the course of the Ph.D., the author was involved in the following additional publications [13, 48, 49]:

- Delio Vicini, Vladlen Koltun, and Wenzel Jakob. 2019. A Learned Shape-Adaptive Subsurface Scattering Model. In Transactions on Graphics (Proceedings of SIGGRAPH) 38(4).
- Merlin Nimier-David*, Delio Vicini*, Tizian Zeltner, and Wenzel Jakob. 2019. Mitsuba 2: A Retargetable Forward and Inverse Renderer. In Transactions on Graphics (Proceedings of SIGGRAPH Asia) 38(6) (*joint first authors)
- Wenzel Jakob, Sébastien Speierer, Nicolas Roussel, Delio Vicini, 2022. Dr.Jit: A Just-In-Time Compiler for Differentiable Rendering. In Transactions on Graphics (Proceedings of SIGGRAPH) 41(4).

The Mitsuba 2 rendering system was instrumental in the development of all our work on differentiable rendering and the Dr.Jit compiler is a key component of the subsequent Mitsuba 3 system.

1.6 Organization of the thesis

We first cover the basics of Monte Carlo integration (Chapter 2) and rendering algorithms (Chapter 3). We then introduce background and related work on differentiable rendering in Chapter 4. The following chapters then each present the contributions of one publication: Chapter 5 is dedicated to the path replay backpropation algorithm. Chapter 6 presents our differentiable signed distance function rendering method. Chapter 7 introduces our non-exponential scene representation model. Chapter 8 concludes the thesis and gives an outlook on future directions.

2 | Monte Carlo Integration

In this chapter, we review the basics of probability theory and Monte Carlo integration. This sets the stage for the later chapters, in which Monte Carlo integration is used to render realistic images and estimate parameter gradients for optimization. At its core, physically-based differentiable rendering requires formulating Monte Carlo estimators for parameter gradients. The overview here is restricted to the concepts that are used in the later chapters. We do not discuss the foundations of measure theory or quasi-Monte Carlo and Markov chain Monte Carlo methods. A comprehensive review of Monte Carlo integration can for example be found in Eric Veach's thesis [51].

2.1 Probability theory

Monte Carlo integration estimates complex integrals using random sampling. This requires reasoning about random variables, their expected values and variance. We briefly recapitulate theoretical concepts to then develop Monte Carlo estimators.

2.1.1 Definitions

Probability space. We first define the *sample space* Ω as the set of all possible outcomes of a random process (e.g., the numbers 1 to 6 on a die). The *event space* $\mathcal{F}$ is defined as the σ-algebra of subsets of Ω. It is a collection of subsets that is closed under set complement and countable union operations. The *probability measure* $P\colon \mathcal{F} \to [0,1]$ is a function that satisfies:

1. $P(\varnothing) = 0$ and $P(\Omega) = 1$
2. $P\left(\bigcup_{i=1}^{\infty} A_i\right) = \sum_{i=1}^{\infty} P(A_i)$, for countable, pairwise disjoint events $A_1, A_2, \ldots \in \mathcal{F}$.

A *probability space* $(\Omega, \mathcal{F}, P)$ is the collection of sample space, event space and probability measure.

Random variables. A *random variable* models an outcome that depends on random events (e.g., the result of rolling a die). We distinguish between discrete and continuous random variables, depending on whether the values attained by the random variable are discrete or continuous. Formally, a random variable is a measurable function $X\colon \Omega \to E$, where E is a measurable space. This could for example be the real line $\mathbb{R}$ or the set

$\{1, 2, 3, 4, 5, 6\}$. We define $P(X \in A) \coloneqq P(\{\omega \in \Omega \colon X(\omega) \in A\})$ as a notation to express the probability of X taking a value in set A.

Probability density function. The *probability density function* (PDF) of a continuous random variable X is defined as a non-negative function p_X such that

$$P(X \in A) = \int_A p_X(\mathbf{x}) \, \mathrm{d}\mathbf{x}. \tag{2.1}$$

This definition is valid both for scalar and vectorial random variables. We generally use bold variable names (e.g., $\mathbf{x}$) for vectorial quantities and regular text (e.g., x) for scalar variables. Since $P(X \in \Omega) = 1$, the PDF integrates to one when integrated over the entire domain. We will use the notation $X \sim p_X$ to indicate that the distribution of the random variable X is described by the PDF p_X. For real-valued random variables, we can further define the *cumulative density function* (CDF) as

$$F_X(t) = P(X \leq t) = \int_{-\infty}^{t} p_X(x) \, \mathrm{d}x. \tag{2.2}$$

The PDF can also be defined as the derivative of the CDF and exists if and only if the CDF is differentiable almost everywhere. For our purposes, we assume that all our continuous random variables admit a valid PDF.

Another important concept is the transformation of random variables. Given a variable X and an invertible function T, the PDF of the transformed variable $Y = T(X)$ can be written as:

$$p_Y(\mathbf{y}) = \frac{p_X(\mathbf{x})}{|\mathbf{J}_T(\mathbf{x})|}, \tag{2.3}$$

where $|\mathbf{J}_T(\mathbf{x})|$ is the Jacobian determinant of the transformation T. For scalar functions, the Jacobian determinant is simply the derivative.

Probability mass function. The discrete analog of the PDF is the *probability mass function* (PMF):

$$p_X(x) = P(X = x) = P(\{\omega \in \Omega : X(\omega) = x\}). \tag{2.4}$$

The following sections will explain various concepts using continuous random variables, but many definitions trivially generalize to discrete variables. We will mostly deal with continuous random variables for the remainder of this thesis and hence do not further delve into details on discrete variables.

2.1.2 Expected value and variance

The *expected value* characterizes the average outcome of a random variable. In the continuous case, it is defined as an integral:

$$\mathbb{E}[X] = \int_{\Omega} \mathbf{x}\, p_X(\mathbf{x})\, \mathrm{d}\mathbf{x}. \tag{2.5}$$

As an integral, the expected value operator is *linear* and therefore satisfies $\mathbb{E}[aX + Y] = a\,\mathbb{E}[X] + \mathbb{E}[Y]$, where a is a scalar and X, Y are random variables. Using the expected value, we can define the *n-th moment* of a random variable as the value $\mathbb{E}[X^n]$. A related quantity is the *variance*:

$$\mathrm{Var}[X] = \mathbb{E}\left[(X - \mathbb{E}[X])^2\right] = \mathbb{E}\left[X^2\right] - \mathbb{E}[X]^2. \tag{2.6}$$

The variance is a measure of the spread of the random variable. In the context of Monte Carlo rendering, a higher variance indicates a higher error in the rendered image. The *standard deviation* σ is defined as the root of the variance:

$$\sigma(X) = \sqrt{\mathrm{Var}[X]}. \tag{2.7}$$

Two random variables X and Y might exhibit correlation in their outcomes. The *covariance* is a measure of that relation:

$$\mathrm{Cov}[X, Y] = \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])] = \mathbb{E}[XY] - \mathbb{E}[X]\,\mathbb{E}[Y]. \tag{2.8}$$

The covariance will be zero if X and Y are *independent*. The expectation of the product of two random variables can be expressed using the covariance:

$$\mathbb{E}[XY] = \mathbb{E}[X]\,\mathbb{E}[Y] + \mathrm{Cov}[X, Y]. \tag{2.9}$$

If X and Y are independent, the covariance term on the right disappears and the expectation of the product is simply the product of expectations.

2.1.3 Estimators

In the following, we will deal with *estimators* that are designed to estimate a certain quantity (e.g., an image pixel or a derivative). An estimator is a function $\hat{\theta}_N = \hat{\theta}(X_1, ..., X_N)$, where $X_1, ..., X_N$ are random samples. For an estimator $\hat{\theta}_N$ of a true quantity θ, the expected squared error decomposes into *bias* and variance:

$$\mathbb{E}\left[(\hat{\theta}_N - \theta)^2\right] = \underbrace{\mathbb{E}\left[\hat{\theta}_N - \theta\right]}_{\text{Bias}} + \underbrace{\mathrm{Var}\left[\hat{\theta}_N\right]}_{\text{Variance}}. \tag{2.10}$$

An estimator is called *unbiased* if its expected value matches the ground truth (i.e., $\mathbb{E}[\hat{\theta}_N] = \theta$). In that case, the bias is zero and the error only consists of variance. In physically-based rendering, our goal is usually to construct unbiased estimators that produce the correct output image up to some amount of variance. The variance of the estimator is visible as per-pixel noise, which gradually disappears as the number of samples N is increased.

An estimator is called *consistent* if its value converges to the ground truth as the number of samples is increased, i.e., $\lim_{N\to\infty} \hat{\theta}_N = \theta$. While we usually strive to construct unbiased estimators, biased consistent estimators are also frequently encountered in rendering. In some cases, allowing a non-zero bias enables reducing the variance such that the overall error is lower. An example of this are image-based *denoising* algorithms [52], which reduce the variance of the rendered image by applying a non-linear filter to the rendered image. The noise reduction produces a biased, consistent estimator of the image that has less visible error than an unbiased estimator. *De-biasing* methods eliminate bias of certain consistent estimators at the cost of additional variance and computation time [53]. Technically, it is also possible to construct unbiased estimators that are not consistent, but these are of little practical use.

2.2 Monte Carlo integration

Rendering requires the numerical estimation of high-dimensional integrals such as

$$I = \int_{\mathcal{X}} f(\mathbf{x}) \, \mathrm{d}\mu(\mathbf{x}). \tag{2.11}$$

Here, $\mathcal{X}$ denotes the integration domain, f a scalar-valued integrand and μ a measure on $\mathcal{X}$. We will mostly omit the measure and simply write $\mathrm{d}\mathbf{x}$ instead of $\mathrm{d}\mu(\mathbf{x})$.

It is generally not possible to compute the value of such an integral analytically. Deterministic *quadrature methods* evaluate the integrand at regularly spaced sample locations and compute a piecewise polynomial approximation that can then be integrated analytically. This can work well for low-dimensional integrals, but suffers from the curse of dimensionality. The number of evaluations grows exponentially as the dimensionality increases. Rendering instead relies on *Monte Carlo integration*, in which the integral is approximated by randomly sampling the integration domain. More specifically, we can estimate the value of I as:

$$I \approx \hat{I}_N = \frac{1}{N} \sum_{i=1}^{N} \frac{f(\mathbf{x}_i)}{p(\mathbf{x}_i)}, \tag{2.12}$$

where $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N \in \mathcal{X}$ are independent samples drawn from a distribution with the PDF $p(\mathbf{x})$. In the simplest case, $p(\mathbf{x})$ could be the uniform distribution over the domain $\mathcal{X}$. More sophisticated sampling strategies are needed to obtain efficient estimators with good convergence characteristics as $N \to \infty$. If $p(\mathbf{x}) > 0$ for all $\mathbf{x} \in \mathcal{X}$ where $f(\mathbf{x}) \neq 0$, then the Monte Carlo estimator is unbiased. We can show that indeed $\mathbb{E}\left[\hat{I}_N\right] = I$ by using the linearity and definition of the expected value operator:

$$\begin{aligned}\mathbb{E}\left[\hat{I}_N\right] &= \mathbb{E}\left[\frac{1}{N}\sum_{i=1}^{N}\frac{f(\mathbf{x}_i)}{p(\mathbf{x}_i)}\right] = \frac{1}{N}\sum_{i=1}^{N}\mathbb{E}\left[\frac{f(\mathbf{x}_i)}{p(\mathbf{x}_i)}\right] = \frac{1}{N}\sum_{i=1}^{N}\int_{\mathcal{X}}\frac{f(\mathbf{x})}{p(\mathbf{x})}p(\mathbf{x})\,\mathrm{d}\mathbf{x} \\ &= \frac{1}{N}\sum_{i=1}^{N}\int_{\mathcal{X}} f(\mathbf{x})\,\mathrm{d}\mathbf{x} = \frac{1}{N}\sum_{i=1}^{N} I = I. \end{aligned} \tag{2.13}$$

Additionally, the law of large numbers guarantees that the average of independent and identically distributed random variables converges to their mean as the number of samples goes to infinity. Therefore, the Monte Carlo estimator is consistent. Regardless of the dimensionality of $\mathcal{X}$, the convergence rate of the root mean squared error of this estimator is $O(N^{-1/2})$.

2.3 Variance reduction

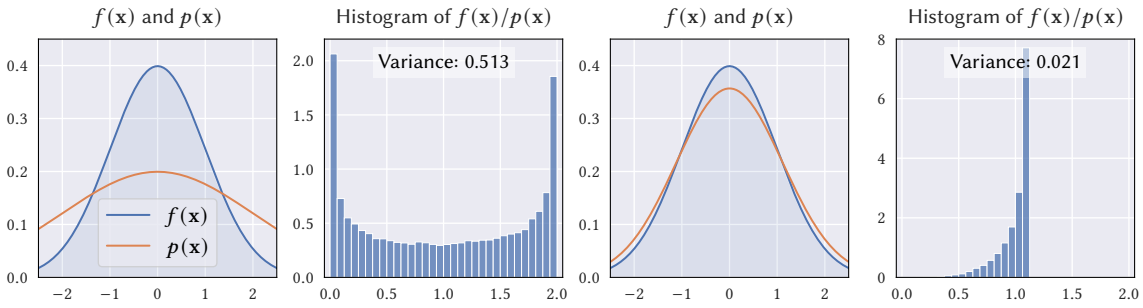
While Monte Carlo integration is unbiased, it suffers from error in the form of variance. Applications of Monte Carlo integration usually rely on a range of variance reduction strategies. Common variance reduction methods form the basis of all modern rendering algorithms. We introduce the main variance reduction techniques that are referenced in the remainder of the thesis. We will not discuss additional techniques such as low discrepancy sequences, stratified sampling and adaptive sampling.

2.3.1 Importance sampling

A key factor determining the variance of a Monte Carlo estimator is the used sampling distribution. The ideal sampling distribution is proportional to the integrand and has the following PDF:

$$p(\mathbf{x}) = \frac{1}{\int_{\mathcal{X}} f(\mathbf{y})\,\mathrm{d}\mathbf{y}} f(\mathbf{x}). \tag{2.14}$$

The normalization constant $1/\int_{\mathcal{X}} f(\mathbf{y})\,\mathrm{d}\mathbf{y}$ ensures that the PDF integrates to 1. Using this sampling distribution, each individual Monte Carlo sample $\mathbf{x}_i$ exactly computes I and



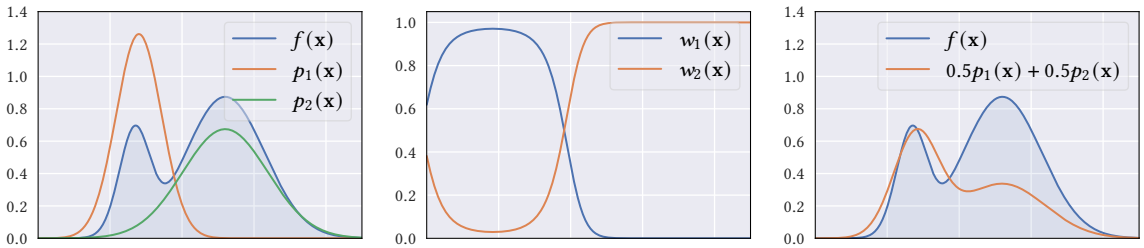
(a) Sampling strategy 1 (b) Sampling strategy 2

Figure 2.1: We visualize an example integrand $f(\mathbf{x})$ and the used importance sampling distribution $p(\mathbf{x})$. Depending on how closely the importance sampling distribution matches $f(\mathbf{x})$, we get a different distribution of Monte Carlo sample values $f(\mathbf{x})/p(\mathbf{x})$. The strategy in **(a)** only loosely matches the integrand, while the strategy in **(b)** is close to the optimal strategy. It achieves a 24× lower variance than the less optimal sampling distribution.

the variance of the estimator is therefore 0:

$$\frac{f(\mathbf{x}_i)}{p(\mathbf{x}_i)} = \int_{\mathcal{X}} f(\mathbf{y})\,\mathrm{d}\mathbf{y} f(\mathbf{x}_i)/f(\mathbf{x}_i) = \int_{\mathcal{X}} f(\mathbf{y})\,\mathrm{d}\mathbf{y} = I. \tag{2.15}$$

In practice, we cannot use this ideal distribution, as constructing it would require the solution of the original integration problem. Nevertheless, this ideal distribution provides the important intuition that the variance of a Monte Carlo estimator depends on how closely the PDF follows the shape of the integrand. A simple numerical example of this is shown in Figure 2.1, where we compare two different sampling distributions on the same integrand. By more closely matching the importance sampling distribution to the integrand, the variance of individual samples can be reduced significantly. Often there is a tradeoff between using a more complex sampling strategy and simply using a larger number of samples. An advanced sampling method might be too computationally expensive to reduce error compared to a simpler method at equal time. A significant fraction of rendering research is aimed at constructing efficient sampling algorithms that more closely approximate the ideal strategy. For example, bidirectional path tracing [54, 55] improves importance sampling of indirect illumination for challenging scenes and path guiding methods [56, 57, 58] build scene-specific sampling strategies on the fly. Visible normal sampling [59] reduces variance for microfacet surface appearance models. As we will see in Chapter 3, rendering algorithms employ a combination of local sampling strategies to achieve a high-quality image estimate.



(a) $f(\mathbf{x})$ and $p(\mathbf{x})$ (b) MIS weights (c) Effective PDF

Figure 2.2: **(a)** We visualize an example integrand $f(\mathbf{x})$ and two sampling strategies $p_1(\mathbf{x})$ and $p_2(\mathbf{x})$. Plot **(b)** shows the balance heuristic MIS weights and **(c)** the effective resulting PDF, which is simply the average of the two sampling strategies. It matches the integrand $f(\mathbf{x})$ more closely than each individual sampling strategy.

2.3.2 Multiple importance sampling

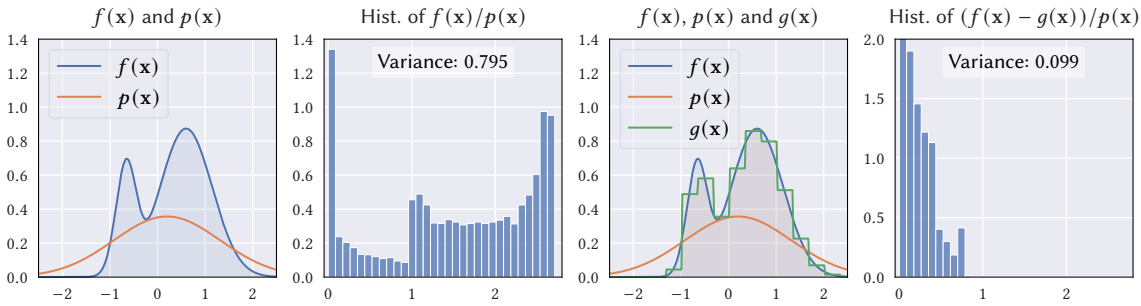
It can be difficult to formulate practical algorithms that perfectly sample all parts of the integrand. However, for some problems, we can come up with strategies that focus on different terms in the integrand. *Multiple importance sampling (MIS)* [60] can then be used to combine different strategies to form a unified estimator. Given M sampling strategies, we can define weighting functions $w_1, w_2, ..., w_M \colon \mathcal{X} \to \mathbb{R}$ and formulate a weighted Monte Carlo estimator:

$$I = \int_{\mathcal{X}} f(\mathbf{x})\,\mathrm{d}\mathbf{x} = \sum_{j=1}^{M} \int_{\mathcal{X}} w_j(\mathbf{x}) f(\mathbf{x})\,\mathrm{d}\mathbf{x} \approx \frac{1}{N} \sum_{j=1}^{M} \sum_{i=1}^{N} w_j(\mathbf{x}_{ij}) \frac{f(\mathbf{x}_{ij})}{p_j(\mathbf{x}_{ij})}, \tag{2.16}$$

where $\mathbf{x}_{ij} \sim p_j$. This estimator assumes that we draw N samples using each strategy. The weights can be arbitrary, but need to form a pointwise partition of unity, i.e., $\sum_{j=1}^{M} w_j(\mathbf{x}) = 1$ for each $\mathbf{x}$ where $f(\mathbf{x}) \neq 0$. If non-negative weights are assumed, the optimal weights are given by the *balance heuristic* [60]

$$w_j(\mathbf{x}) = \frac{p_j(\mathbf{x})}{\sum_{k=1}^{M} p_k(\mathbf{x})}. \tag{2.17}$$

Effectively, using the balance heuristic amounts to sampling using the average PDF, while ensuring that all sampling strategies are being used equally often. A simple example is shown in Figure 2.2, where the balance heuristic leads to an effective PDF that more closely matches the integrand than the individual sampling strategies. The variance can sometimes be improved by using the *power heuristic*, which raises the individual PDF values to a power q when computing the weights. Interestingly, the variance can be further reduced by allowing negative weights [61].



(a) Without control variate (b) With control variate g

Figure 2.3: We visualize an example integrand $f(\mathbf{x})$ and the used importance sampling distribution $p(\mathbf{x})$. We compare the distribution of Monte Carlo sample values both without **(a)** and with **(b)** a piecewise constant control variate g. When using the control variate, the Monte Carlo integration only needs to estimate the difference between f and g. The integral of g can trivially be computed analytically.

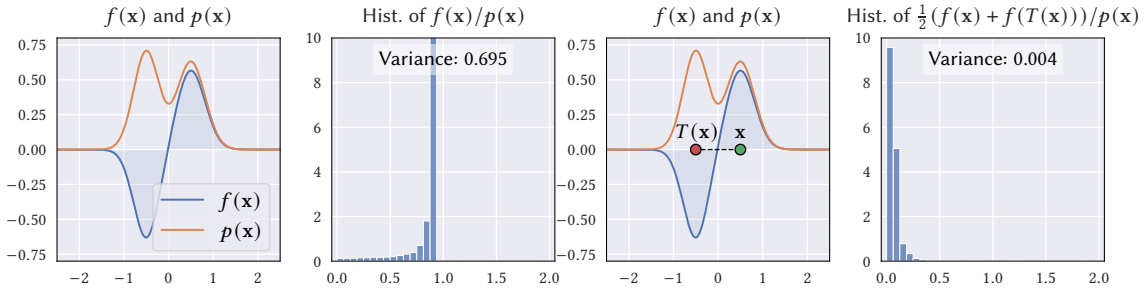
2.3.3 Control variates

Importance sampling and MIS are the main variance reduction techniques used for Monte Carlo rendering. Another common method are *control variates*. The idea is to use an auxiliary integrand g to reformulate the problem as follows:

$$I = \int_{\mathcal{X}} f(\mathbf{x})\,\mathrm{d}\mathbf{x} = \int_{\mathcal{X}} f(\mathbf{x}) - \alpha g(\mathbf{x})\,\mathrm{d}\mathbf{x} + \alpha \int_{\mathcal{X}} g(\mathbf{x})\,\mathrm{d}\mathbf{x}, \tag{2.18}$$

where α is a scalar weight. This formulation is useful if the integral over g is either known analytically or can be estimated more easily than the original integral. The integral over the difference of f and g can then be easier to solve than the original problem. Ideally, $g(\mathbf{x}) = f(\mathbf{x}) + C$, where C is some constant. In that case, the difference integral can be estimated with zero variance using a single sample. Similar to importance sampling, we cannot use this ideal g without solving the original problem. However, as in importance sampling, we can already achieve a significant variance reduction if g approximates f in some way. The optimal weight α can be computed by considering the covariance between evaluations of f and g in the difference integral [62].

In Figure 2.3, we show the effect of using a piecewise constant control variate on a 1D problem. In that case, the integral of g can be computed analytically. In rendering, there has been work on constructing scene-specific control variates using a binary tree [56], piecewise polynomials [63] or neural networks [64]. Another use of control variates are *gradient-domain* rendering algorithms [62, 65, 66], which use neighboring pixels as control variates to reduce the noise level of the final image.



(a) Without antithetic sampling (b) With antithetic sampling

Figure 2.4: For a nearly symmetrical integrand $f(\mathbf{x})$, antithetic sampling can drastically reduce the variance. In this example here, we set $T(\mathbf{x}) = -\mathbf{x}$, as the integrand exhibits a symmetry around the origin. Each sample $\mathbf{x}$ (green point) then produces a corresponding sample $-\mathbf{x}$ (red point). The values $f(\mathbf{x})$ and $f(-\mathbf{x})$ are highly correlated, leading to most Monte Carlo samples having a value close to zero. Both with and without antithetic sampling we use the same sampling distribution $p(\mathbf{x})$.

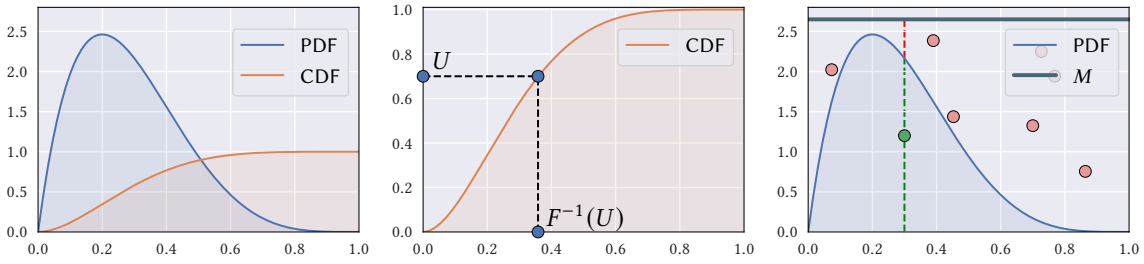
2.3.4 Antithetic sampling

A closely related technique is *antithetic sampling*, which reduces variance by correlating samples. This is for example useful for integrands that consist of a positive and negative part, that overall integrate to a value that is small relative to the absolute magnitude of the integrand (or even zero). In this case, the variance can be reduced by introducing correlation between some of the Monte Carlo samples:

$$I = \int_{\mathcal{X}} f(\mathbf{x})\,\mathrm{d}\mathbf{x} \approx \frac{1}{N}\sum_{i=1}^{N/2} \frac{f(\mathbf{x}_i) + f(T(\mathbf{x}_i))}{p(\mathbf{x})}, \tag{2.19}$$

where $T\colon \mathcal{X} \to \mathcal{X}$ is a transform that maps a sample to another location inside the domain. T should be designed to maximize *anticorrelation* between $f(\mathbf{x})$ and $f(T(\mathbf{x}))$. Ideally, $|f(\mathbf{x}_i) + f(T(\mathbf{x}_i))|$ is then smaller in magnitude than f itself, reducing the overall variance of the estimator. Equation 2.19 assumes that T does not distort the domain, i.e., the Jacobian determinant satisfies $|\mathbf{J}_T| = 1$. If this is not the case, $f(T(\mathbf{x}_i))$ needs to be multiplied by this Jacobian determinant to account for the change in sampling density. Figure 2.4 shows a simple example of antithetic sampling applied to a nearly symmetrical integrand.

Antithetic sampling can be particularly useful for differentiable rendering, where derivatives of integrands may exhibit symmetries that can be exploited to reduce variance [67, 68, 69]. For example, the positional derivative of a Gaussian PDF will exhibit a similar symmetry as the example integrand in Figure 2.4.



(a) PDF and CDF (b) Inverse transform sampling (c) Rejection sampling

Figure 2.5: Given a probability distribution in 1D **(a)**, we can generate samples using inverse transform sampling **(b)** or rejection sampling **(c)**. In rejection sampling, we repeatedly draw samples until we generate a point that lies within the blue area under the PDF. All other samples (red) are discarded.

2.4 Sampling random variables

To use Monte Carlo integration, we need to be able to generate random numbers following a certain target distribution. Moreover, the sample generation algorithms must be computationally efficient.

Pseudo-random generators. While true randomness can only be achieved by dedicated hardware, pseudo-random generators such as the *permuted congruential generator (PCG)* [70] suffice for rendering applications. PCG and similar random number generators deterministically generate a sequence of random numbers from a seed number. The generated numbers statistically closely resemble true random numbers. All commonly used sampling methods in rendering rely on first generating pseudo-random samples uniformly in $[0, 1]$. These samples can then be used to generate random variables following a desired distribution.

Inverse transform sampling. For many practically relevant distributions in rendering, we can generate samples using *inverse transform sampling* [71]. The idea is to leverage the distribution's CDF to transform uniformly distributed samples to the target distribution. While the CDF is generally not invertible, we can define $F_X^{-1}(u) := \min\{t \in \mathbb{R} : F_X(t) \leq u\}$. This "inverse" CDF is useful to generate random samples following the distribution defining the CDF. We first generate a uniformly distributed random variable U in $[0, 1]$. Then, $X = F_X^{-1}(U)$ is a random variable that is distributed according to p_X. This sampling method is illustrated in Figure 2.5b.

If F is invertible, the correctness proof of this method is a direct application of the

density change formula (Equation 2.3):

$$p(x) = p(F_X^{-1}(u)) = \frac{p_U(u)}{\left(F_X^{-1}\right)'(u)} = 1 \cdot p_X(F_X^{-1}(u)) = p_X(x), \tag{2.20}$$

where we used the inverse function derivative rule $\left(F_X^{-1}\right)'(u) = 1/F_X'(F_X^{-1}(u)) = 1/p_X(F_X^{-1}(u))$.

This concept can be generalized to higher dimensions by using the marginal and conditional distributions. For example, a 2D distribution $p(x, y)$ could be sampled by first sampling $X \sim p(x)$, where $p(x) = \int_y p(x, y)\,\mathrm{d}y$. In a second step, Y can be sampled according to the conditional distribution $p(y|x) = p(x,y)/p(x)$.

Rejection sampling. If the CDF cannot be inverted efficiently, the solution is oftentimes to use *rejection sampling*. This method repeatedly draws candidate samples, until one is accepted. For a real-valued continuous distribution $p(x)$ over an interval $[a, b]$, we define M to be an upper bound on the values of $p(x)$. The rejection sampling algorithm then works as follows: we first sample a value X uniformly in the interval $[a, b]$. Then, we generate a random number Y in $[0, M]$ uniformly and check if $Y < p(x)$. If this is the case, we accept the sample and otherwise we start again by sampling a value X. We illustrate this method in Figure 2.5c. A similar scheme can also be used to sample points inside subsets of Euclidean space. For example, we can generate uniformly distributed points inside a disk by first drawing a sample from an enclosing rectangle and accepting it only if it lies within the disk. Rejection sampling is typically less efficient than inverse transform sampling and is only used if no other method is available.

3 | Light Transport and Rendering Algorithms

Rendering and inverse rendering both rely on simulating light transport in a virtual scene. We will in the following define basic quantities and equations that describe steady-state light transport. This means we do not attempt to model temporal effects such as temperature changes affecting infrared radiation. We restrict the discussion to the large range of phenomena described by *geometric optics*, and ignore the wave nature of light (e.g., interference or diffraction). The chapter concludes with a brief overview of the Monte Carlo rendering algorithms that we use in the remainder of the thesis.

3.1 Radiometry

To define how light interacts with a scene, we first need to establish relevant units and measures of electromagnetic radiation. The study of these quantities is called *radiometry*. In geometric optics, we can think of light as consisting of photon particles traveling at light speed along trajectories in 3D space. A photon has a certain wavelength λ and can either be emitted, scattered or absorbed. The pixel response of a (virtual) camera sensor depends on the number of incident photons and their energy as determined by the wavelength. The energy of a single photon is

$$E = \frac{hc}{\lambda}, \tag{3.1}$$

where $h = 6.626 \times 10^{-34}\,\mathrm{m^2 kg/s}$ is Planck's constant, $c = 299\,792\,458\,\mathrm{m/s}$ the speed of light and λ the photon's wavelength (in meters). The human visual system is sensitive to wavelengths from 380 nm to 700 nm.

While one could describe light transport by modeling individual photon events, we do not need to work at that granularity for the types of problems discussed in this thesis. The involved numbers of photons are large enough that we can safely assume their total energy to be a continuous quantity. For example, a typical camera sensor collects around 10^5 photons per pixel [72]. We do not discuss applications such as single-photon imaging [73], where photons need to be modeled individually.

Energy We first define the *total energy* of photons in a region of space (e.g., emitted by a surface $S \subset \mathbb{R}^3$) in a time interval $[0, t]$ as $Q(t)$. The units of energy are joules J.

Radiant flux. Given the total energy, we define the *radiant flux* (or power) as its infinitesimal temporal change:

$$\Phi(t) = \frac{\mathrm{d}Q(t)}{\mathrm{d}t}. \tag{3.2}$$

The flux is measured in watts W = J/s. When rendering a scene in steady state, we are concerned about flux rather than the total amount of energy. For example, we specify the intensity of a light source in watts instead of the total amount of energy emitted over a time interval. Since we are interested in static scenes, we will drop the explicit dependency of Φ on the time t.

Irradiance. For a given surface S, the *irradiance* describes the density of incident flux per area and is defined as

$$E(\mathbf{x}) = \frac{\mathrm{d}\Phi(\mathbf{x})}{\mathrm{d}A(\mathbf{x})}. \tag{3.3}$$

The units of irradiance are $\mathrm{W/m^2}$. Irradiance measures incident flux density and the corresponding outgoing quantity is called *radiant exitance*. The definition here can be understood as a derivative of the flux with respect to the area measure $\mathrm{d}A$. More precisely, this derivative is the *Radon-Nikodym* derivative of measures. This means that $E(\mathbf{x})$ is a function that integrates to the flux Φ when using the area measure on the surface S:

$$\Phi(\mathbf{x}) = \int_S E(\mathbf{x})\, \mathrm{d}A(\mathbf{x}). \tag{3.4}$$

The Radon-Nikodym derivative is uniquely defined up to sets of measure zero. A precise measure-theoretic derivation of all radiometric quantities is not required for the methods introduced in this thesis and we refer to Eric Veach's Ph.D. thesis for details [51, Chapter 3].

Radiance. The central quantity of interest in rendering is *radiance*. Radiance is defined as flux per solid angle per projected area. A *solid angle* measures the area subtended by an object on the unit sphere $\mathcal{S}^2$. Its units are steradians sr and it is the two-dimensional analog of one-dimensional angles. Radiance is measured per *projected area*, which is a small surface patch perpendicular to the direction of interest. Formally, the radiance L is defined as:

$$L(\mathbf{x}, \boldsymbol{\omega}) = \frac{\mathrm{d}^2\Phi(\mathbf{x}, \boldsymbol{\omega})}{\mathrm{d}\sigma(\boldsymbol{\omega})\, \mathrm{d}A^{\perp}(\mathbf{x})}. \tag{3.5}$$

The units of radiance are watts per steradian per square meter $\mathrm{W/sr\,m^2}$. The projected area measure relates to the standard area measure on a surface as

$$\mathrm{d}A^{\perp}(\mathbf{x}) \coloneqq |\boldsymbol{\omega} \cdot \mathbf{n}|\, \mathrm{d}A(\mathbf{x}) = |\cos\theta|\, \mathrm{d}A(\mathbf{x}), \tag{3.6}$$

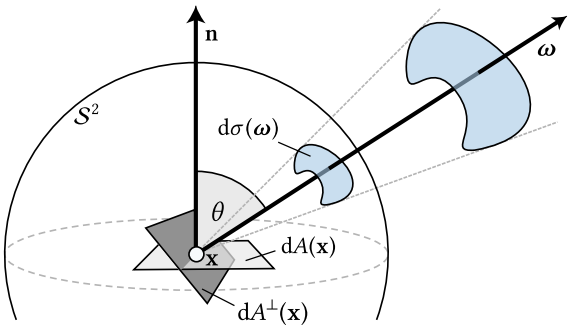


Figure 3.1: Illustration of the terms used in the definition of radiance.

where $\mathbf{n}$ is the surface normal and θ the angle between the surface normal and $\boldsymbol{\omega}$. The cosine term $|\cos\theta| = |\boldsymbol{\omega} \cdot \mathbf{n}|$ accounts for the foreshortening effect: incident light at a grazing angle will hit a larger surface patch. See Figure 3.1 for an illustration of the various terms used in the definition of radiance.

The cosine term is sometimes absorbed into the solid angle measure instead of the area measure, forming the *projected solid angle* measure:

$$\mathrm{d}\sigma^\perp(\boldsymbol{\omega}) \coloneqq |\boldsymbol{\omega} \cdot \mathbf{n}| \, \mathrm{d}\sigma(\boldsymbol{\omega}). \tag{3.7}$$

To avoid notational clutter, we will often use the more compact notation $\mathrm{d}\boldsymbol{\omega}^\perp \coloneqq \mathrm{d}\sigma^\perp(\boldsymbol{\omega})$. We will also distinguish between *incident* radiance L_i and *outgoing* radiance L_o. For a point $\mathbf{x}$ that is not on a surface (or inside of a participating medium), it holds that $L_i(\mathbf{x}, \boldsymbol{\omega}) = L_o(\mathbf{x}, -\boldsymbol{\omega})$. In other words, radiance is constant along a ray through empty space. This property does not hold on surfaces or inside participating media, as then the incident radiance might be reflected or absorbed.

Spectral radiance. We so far implicitly assumed the radiance L to integrate over all wavelengths. However, to render RGB images, we need to handle spectrally resolved quantities. The most important one is *spectral radiance*:

$$L(\mathbf{x}, \boldsymbol{\omega}, \lambda) = \frac{\mathrm{d}^3\Phi(\mathbf{x}, \boldsymbol{\omega}, \lambda)}{\mathrm{d}\sigma(\boldsymbol{\omega}) \, \mathrm{d}A^\perp(\mathbf{x}) \, \mathrm{d}\lambda}, \tag{3.8}$$

where we assumed Φ to be the spectral energy of incident illumination. We will typically not write out the wavelength dependency of radiance for conciseness. But unless stated otherwise, any use of radiance L in the following is assumed to be spectral radiance.

3.2 Image formation

Measurement equation. We will now specify the equations that govern image formation. While the presented theory describes arbitrary radiance measurement processes, the following discussion assumes that our goal is to ultimately compute (RGB) intensity values of a digital image consisting of (square) pixels. This image is captured by a *virtual camera* that is placed in a virtual scene. Given a virtual scene, the intensity of a pixel j can be written using the *measurement equation*:

$$I_j = \int_{\mathcal{M}} \int_{\mathcal{S}^2} W_e^{(j)}(\mathbf{x}, \boldsymbol{\omega}) L_i(\mathbf{x}, \boldsymbol{\omega}) \, \mathrm{d}\sigma^{\perp}(\boldsymbol{\omega}) \, \mathrm{d}A(\mathbf{x}), \tag{3.9}$$

where $\mathcal{M} \subset \mathbb{R}^3$ is the union of all scene surfaces (including the sensor itself), $\mathcal{S}^2$ the unit sphere of directions, L_i the incident radiance and $W_e^{(j)}(\mathbf{x}, \boldsymbol{\omega})$ is the *sensor importance function*. Intuitively, the measurement equation computes a weighted average of incident radiance to produce the intensity of a pixel j. The importance function $W_e^{(j)}(\mathbf{x}, \boldsymbol{\omega})$ models the (spectral) response of the virtual sensor's pixel j to radiance arriving at a given location $\mathbf{x}$ from the direction $\boldsymbol{\omega}$. Its specific form depends on the camera model that is used (e.g., perspective or orthographic camera) and its position and orientation. Moreover, it depends on the *pixel filter*, which controls the response of a single pixel to radiance arriving at the sensor. A box pixel filter implies that a pixel only accumulates radiance incident on its physical area, whereas a continuous filter with a larger support (e.g., a Gaussian) also considers incident radiance slightly outside the pixel. The latter is often preferred, as it leads to a smoother image with less visible aliasing.

Camera models. The most commonly used camera model is that of a perspective pinhole camera. In the real world, a pinhole camera is a primitive, lensless camera that contains a film (or digital sensor) opposite a tiny hole in an otherwise opaque box. The image is produced by light passing through the opening and hitting the camera film. In rendering, the idealized pinhole camera models a setting where the pinhole itself is infinitesimally small and hence produces a perfectly sharp image of the scene. A better approximation of a real lens-based camera is the *thin lens* model, which assumes a camera lens consisting of a single lens element focusing incident illumination onto the sensor, which enables modeling of defocus blur. More advanced camera models simulate multi-element optics [74].

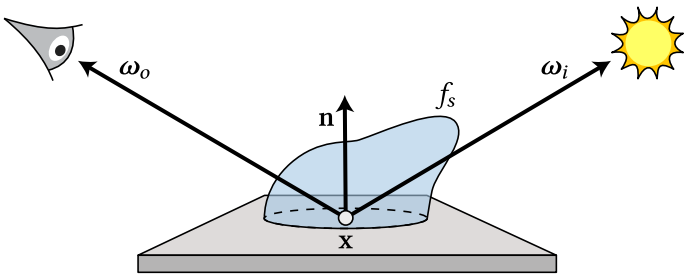


Figure 3.2: Illustration of light scattering on a surface. Incident radiance arriving from direction ω_i is attenuated by the BSDF f_s before being reflected in direction ω_o. The BSDF might also depend on the surface location $\mathbf{x}$.

3.3 Surface light transport

The measurement equation depends on the unknown incident radiance. In this section, we describe how radiance relates to surface reflectance models and emission. This will lead to an integral formulation of light transport, that allows estimating the pixel colors using Monte Carlo integration. In this section, we assume the scene to consist of surfaces in a vacuum. Objects are made up of infinitesimally thin "shells" (e.g., triangle meshes) and light travels uninterrupted through free space. We will discuss participating media and volumetric scattering in Section 3.4.

3.3.1 Surface light scattering

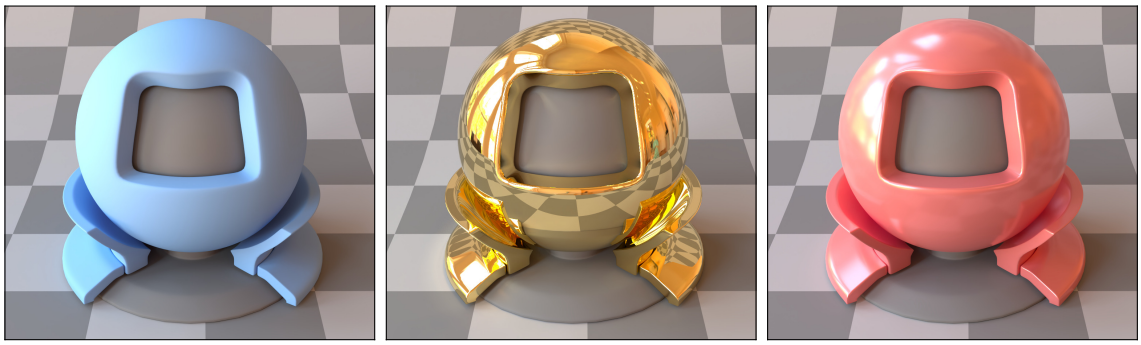
Surface scattering converts the radiance incident on a surface to outgoing radiance. The outgoing radiance L_o linearly depends on the incident radiance L_i arriving from all possible incident directions:

$$L_o(\mathbf{x}, \boldsymbol{\omega}_o) = \int_{S^2} f_s(\mathbf{x}, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i) L_i(\mathbf{x}, \boldsymbol{\omega}_i) \, \mathrm{d}\sigma^\perp(\boldsymbol{\omega}_i). \tag{3.10}$$

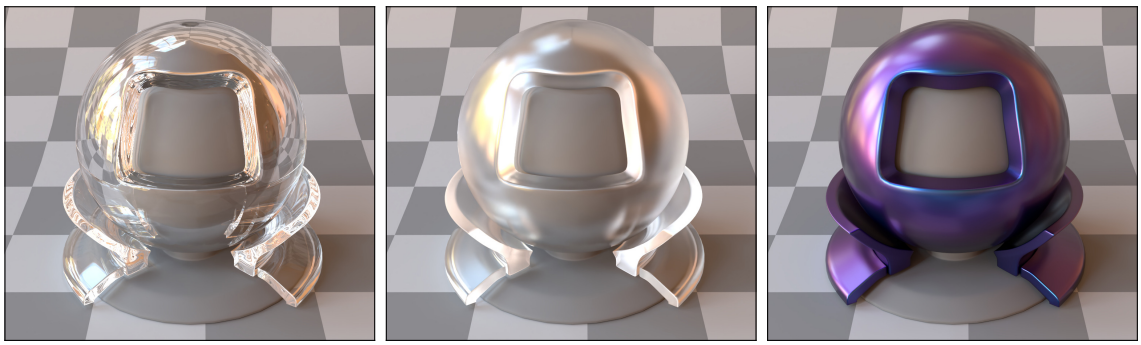
Figure 3.2 illustrates some of the terms used in this equation. The *bidirectional scattering distribution function (BSDF)* f_s represents the constant of proportionality between incident and outgoing radiance. It can equivalently be defined as the derivative of outgoing radiance divided by the incident radiance:

$$f_s(\mathbf{x}, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i) = \frac{\mathrm{d}L_o(\mathbf{x}, \boldsymbol{\omega}_o)}{L_i(\mathbf{x}, \boldsymbol{\omega}_i) \, \mathrm{d}\sigma^\perp(\boldsymbol{\omega}_i)}. \tag{3.11}$$

The BSDF captures both absorption and the directional distribution of scattered light. These effects can be spatially, directionally and spectrally varying. For example, a red object will mostly reflect at wavelengths that we perceive as red while absorbing others.



(a) Diffuse BSDF (b) Conductor BSDF (c) Disney principled BSDF



(d) Dielectric BSDF (e) Rough dielectric BSDF (f) Measured BSDF

Figure 3.3: Renderings of a test object with different analytic BSDFs **(a)**–**(e)** and an example of a complex BSDF measured using a goniophotometer **(f)**.

For our rendering to be physically plausible, we require BSDFs to be non-negative functions that satisfy *energy conservation*: the total amount of reflected light can never exceed the amount of incident light. Formally, we can write this condition as

$$\int_{S^2} f_s(\mathbf{x}, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i)\, \mathrm{d}\sigma^{\perp}(\boldsymbol{\omega}_i) \leq 1. \tag{3.12}$$

Additionally, if we only consider the reflective part of the BSDF (i.e., the hemisphere of directions on one side of the surface), a physically-based BSDF is *reciprocal*, which means that the incident and outgoing direction can be swapped: $f_s(\mathbf{x}, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i) = f_s(\mathbf{x}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o)$. Somewhat counterintuitively, the reciprocity does not hold for refractions, where $\boldsymbol{\omega}_o$ and $\boldsymbol{\omega}_i$ are on different sides of the surface [51].

3.3.2 BSDF models

Different BSDFs are used to model the range of surface appearances that are present in the real world. For example, the BSDF for a metal is different from the BSDF of glass. A digital replica of a real material can be acquired by measuring its reflectance spectrum

for all combinations of incident and outgoing directions. A *goniophotometer* is a device that does exactly that by moving a sensor and light source around a given material sample. Depending on the desired accuracy of the reconstruction, this process can be quite time-consuming. Even a state-of-the-art commercial goniophotometer might need several hours per material [75]. The result of the measurement is a tabulated BSDF that replicates the appearance of a material sample.

Many common real-world materials can be approximated quite well by analytic models. For most rendering applications, such models are more useful than measured materials, as they offer direct (artistic) control over material appearances using a small number of parameters. For inverse problems, parametric BSDF models help to reduce the number of parameters that need to be considered in the optimization. In many inverse rendering applications, we simply do not have enough observations to recover a tabulated BSDF. Figure 3.3 shows a test object rendered using a few different analytic BSDFs and also shows an example of a more complex measured material.

Diffuse BSDF. For example, the *diffuse* BSDF (Figure 3.3a) assumes that light is reflected uniformly in the hemisphere above the surface. This is an idealized BSDF, since no real material has this perfectly uniform behavior. However, it is a plausible approximation for some matte objects and it has an extremely simple closed form:

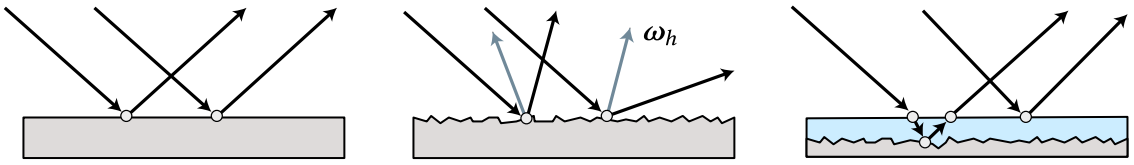
$$f_s(\mathbf{x}, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i) = \frac{1}{\pi}\rho, \tag{3.13}$$

where ρ is the spectrally varying surface albedo and the division by π ensures energy conservation.

Specular BSDFs. Another important class of materials are *smooth specular* BSDFs, which reflect or refract light into a discrete set of directions. Such BSDFs approximate the appearance of polished metal (Figure 3.3b) or glass (Figure 3.3d). The BSDF then becomes a combination of weighted Dirac delta functions. For example, an idealized mirror has the following BSDF

$$f_s(\mathbf{x}, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i) = \frac{\delta(\boldsymbol{\omega}_i - \boldsymbol{\omega}_r)}{|\cos\theta_i|}, \tag{3.14}$$

where $\boldsymbol{\omega}_r = 2\langle \mathbf{n}, \boldsymbol{\omega}_o \rangle \mathbf{n} - \boldsymbol{\omega}_o$ is the direction $\boldsymbol{\omega}_o$ reflected on a surface of normal $\mathbf{n}$. The delta function $\delta(\boldsymbol{\omega}_i - \boldsymbol{\omega}_r)$ is 1 if the incident direction matches the reflected direction and 0 otherwise. The division by the cosine term is necessary to cancel out the cosine term from the projected solid angle measure. For dielectric surfaces such as glass, the



(a) Smooth specular BSDF (b) Microfacet BSDF (c) Layered BSDF

Figure 3.4: Illustration of different BSDF models. A smooth specular BSDF **(a)** reflects light according to the surface normal. A microfacet BSDF **(b)** models reflectance due to microscopic surface imperfections. A layered BSDF **(c)** might consist of a rough base surface with a smooth dielectric coating and has to account for light interacting with both layers.

BSDF both reflects and refracts incident illumination at a ratio that is determined by the *Fresnel equations*. For metal surfaces, the BSDF will include Fresnel terms computed from a material-dependent complex index of refraction. We will not go into details of these equations here.

Microfacet theory In reality, most surfaces are not perfect mirrors or smooth glass, but exhibit some kind of microscopic imperfections that determine their appearance. For a specular surface, imperfections cause light to be reflected in a non-trivial set of directions, instead of just one discrete direction. We illustrate this effect in Figure 3.4b. *Microfacet theory* [76, 77, 78, 79, 80] is a statistical framework to model such imperfections. It formalizes the idea of surface imperfections by assuming the surface to be made up of small facets, so-called microfacets, with a certain statistical distribution of surface normals. The *normal distribution function* (NDF) characterizes this distribution of surface normals and controls the *roughness* of the surface. If all surface normals point in the same direction, we are back to a smooth surface. If the normals follow a uniform distribution over the hemisphere, we get a rough appearance that resembles a diffuse surface. Microfacet BSDFs are convenient, as they allow to parameterize surface appearances of varying roughness.

During rendering, microfacet BSDFs do not need to instantiate actual microfacets and instead directly consider the aggregate effect of a certain NDF. A classic microfacet BSDF is the Torrance-Sparrow [77] model:

$$f_s(\mathbf{x}, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i) = \frac{D(\boldsymbol{\omega}_h)G(\boldsymbol{\omega}_o, \boldsymbol{\omega}_i)F_r(\boldsymbol{\omega}_o)}{4\cos\theta_o \cos\theta_i}, \tag{3.15}$$

where $\boldsymbol{\omega}_h$ is the microfacet normal, D the normal distribution function and F_r the fresnel term. The function G accounts for shadowing and masking of microfacets, meaning that it models that some microfacets might block each other when seen from the observer

(masking) or occlude the incident illumination (shadowing). One caveat of this BSDF model is that it does not account for interreflection between microfacets. This leads to a visible loss in energy (i.e., a darkened appearance), especially when rendering very rough surfaces. One way to address this problem is to estimate microfacet interreflection using a nested Monte Carlo simulation [81, 82].

Layered BSDFs. The above microfacet BSDF describes a rough specular material. Many materials in the real world exhibit some form of layered structure, such as for example a ceramic object with a transparent glossy glaze applied to it. We illustrate a simple layered surface schematically in Figure 3.4c. A range of works suggest algorithms that can layer different BSDFs while accounting for effects such as light scattering inside or between layers [17, 82, 83, 84]. Many production rendering systems use some form of multi-layer BSDF that is designed to model large ranges of appearances in a unified way. A popular such model is the *Disney principled BSDF* [85], which supports a number of different BSDF lobes. An example rendering using this BSDF is shown in Figure 3.3c. While not strictly physically accurate, models such as this one offer a simple way to parameterize the complex space of surface appearances. This is relevant for artistic purposes, but also for inverse problems addressed by differentiable rendering. For differentiable rendering, it is useful to constrain the space of BSDFs to a small set of meaningful parameters in order to restrict the solution space.

3.3.3 Rendering equation

The full light transport in a scene is described by the *rendering equation* [3]:

$$L_o(\mathbf{x}, \boldsymbol{\omega}_o) = L_e(\mathbf{x}, \boldsymbol{\omega}_o) + \int_{S^2} L_i(\mathbf{x}, \boldsymbol{\omega}_i) f_s(\mathbf{x}, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i) \, \mathrm{d}\sigma^{\perp}(\boldsymbol{\omega}_i). \tag{3.16}$$

Here, L_e is radiance emitted from position $\mathbf{x}$ into direction $\boldsymbol{\omega}_o$. This term models the different light sources illuminating the scene, such as emissive surfaces or distant emitters like the sun.

Since photons traverse empty space unimpeded, the incident radiance relates to the outgoing radiance using the *ray tracing function* $r(\mathbf{x}, \boldsymbol{\omega})$. This function traces a ray from $\mathbf{x}$ in direction $\boldsymbol{\omega}$ and returns the closest intersection with another surface in the scene. We can then write:

$$L_i(\mathbf{x}, \boldsymbol{\omega}) = L_o(r(\mathbf{x}, \boldsymbol{\omega}), -\boldsymbol{\omega}). \tag{3.17}$$

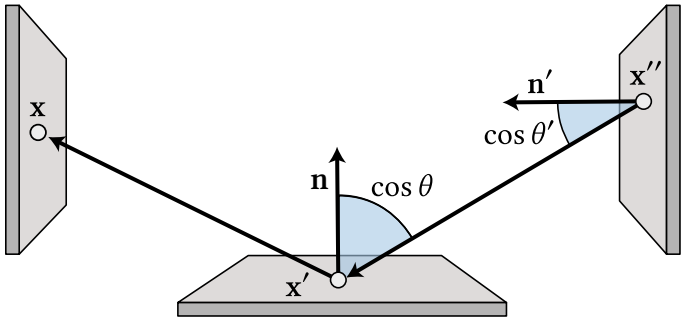


Figure 3.5: Illustration of the geometric configuration used in the definition of the three-point form of the rendering equation.

This makes the rendering equation a recursive equation, where the outgoing radiance occurs on both sides. This type of integral equation is called a *Fredholm equation of the second kind*.

The above rendering equation is in projected solid angle form, but it can also be useful to write it as an integration over surfaces in the scene. This is also called the *three-point form* [51]:

$$L_o(\mathbf{x}' \to \mathbf{x}) = L_e(\mathbf{x}' \to \mathbf{x}) + \int_{\mathcal{M}} G(\mathbf{x} \leftrightarrow \mathbf{x}') L_i(\mathbf{x}'' \to \mathbf{x}') f_s(\mathbf{x}'' \to \mathbf{x}' \to \mathbf{x}) \, \mathrm{d}A(\mathbf{x}''), \tag{3.18}$$

where we now integrate in surface area measure over the union of scene surfaces $\mathcal{M}$. We use the shorthand notation $\mathbf{x}' \to \mathbf{x}$ to indicate the direction of light propagation. This is simply a convention to not have to write directions as functions of $\mathbf{x}$ and $\mathbf{x}'$. For example, $L_o(\mathbf{x}' \to \mathbf{x})$ is defined as:

$$L_o(\mathbf{x}' \to \mathbf{x}) = L_o\left(\mathbf{x}', \frac{\mathbf{x} - \mathbf{x}'}{\|\mathbf{x} - \mathbf{x}'\|}\right). \tag{3.19}$$

The definitions are analogous for L_i, L_e and f_s. The *geometry term* G accounts for the change from solid angle to area measure:

$$G(\mathbf{x} \leftrightarrow \mathbf{x}') = V(\mathbf{x} \leftrightarrow \mathbf{x}') \frac{|\cos\theta \cos\theta'|}{\|\mathbf{x} - \mathbf{x}'\|^2}. \tag{3.20}$$

The angles θ and θ' denote angles between the segment $\mathbf{x}'' \to \mathbf{x}'$ and the surface normals at $\mathbf{x}'$ and $\mathbf{x}''$. Figure 3.5 illustrates the geometrical configuration of all these terms. The visibility function $V(\mathbf{x} \leftrightarrow \mathbf{x}')$ is zero if a surface is intersecting the straight line segment $\mathbf{x} \to \mathbf{x}'$ and one otherwise. Deriving the geometry term is challenging and we refer to Lessig et al. [86] for a full derivation using exterior calculus.

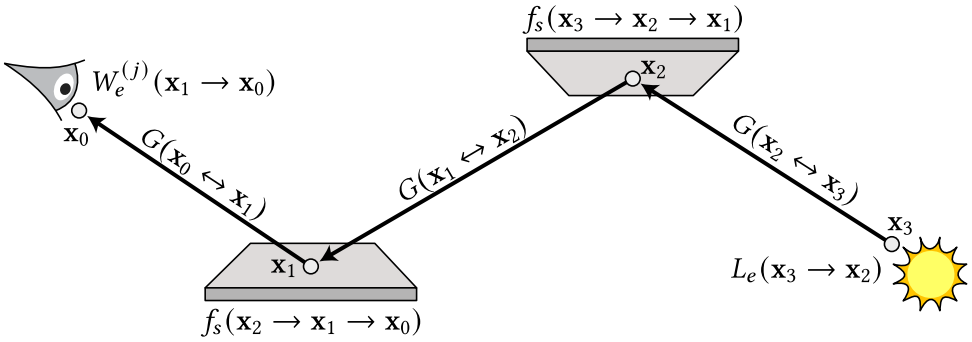


Figure 3.6: Illustration of the the terms used in the definition of the image contribution function $f_j(\bar{\mathbf{x}})$.

3.3.4 Path space integral

We can transform the recursive rendering equation into a non-recursive, high-dimensional integral. This *path integral* formulation expresses the pixel intensity as an integral over the infinitely-dimensional space of light paths. In this context, a light path is assumed to be consisting of straight segments[1]. The path integral formulation is fundamental to developing various Monte Carlo estimators, in particular, more advanced methods such as bidirectional path tracing [54, 55] or Metropolis light transport [87]. It is convenient to be able to reason about (differentiable) rendering algorithms in a non-recursive manner.

The path space formulation can be derived by recursively substituting Equation 3.18 into the measurement equation (Equation 3.9). We express the intensity of a pixel as a sum of integrals, each accounting for paths of a different number of segments:

$$
\begin{aligned}
I_j = &\int_{\mathcal{M}^2} W_e^{(j)}(\mathbf{x}_1 \to \mathbf{x}_0) G(\mathbf{x}_0 \leftrightarrow \mathbf{x}_1)\, L_e(\mathbf{x}_1 \to \mathbf{x}_0)\, \mathrm{d}\mathbf{x}_1\, \mathrm{d}\mathbf{x}_0 \\
&+ \int_{\mathcal{M}^3} W_e^{(j)}(\mathbf{x}_1 \to \mathbf{x}_0) G(\mathbf{x}_0 \leftrightarrow \mathbf{x}_1)\, f_s(\mathbf{x}_2 \to \mathbf{x}_1 \to \mathbf{x}_0) \\
&\qquad\qquad G(\mathbf{x}_1 \leftrightarrow \mathbf{x}_2)\, L_e(\mathbf{x}_2 \to \mathbf{x}_1)\, \mathrm{d}\mathbf{x}_2\, \mathrm{d}\mathbf{x}_1\, \mathrm{d}\mathbf{x}_0 \\
&+ \int_{\mathcal{M}^4} W_e^{(j)}(\mathbf{x}_1 \to \mathbf{x}_0) G(\mathbf{x}_0 \leftrightarrow \mathbf{x}_1)\, f_s(\mathbf{x}_2 \to \mathbf{x}_1 \to \mathbf{x}_0) \\
&\quad G(\mathbf{x}_1 \leftrightarrow \mathbf{x}_2)\, f_s(\mathbf{x}_3 \to \mathbf{x}_2 \to \mathbf{x}_1) G(\mathbf{x}_2 \leftrightarrow \mathbf{x}_3)\, L_e(\mathbf{x}_3 \to \mathbf{x}_2)\, \mathrm{d}\mathbf{x}_3\, \mathrm{d}\mathbf{x}_2\, \mathrm{d}\mathbf{x}_1\, \mathrm{d}\mathbf{x}_0 \\
&+ \int_{\mathcal{M}^5} \dots
\end{aligned}
\tag{3.21}
$$

We use the simplified notation $\mathrm{d}\mathbf{x}_i = \mathrm{d}A(\mathbf{x}_i)$ to denote integration using the local area measure. The product of sensor importance, BSDF and geometry terms is also called the

[1]This path integral formulation is not to be confused with *Feynman path integrals*, in which all *curved* paths between two end points are considered.

throughput β. For a light path $\bar{\mathbf{x}} = (\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_n)$ it is defined as:

$$\beta(\bar{\mathbf{x}}) = W_e^{(j)}(\mathbf{x}_1 \to \mathbf{x}_0) \prod_{i=1}^{n-1} G(\mathbf{x}_i \leftrightarrow \mathbf{x}_{i-1}) f_s(\mathbf{x}_{i+1} \to \mathbf{x}_i \to \mathbf{x}_{i-1}). \tag{3.22}$$

The *image contribution function* $f_j(\bar{\mathbf{x}})$ measures the total contribution of a light path to pixel j. It is defined separately for each path length n and is the product of the throughput and the emission at the last path vertex:

$$f_j(\bar{\mathbf{x}}) = W_e^{(j)}(\mathbf{x}_1 \to \mathbf{x}_0) \prod_{i=1}^{n-1} G(\mathbf{x}_i \leftrightarrow \mathbf{x}_{i-1}) f_s(\mathbf{x}_{i+1} \to \mathbf{x}_i \to \mathbf{x}_{i-1}) L_e(\mathbf{x}_n \to \mathbf{x}_{n-1}). \tag{3.23}$$

Figure 3.6 illustrates the terms used in this definition for an example path. Using this function we can write the value of a pixel as integral over the *path space* $\mathcal{P}$:

$$I_j = \int_{\mathcal{P}} f_j(\bar{\mathbf{x}}) \, \mathrm{d}\mu(\bar{\mathbf{x}}). \tag{3.24}$$

The path space is defined as the set of all possible light paths in a scene. The measure μ is the product of the local surface area measures:

$$\mu(\bar{\mathbf{x}}) = \prod_{i=0}^{n} \mathrm{d}A(\mathbf{x}_i). \tag{3.25}$$

3.4 Volumetric light transport

The discussion so far assumed the scene to be made up of surfaces in a vacuum. *Volumetric* light transport occurs whenever the space between surfaces is filled with a *participating medium* that interacts with light. The appearance of fire, smoke, clouds, cloth or human skin is largely due to volumetric light transport, see also Figure 3.7 for a few example photographs.

For inverse rendering applications, simulating volumetric scattering is essential to recover the appearance of scenes containing such effects. Many scientific applications of inverse rendering heavily feature participating media. For example, the scattering in biological tissue, Earth's atmosphere and 3D printed plastics is volumetric. Moreover, volumetric appearance is oftentimes strongly determined by multiple scattering, which makes it a prime use case for physically-based differentiable rendering. Additionally, volumetric rendering models can be useful as a general scene representation, as discussed further in Chapters 4 and 7. While in forward rendering volumetric light transport is somewhat of a special case, for inverse rendering it is at least equally as important as surface light transport. In the following, we outline the fundamental theory of participating media and volumetric light transport.

(a) Cloud (b) Orange juice (c) Human skin (d) Candle

Figure 3.7: Various examples of real world scenes containing participating media. The appearance of all these scenes is largely determined by volumetric scattering. For the candle **(d)**, both the wax and the flame are participating media.

3.4.1 Radiative transfer

For now, we assume that the participating media we consider are made up of uncorrelated, microscopic particles. The size of the particles is so small that we can reason about light interactions in a statistical sense, without explicitly considering individual particles. Along a beam, the medium particles can either absorb, out-scatter, in-scatter or emit light. Figure 3.8 schematically illustrates these different effects. In the following, we provide the mathematical description of these interactions.

Absorption. At any point along a beam, a fraction of photons might be absorbed by medium particles. Since we do not intend to model individual photons or particles, we reason about the absorption in a differential sense: what is the derivative of the radiance $L(\mathbf{x}, \boldsymbol{\omega})$ with respect to a small step t along the ray? Formally, we want to characterize

$$\frac{\mathrm{d}}{\mathrm{d}t} L(\mathbf{x} + \boldsymbol{\omega} t, \boldsymbol{\omega})\Big|_{t=0} = (\boldsymbol{\omega} \cdot \nabla) L(\mathbf{x}, \boldsymbol{\omega}), \tag{3.26}$$

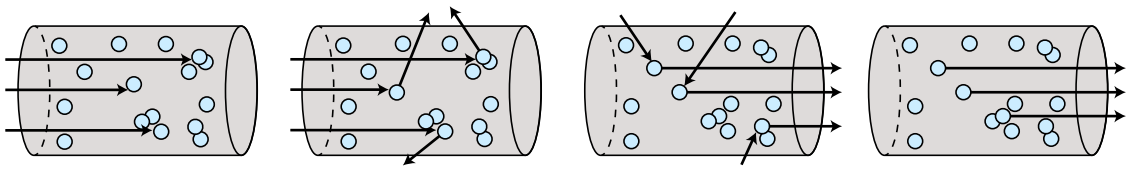
where $\boldsymbol{\omega} \cdot \nabla$ denotes the directional derivative. Assuming uncorrelated particles, the derivative due to absorption is:

$$(\boldsymbol{\omega} \cdot \nabla) L(\mathbf{x}, \boldsymbol{\omega}) = -\sigma_a L(\mathbf{x}, \boldsymbol{\omega}), \tag{3.27}$$

where the absorption coefficient σ_a specifies the amount of absorption in m^{-1}.

Out-scattering. Radiance can further be scattered away from the current beam, which results in a loss quantified by the scattering coefficient σ_s $[\mathrm{m}^{-1}]$:

$$(\boldsymbol{\omega} \cdot \nabla) L(\mathbf{x}, \boldsymbol{\omega}) = -\sigma_s L(\mathbf{x}, \boldsymbol{\omega}). \tag{3.28}$$



(a) Absorption (b) Out-scattering (c) In-scattering (d) Emission

Figure 3.8: Along a ray through a medium, radiance can either be absorbed **(a)**, out-scattered **(b)**, in-scattered **(c)** or emitted **(d)**. Out-scattering refers to the loss in radiance due to light scattering away from the current direction, and in-scattering summarizes the gains due to light scattering towards the current direction. Conceptually, these effects are caused by photons interacting with medium particles. In practice, we do not need to explicitly simulate individual photons or particles.

In-scattering. On the other hand, the radiance along a beam can also be increased by light being in-scattered. The corresponding differential change is

$$(\boldsymbol{\omega} \cdot \nabla) L(\mathbf{x}, \boldsymbol{\omega}) = \sigma_s L_s(\mathbf{x}, \boldsymbol{\omega}), \tag{3.29}$$

where the incident radiance is computed as the spherical integral

$$L_s(\mathbf{x}, \boldsymbol{\omega}_o) = \int_{\mathcal{S}^2} f_p(\mathbf{x}, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i) L_i(\mathbf{x}, \boldsymbol{\omega}_i) \, \mathrm{d}\boldsymbol{\omega}_i. \tag{3.30}$$

The *phase function* f_p models the angular variation of scattering, similar to the BSDF in surface rendering. Unlike a BSDF, the phase function is usually normalized, as absorption is accounted for separately:

$$\int_{\mathcal{S}^2} f_p(\mathbf{x}, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i) \, \mathrm{d}\boldsymbol{\omega}_i = 1. \tag{3.31}$$

Emission. Finally, the medium might emit photons itself. The radiance derivative is then simply

$$(\boldsymbol{\omega} \cdot \nabla) L(\mathbf{x}, \boldsymbol{\omega}) = \sigma_a L_e(\mathbf{x}, \boldsymbol{\omega}), \tag{3.32}$$

where the function L_e is the emitted radiance. Sometimes the multiplication by the absorption coefficient is left out, but this is simply a matter of the definition of L_e.

Radiative transfer equation. Summing up all the differential changes to radiance, we obtain the *radiative transfer equation* (RTE) [88] that describes the equilibrium radiance distribution:

$$(\boldsymbol{\omega} \cdot \nabla) L(\mathbf{x}, \boldsymbol{\omega}) = \sigma_a L_e(\mathbf{x}, \boldsymbol{\omega}) + \sigma_s \int_{\mathcal{S}^2} f_p(\mathbf{x}, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i) L(\mathbf{x}, \boldsymbol{\omega}) \, \mathrm{d}\boldsymbol{\omega}_i - \sigma_a L(\mathbf{x}, \boldsymbol{\omega}) - \sigma_s L(\mathbf{x}, \boldsymbol{\omega}). \tag{3.33}$$

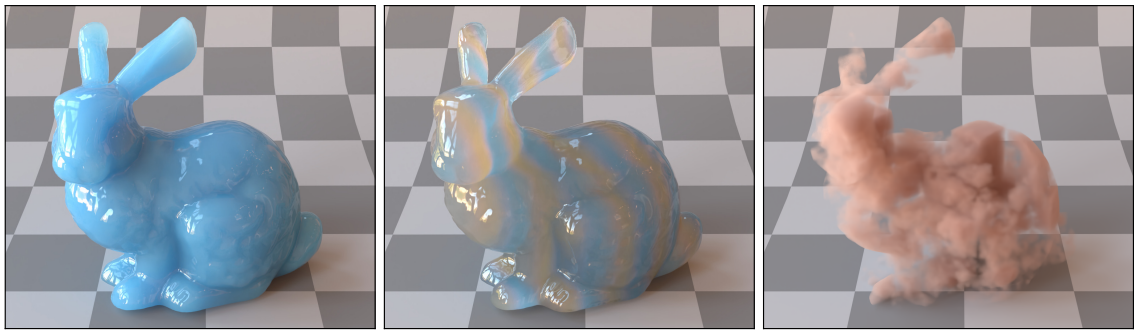


Figure 3.9: Example renderings of participating media with different (spatially-varying) coefficients. Participating media can represent a range of appearances, from objects exhibiting subsurface scattering to clouds of smoke or dust.

In Figure 3.9 we show a few examples renderings of participating media produced by solving the RTE using Monte Carlo integration.

The losses due to absorption and out-scattering can be combined using the *extinction coefficient* $\sigma_t = \sigma_a + \sigma_s$. The *transmittance* function measures the total fraction of radiance that is lost along a beam between two points in the medium. It can be derived from the RTE and has the following explicit form:

$$\mathrm{T}(\mathbf{x}, \mathbf{x}') = \exp\left(-\int_0^{\|\mathbf{x}'-\mathbf{x}\|} \sigma_t(\mathbf{x}_t)\,\mathrm{d}t\right). \tag{3.34}$$

All the medium parameters are potentially spatially varying. We thus write σ_t as a function of $\mathbf{x}_t$, which is a shorthand notation for a point at distance t between $\mathbf{x}$ and $\mathbf{x}'$:

$$\mathbf{x}_t \coloneqq \mathbf{x} + t\frac{\mathbf{x}' - \mathbf{x}}{\|\mathbf{x}' - \mathbf{x}\|}. \tag{3.35}$$

The inner integral over the extinction coefficient is also called the *optical depth* τ. The transmittance function here is exponential. This is a consequence of the assumption of uncorrelated medium particles. We will discuss non-exponential transmittance in Chapter 7, where we use such alternative formulations to improve volumetric scene representation.

The RTE can be integrated to derive the *volume rendering equation* (VRE), which, analogous to the surface case, is a recursive integral equation:

$$\begin{aligned} L_o(\mathbf{x}, \boldsymbol{\omega}_o) = \int_0^s \mathrm{T}(\mathbf{x}, \mathbf{x}_t)\left[\sigma_a(\mathbf{x}_t)L_e(\mathbf{x}_t, \boldsymbol{\omega}_o) + \sigma_s(\mathbf{x}_t)\int_{\mathcal{S}^2} f_p(\mathbf{x}_t, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i)L_i(\mathbf{x}_t, \boldsymbol{\omega}_i)\,\mathrm{d}\boldsymbol{\omega}_i\right]\mathrm{d}t \\ + \mathrm{T}(\mathbf{x}, \mathbf{x}_s)L_o(\mathbf{x}_s, \boldsymbol{\omega}_o), \end{aligned} \tag{3.36}$$

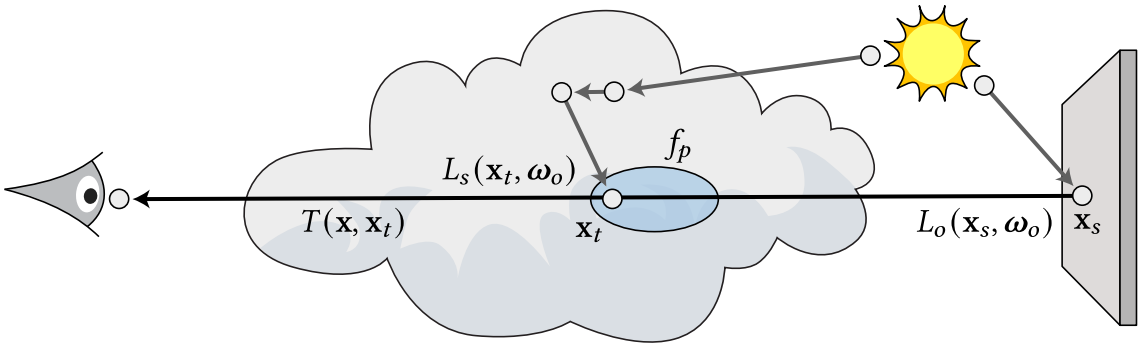


Figure 3.10: This illustration shows some of the terms used in the volume rendering equation. The outgoing radiance consists of in-scattered radiance on points in the volume and the outgoing radiance at the closest surface along the ray. Both quantities are attenuated by the transmittance T.

where t is the distance along the ray $(\mathbf{x}, -\boldsymbol{\omega}_o)$ and s is the distance to the closest surface along that ray. The term $L_o(\mathbf{x}_s, \boldsymbol{\omega}_o)$ is the outgoing radiance at the surface at point $\mathbf{x}_s$. Some of the terms in this equation are illustrated in Figure 3.10

3.4.2 Phase functions

Similar to the BSDF for surfaces, the phase function is an important factor determining the appearance of scattering volumes. A variety of analytical phase function models have been derived. The simplest one is the *isotropic* phase function, which models a uniform directional distribution:

$$f_p^{\text{iso}}(\mathbf{x}, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i) = \frac{1}{4\pi}. \tag{3.37}$$

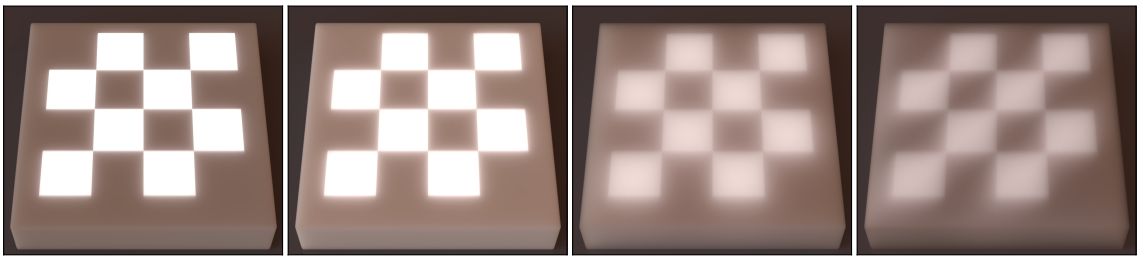
Like the diffuse BSDF, this is an idealized model that is unlikely to hold for real scenes.

Henyey-Greenstein phase function. A slightly more complex model is the anisotropic *Henyey-Greenstein* [89] phase function. It was designed to fit measured data and is defined as

$$f_p^{\text{HG}}(\mathbf{x}, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i) = \frac{1}{4\pi} \frac{1 - g^2}{(1 + g^2 - 2g\langle -\boldsymbol{\omega}_o, \boldsymbol{\omega}_i \rangle)^{3/2}}. \tag{3.38}$$

The model is parameterized by the mean cosine $g \in [-1, 1]$. The mean cosine of a phase function is the expected value of the cosine of the angle between incident and outgoing direction:

$$g = \int_{S^2} \langle -\boldsymbol{\omega}_o, \boldsymbol{\omega}_i \rangle f_p(\mathbf{x}, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i) \, \mathrm{d}\boldsymbol{\omega}_i. \tag{3.39}$$



(a) HG, $g = -0.75$ (b) HG, $g = 0$ (c) HG, $g = 0.75$ (d) SGGX

Figure 3.11: Example renderings using different phase functions. The scene consists of a slab illuminated by a projected checkerboard pattern. For the Henyey-Greenstein phase function, the mean cosine parameter g controls if the phase function is backscattering **(a)**, isotropic **(b)** or forward scattering **(c)**. The SGGX phase function **(d)** models an anisotropic medium. The parameters here are chosen such that light scatters primarily along the diagonal from bottom left to top right.

A *forward scattering* phase function has a positive mean cosine and a *backscattering* phase function has a negative mean cosine. The Henyey-Greenstein phase function conveniently parameterizes the space between forward and backscattering, reducing to the isotropic phase function for $g = 0$. Figure 3.11 shows example renderings using different values for g.

Microflake theory. The previously described models describe media that are locally rotation invariant, or *isotropic*. Note that the terms isotropic and anisotropic have two different uses in the context of participating media. Both the phase function itself and the medium can either be isotropic or anisotropic. For example, the Henyey-Greenstein phase function is an anisotropic phase function describing an isotropic medium. The Henyey-Greenstein scattering only depends on the angle between incident and outgoing direction, but not on their absolute orientation.

In an anisotropic medium, the scattering of light is no longer rotation invariant. An important class of phase function models for anisotropic media are microflake phase functions [90, 91]. These allow modeling oriented structures, such as observed in wood or woven fabrics [92]. In these materials, the light scattering is strongly influenced by the fiber orientation. In Chapter 7, we use a microflake model to convert scenes consisting of surfaces to volumes. By using a microflake phase function, we can more closely match the appearance of opaque surfaces in a purely volumetric rendering model. Anisotropic media have also been used as a model to accurately downsample existing isotropic volumes [93].

Conceptually, microflake phase functions assume that the medium is made up of microscopic flakes. These flakes are oriented disks with a surface BSDF. The microflake

model is the volumetric analog of the microfacet model used for surfaces. Just like for microfacets, we do not reason about individual flakes, but only consider the aggregate effect caused by a certain normal distribution function (NDF). The phase function then becomes an expected value over microflake normals:

$$f_p(\mathbf{x}, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i) = \frac{1}{\sigma(\boldsymbol{\omega}_i)} \int_{S^2} p(\boldsymbol{\omega}_m, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i) \langle \boldsymbol{\omega}_i, \boldsymbol{\omega}_m \rangle D(\boldsymbol{\omega}_m) \, \mathrm{d}\boldsymbol{\omega}_m, \tag{3.40}$$

where $\boldsymbol{\omega}_m$ is a microflake normal, $D(\boldsymbol{\omega}_m)$ the microflake NDF and $p(\boldsymbol{\omega}_m, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i)$ the microflake's surface BSDF. For a microflake distribution, the projected area along a direction $\boldsymbol{\omega}_i$ is

$$\sigma(\boldsymbol{\omega}_i) = \int_{S^2} \langle \boldsymbol{\omega}_i, \boldsymbol{\omega}_m \rangle D(\boldsymbol{\omega}_m) \, \mathrm{d}\boldsymbol{\omega}_m. \tag{3.41}$$

The microflake model has the consequence that the medium's extinction is scaled by this projected area and thus becomes directionally varying:

$$\sigma_t(\boldsymbol{\omega}_i) = \sigma_t \sigma(\boldsymbol{\omega}_i). \tag{3.42}$$

Intuitively, the extinction coefficient depends on the proportion of microflakes facing direction $\boldsymbol{\omega}_i$.

A common microflake phase function is the *SGGX* [91] phase function. It is the three-dimensional generalization of the Trowbridge-Reitz (GGX) [78, 80] microfacet distribution used for surface rendering. The SGGX microflake normal distribution is defined to match the distribution of surface normals of a 3D ellipsoid. The ellipsoid can approximate the normal distribution of both fibers and flat surfaces, and anything in between. Figure 3.11d contains an example rendering using fiber-like settings. The ellipsoid can be parameterized in terms of a 3D rotation matrix $[\boldsymbol{\omega}_1, \boldsymbol{\omega}_2, \boldsymbol{\omega}_3]$ and the projected areas along each coordinate axis:

$$S = (\boldsymbol{\omega}_1, \boldsymbol{\omega}_2, \boldsymbol{\omega}_3) \begin{pmatrix} \sigma^2(\boldsymbol{\omega}_1) & 0 & 0 \\ 0 & \sigma^2(\boldsymbol{\omega}_2) & 0 \\ 0 & 0 & \sigma^2(\boldsymbol{\omega}_3) \end{pmatrix} (\boldsymbol{\omega}_1, \boldsymbol{\omega}_2, \boldsymbol{\omega}_3)^T. \tag{3.43}$$

The matrix S is a symmetrical 3-by-3 matrix and can be stored compactly using just six floating point numbers. In Figure 3.12, we show a few example ellipsoids and their corresponding normal distributions. Heitz et al. [91] show how to fit S to a given distribution of microflake normals and present efficient sampling and evaluation routines for both specular and diffuse microflake BSDFs. The sampling uses visible normal sampling [59, 94] to only sample to microflake normals that face the incident direction.

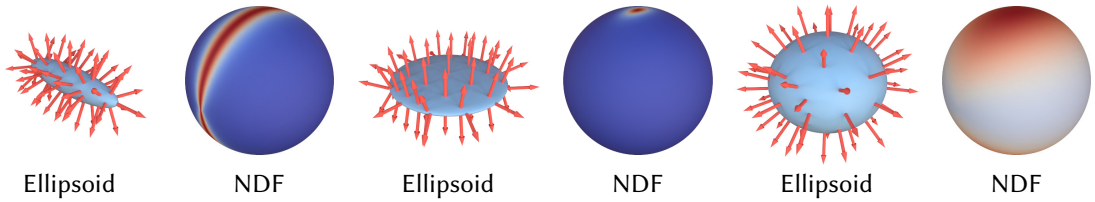


Figure 3.12: The SGGX normal distribution function (NDF) matches the normals of a 3D ellipsoid. In these three examples, different ellipsoid parameters are used to produce a range of NDFs, including fiber-like (left), surface-like (middle) and nearly uniform (right) distributions.

3.4.3 Volumetric path space integral

The path space integral formulation from Section 3.3.4 can be extended to volumetric light transport [9, 95]. We can write a unified path integral formulation that accounts for both surface and volumetric scattering. The image contribution function is then written

$$f_j(\bar{\mathbf{x}}) = W_e^{(j)}(\mathbf{x}_1 \to \mathbf{x}_0) \prod_{i=1}^{n-1} G(\mathbf{x}_i \leftrightarrow \mathbf{x}_{i-1}) \\ \mathrm{T}(\mathbf{x}_i \leftrightarrow \mathbf{x}_{i-1}) \hat{f}(\mathbf{x}_{i+1} \to \mathbf{x}_i \to \mathbf{x}_{i-1}) L_e(\mathbf{x}_n \to \mathbf{x}_{n-1}). \tag{3.44}$$

Instead of the BSDF, we use a generalized term $\hat{f}$ that is evaluates either the BSDF or the phase function scaled by the scattering coefficient σ_s, depending on if the current path vertex is on a surface or inside the volume:

$$\hat{f}(\mathbf{x} \to \mathbf{x}' \to \mathbf{x}'') = \begin{cases} f_s(\mathbf{x} \to \mathbf{x}' \to \mathbf{x}'') & \mathbf{x}' \in \mathcal{M} \\ \sigma_s(\mathbf{x}') f_p(\mathbf{x} \to \mathbf{x}' \to \mathbf{x}'') & \mathbf{x}' \in \mathcal{V} \end{cases} \tag{3.45}$$

The emission term $L_e(\mathbf{x}_n \to \mathbf{x}_{n-1})$ will evaluate either surface or volume emission, depending on the type of vertex $\mathbf{x}_n$. The geometry term is also slightly modified:

$$G(\mathbf{x} \leftrightarrow \mathbf{x}') = V(\mathbf{x} \leftrightarrow \mathbf{x}') \frac{D_{\mathbf{x}}(\mathbf{x}') D_{\mathbf{x}'}(\mathbf{x})}{\|\mathbf{x} - \mathbf{x}'\|^2}, \tag{3.46}$$

where $D_{\mathbf{x}}(\mathbf{x}')$ evaluates to the usual cosine term if $\mathbf{x}$ is on a surface and to 1 otherwise. There is no foreshortening effect for a path vertex that lies in the volume, but the division by the squared distance is still needed. The integration measure is now the product of the local area and volume measures, depending on the type of the current vertex:

$$\mu(\bar{\mathbf{x}}) = \prod_{i=0}^{n} \begin{cases} \mathrm{d}A(\mathbf{x}_i) & \mathbf{x}_i \in \mathcal{M} \\ \mathrm{d}\mathcal{V}(\mathbf{x}_i) & \mathbf{x}_i \in \mathcal{V} \end{cases} \tag{3.47}$$

3.4.4 Null-scattering path integral

The above path integral still contains a nested integral over optical depth in the definition of the transmittance term (see Equation 3.34), which complicates the formulation of unbiased Monte Carlo integration algorithms. We can eliminate the nested transmittance integral by introducing *null scattering* [95, 96, 97]. To do so, we add a certain amount of fictitious *null* particles, which do not scatter or absorb radiance. We first define $\bar{\sigma}$ to be a constant upper bound of the medium's extinction and then define the null scattering coefficient $\sigma_n(\mathbf{x}) = \bar{\sigma} - \sigma_t(\mathbf{x}) \geq 0$. We then introduce null scattering terms into the RTE (Equation 3.33) without modifying the result:

$$\begin{aligned}
(\boldsymbol{\omega} \cdot \nabla) L(\mathbf{x}, \boldsymbol{\omega}) &= \sigma_a L_e(\mathbf{x}, \boldsymbol{\omega}) + \sigma_s \int_{\mathcal{S}^2} f_p(\mathbf{x}, \boldsymbol{\omega}, \boldsymbol{\omega}) L(\mathbf{x}, \boldsymbol{\omega}) \, \mathrm{d}\boldsymbol{\omega}' - \sigma_t L(\mathbf{x}, \boldsymbol{\omega}) \\
&= \sigma_a L_e(\mathbf{x}, \boldsymbol{\omega}) + \sigma_s \int_{\mathcal{S}^2} f_p(\mathbf{x}, \boldsymbol{\omega}, \boldsymbol{\omega}) L(\mathbf{x}, \boldsymbol{\omega}) \, \mathrm{d}\boldsymbol{\omega}' + \sigma_n L(\mathbf{x}, \boldsymbol{\omega}) - \sigma_t L(\mathbf{x}, \boldsymbol{\omega}) - \sigma_n L(\mathbf{x}, \boldsymbol{\omega}) \\
&= \sigma_a L_e(\mathbf{x}, \boldsymbol{\omega}) + \sigma_s \int_{\mathcal{S}^2} f_p(\mathbf{x}, \boldsymbol{\omega}, \boldsymbol{\omega}) L(\mathbf{x}, \boldsymbol{\omega}) \, \mathrm{d}\boldsymbol{\omega}' + \sigma_n L(\mathbf{x}, \boldsymbol{\omega}) - \bar{\sigma} L(\mathbf{x}, \boldsymbol{\omega}). \qquad (3.48)
\end{aligned}$$

The null-scattering RTE is then integrated along $\boldsymbol{\omega}$ to yield the null-scattering volume rendering equation [97, 98]:

$$\begin{aligned}
L_o(\mathbf{x}, \boldsymbol{\omega}_o) = \int_0^s \bar{\mathrm{T}}(\mathbf{x}, \mathbf{x}_t) \Big[& \sigma_a(\mathbf{x}_t) L_e(\mathbf{x}_t, \boldsymbol{\omega}_o) \\
& + \sigma_s(\mathbf{x}_t) \int_{\mathcal{S}^2} f_p(\mathbf{x}_t, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i) L_i(\mathbf{x}_t, \boldsymbol{\omega}_i) \, \mathrm{d}\boldsymbol{\omega}_i + \sigma_n(\mathbf{x}_t) L_o(\mathbf{x}_t, \boldsymbol{\omega}_o) \Big] \, \mathrm{d}t \\
& + \bar{\mathrm{T}}(\mathbf{x}, \mathbf{x}_s) L_o(\mathbf{x}_s, \boldsymbol{\omega}_o), \qquad (3.49)
\end{aligned}$$

where $\bar{\mathrm{T}}(\mathbf{x}, \mathbf{x}_t) = \exp(-t\bar{\sigma})$ is the transmittance computed using the constant extinction majorant. We no longer need to solve an integral to compute the transmittance function. The newly added null-scattering terms compensate for the fact that $\bar{\mathrm{T}}(\mathbf{x}, \mathbf{x}_t) \leq \mathrm{T}(\mathbf{x}, \mathbf{x}_t)$.

Based on this formulation, a null-scattering path integral formulation can be derived [95]. Compared to the previous path integral, there are now three types of vertices: surface, volume and null vertices. A null vertex is a vertex in the volume where we evaluate $\sigma_n(\mathbf{x}_t) L_o(\mathbf{x}_t, \boldsymbol{\omega}_o)$. At such a vertex, radiance continues in the same direction and is scaled by the null-scattering coefficient. Formally, we now write the image-contribution function as:

$$\begin{aligned}
f_j(\bar{\mathbf{x}}) = W_e^{(j)}(\mathbf{x}_1 \to \mathbf{x}_0) & \prod_{i=1}^{R-1} G\left(\mathbf{x}_{r_i} \leftrightarrow \mathbf{x}_{r_{i-1}}\right) \\
& \prod_{i=1}^{n-1} \bar{\mathrm{T}}(\mathbf{x}_i \to \mathbf{x}_{i-1}) \hat{f}(\mathbf{x}_{i+1} \to \mathbf{x}_i \to \mathbf{x}_{i-1}) L_e(\mathbf{x}_n \to \mathbf{x}_{n-1}). \qquad (3.50)
\end{aligned}$$

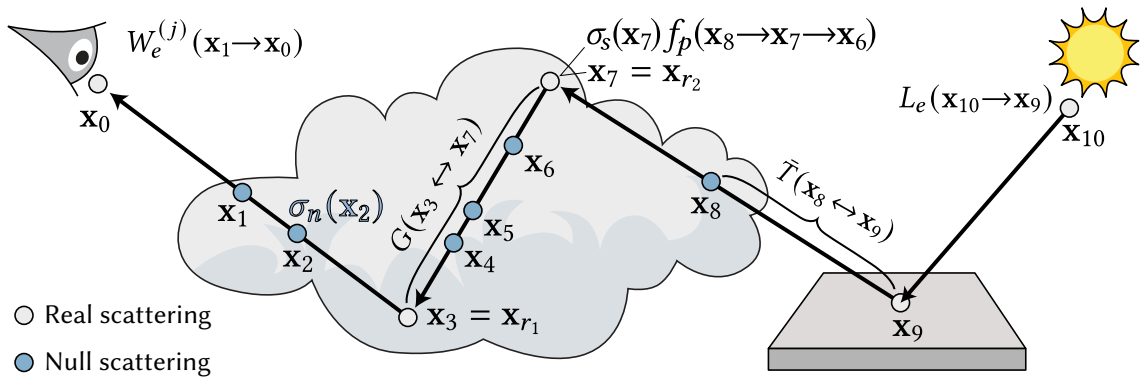


Figure 3.13: Illustration of the terms used in the definition of the null-scattering path integral formulation. We distinguish between real scattering (grey) and null scattering vertices (blue).

Some of the terms used here are illustrated in Figure 3.13. The geometry term G only needs to be evaluated between non-null vertices (i.e., surface or volume vertices), listed with indices $\{r_0, \ldots, r_{R-1}\}$. The function $\hat{f}$ becomes:

$$\hat{f}(\mathbf{x} \to \mathbf{x}' \to \mathbf{x}'') = \begin{cases} f_s(\mathbf{x} \to \mathbf{x}' \to \mathbf{x}'') & \mathbf{x}' \in \mathcal{M} \\ \sigma_s(\mathbf{x}') f_p(\mathbf{x} \to \mathbf{x}' \to \mathbf{x}'') & \mathbf{x}' \in \mathcal{V} \\ \sigma_n(\mathbf{x}') & \mathbf{x}' \in \mathcal{V}_\delta \end{cases} \tag{3.51}$$

where $\mathcal{V}_\delta$ is the space of null vertices. The null vertices are constrained to lie on straight lines between volume and surface vertices. Their contributions are integrated along the line between the previous and next non-null vertices, see Miller et al. [95] for details on the used integration measures. We will discuss how this null scattering formulation can be used to formulate unbiased Monte Carlo estimators in Section 3.5.2.

3.5 Monte Carlo rendering

Starting from these theoretical foundations of light transport, we can render images using Monte Carlo integration. Our goal is to estimate pixel values by sampling a finite number of light paths:

$$I_j = \int_{\mathcal{P}} f_j(\bar{\mathbf{x}}) \, \mathrm{d}\bar{\mathbf{x}} \approx \frac{1}{N} \sum_{i=1}^{N} \frac{f_j(\bar{\mathbf{x}})}{p(\bar{\mathbf{x}})}, \tag{3.52}$$

where f_j is the image contribution function and p is the PDF of sampling a given path. In the following, we first discuss surface rendering and then explain the changes needed for the volumetric case.

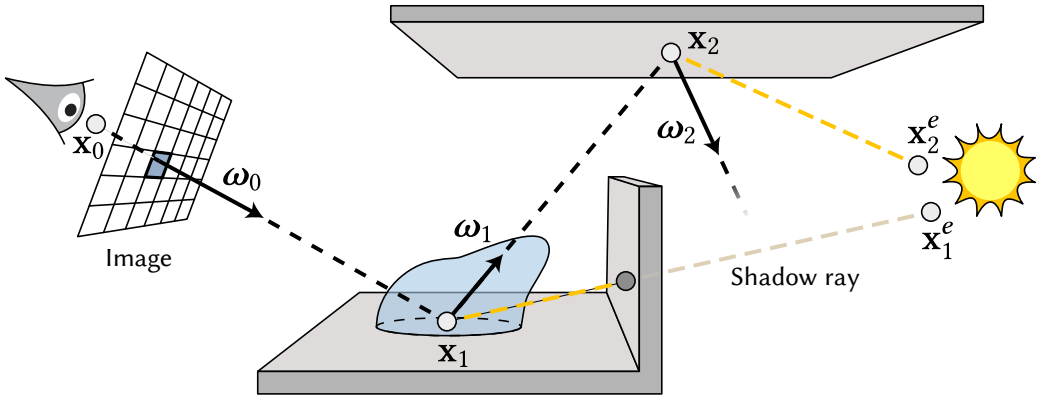


Figure 3.14: Illustration of the path tracing algorithm. Starting from the camera, a light path is constructed by alternating direction sampling and ray intersection. At each path vertex, next event estimation samples a point $\mathbf{x}_i^e$ on the light source and checks for occlusion by tracing a *shadow ray*.

3.5.1 Path tracing

The simplest and at the same time most common Monte Carlo rendering algorithm is unidirectional path tracing [3]. All differentiable rendering techniques described in this thesis use a variation of this method to render images. Path tracing constructs light paths by starting from the camera and iteratively sampling the direction of the next path segment, as shown in Figure 3.14. The basic surface path tracing algorithm consists of the following sequence of steps:

1. Sample a position $\mathbf{x}_0$ and outgoing direction $\boldsymbol{\omega}_0$ according to the sensor. Initialize a throughput variable $\beta = 1$ and running radiance estimate $L = 0$.
2. Intersect the ray $(\mathbf{x}_i, \boldsymbol{\omega}_i)$ with the scene geometry to obtain the next vertex $\mathbf{x}_{i+1}$.
3. If the surface is emissive, add the emission weighted by throughput β to the radiance estimate L.
4. Sample the direction $\boldsymbol{\omega}_{i+1}$ of the next path segment and multiply β by the ratio of the BSDF evaluation and sampling PDF.
5. Repeat steps 2 to 4 until a termination criterion is met. Once the path terminates, accumulate L to the image.

Sampling strategies. This above procedure samples a path $\bar{\mathbf{x}} \in \mathcal{P}$ segment by segment. Therefore, the probability of sampling the entire path is decomposed into a product of the (conditional) probabilities of the local direction sampling steps. The choice of

these local sampling probabilities will affect the variance of the final estimator. Path tracing usually samples the direction of the next segment using a distribution that closely follows the BSDF at the current path vertex. Certain BSDFs admit a perfectly proportional sampling strategy (e.g., the diffuse BSDF), while for others (e.g., microfacet BSDFs) we only have approximately proportional distributions [80, 94].

Next event estimation. The variance of path tracing can be reduced further using *next event estimation*, which explicitly samples the contributions due to directly visible light sources at each path vertex. For this, we sample a position $\mathbf{x}_e$ on a light source at each iteration and evaluate visibility, geometry and BSDF terms for a path connecting from the current vertex $\mathbf{x}_i$ to the sampled point $\mathbf{x}_i^e$. (see also Figure 3.14). Next event estimation and BSDF sampling are combined using multiple importance sampling [60] to obtain a robust estimator that works for a large range of light sources and BSDF parameters.

Path termination. The path tracing algorithm terminates when the path's throughput becomes zero or no further ray intersection is found. The efficiency can be improved by probabilistically terminating light paths using *Russian roulette*. At each iteration, Russian roulette terminates the light path with a probability q. If the light path is continued, the throughput is scaled by the reciprocal of the continuation probability. Intuitively, this scales up the path's throughput to compensate for the energy loss due to early termination. As a result, the estimator remains unbiased. The termination probability can be defined as a constant or consider the current path throughput. Using Russian roulette will reduce the computation time by preventing the algorithm from tracing too many long light paths.

Advanced sampling strategies. Path sampling according to the BSDF completely ignores the distribution of the radiance L_i incident at the current vertex. Next event estimation addresses this, but only handles directly visible light sources. For scenes with complex indirect illumination, path tracing results in a high variance estimator (i.e., noise in the final image). These issues can for example be addressed by bidirectional path tracing methods [54, 55], which construct paths both from the sensor and the emitters. Another option is to replace standard Monte Carlo integration with Markov chain Monte Carlo [26], which constructs paths by mutating a set of initial samples [87, 99]. Path guiding [56, 57, 58, 100, 101] algorithms on the other hand fit a sampling distribution to the current scene. This distribution is then used to importance sample directions that

more closely follow the distribution of incident radiance. All the work in this thesis uses the standard path tracing algorithm without any guiding or bidirectional methods. However, it is clear that for many complex settings, more advanced path sampling strategies could improve the robustness of inverse rendering methods.

3.5.2 Volumetric path tracing

When rendering participating media, we account for volumetric scattering and emission by sampling path vertices in the volume, instead of always continuing at the closest surface. Volumetric path tracing alternates between sampling a distance along the segment and the direction of the next segment. If a path vertex lies within the volume, we sample the next direction proportionally to the phase function. If a surface is intersected, the next direction is sampled according to the BSDF, just as in the surface-only case.

Sampling the *free-flight distance* to the next scattering event is a key challenge. The distance is commonly sampled proportionally to the transmittance. If the medium is *homogeneous*, i.e., the extinction is constant, the transmittance has a closed form:

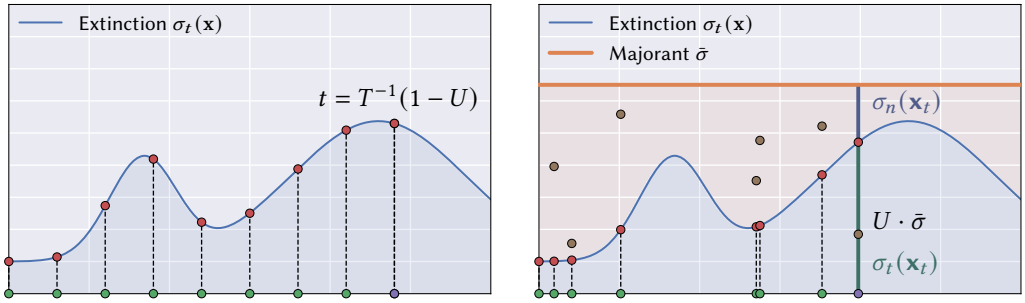
$$\mathrm{T}(\mathbf{x}, \mathbf{x}') = \exp\left(-\int_0^{\|x-x'\|} \sigma_t(\mathbf{x}_t)\,\mathrm{d}t\right) = \exp\left(-\|\mathbf{x}-\mathbf{x}'\|\,\sigma_t\right). \tag{3.53}$$

In that case, the PDF $p(t) = \sigma_t \exp(-t\sigma_t)$ is proportional to the transmittance. The extinction σ_t is simply the normalization constant of this PDF. We can then sample a free-flight distance t using inverse transform sampling:

$$t = -\frac{\log(1-U)}{\sigma_t}, \text{ where } U \sim \mathcal{U}[0,1). \tag{3.54}$$

All results in this thesis use monochromatic extinction coefficients σ_t. If they were spectrally varying, we could for example select a single wavelength for each generated path and sample all the free-flight distances according to the extinction at that wavelength. This would yield a different sampling strategy for each wavelength, which could then be combined using MIS [95, 102].

Evaluating the transmittance and sampling the free-flight distance becomes more difficult if the extinction is spatially varying. If the spatial variation is simple, e.g., it is described by a grid of values, it can still be possible to evaluate the transmittance in closed form. We could then sample a free-flight distance by analytically inverting the CDF. This method is called *regular tracking* and is both computationally expensive and rather inflexible.



(a) Ray marching (b) Delta tracking

Figure 3.15: Illustration of using ray marching **(a)** and delta tracking **(b)** for free-flight distance sampling. The former evaluates the extinction at regular steps, whereas the latter samples interactions until a real scattering event is sampled.

Ray marching. A different approach is *ray marching* [103, 104], which approximates the optical depth integral by evaluating the extinction along regular steps in the medium and using a quadrature rule. The transmittance estimator is then $\exp(-\hat{\tau})$, where $\hat{\tau}$ is the estimated optical depth. The free-flight distance can then be sampled using inverse transform sampling, which means we solve a root-finding problem for t such that $1-U = \mathrm{T}(\mathbf{x}, \mathbf{x}_t)$. This process is shown in a 1D example in Figure 3.15a.

However, even if the optical depth integral is evaluated in an unbiased way [9], the naïve ray marching transmittance estimator will be biased. This is due to the application of the exponential function: $\mathbb{E}\left[\exp(-\hat{\tau})\right] \neq \exp(-\mathbb{E}\left[\hat{\tau}\right])$. Recent methods de-bias the ray marching estimator by using a Taylor expansion of the exponential function or a telescoping series [53, 105]. It turns out that by forming independent estimators of the individual terms in the Taylor series of $\exp(-\tau)$, one can obtain an unbiased estimate of the transmittance even when using ray marching. However, these methods so far only have been used to evaluate the transmittance, but not for free-flight distance sampling.

Delta tracking. Efficient unbiased sampling of the free-flight distance is made possible by the null-scattering formulation (Section 3.4.4) and an algorithm called delta tracking [96, 106]. The key idea is to sample an initial free-flight distance according to the homogeneous upper bound on the extinction. Given such a tentative free-flight sample, the interaction is then probabilistically determined to either be a real or null scattering event. Specifically, the interaction is real if $U \cdot \bar{\sigma} < \sigma_t(\mathbf{x}_t)$, and null otherwise (where again $U \sim \mathcal{U}[0, 1)$). When a null scattering event is sampled, the procedure repeats and the trajectory of the path segment remains unchanged. If a real interaction is sampled, we proceed to sample the scattered direction according to the phase function. This amounts to only evaluating one of the two recursive radiance terms in Equation 3.49.

The procedure is shown in Figure 3.15. We refer to Novák et al. [107] for a thorough discussion of delta tracking and related algorithms. We will discuss how to differentiate this algorithm with respect to the medium parameters in Chapter 5.

Transmittance estimation. When the free-flight distance is sampled proportionally to the transmittance, the transmittance terms along the light path cancel out and do not need to be evaluated explicitly. However, we still need to evaluate the transmittance during next event estimation. Using delta tracking, the transmittance $\mathrm{T}(\mathbf{x}, \mathbf{x}')$ between points $\mathbf{x}$ and $\mathbf{x}'$ can be estimated by sampling the free-flight distance from $\mathbf{x}$ towards $\mathbf{x}'$ and returning 1 if $\mathbf{x}'$ is reached, and 0 otherwise [106]. A better estimator is *ratio tracking* [10], which replaces the probabilistic termination of delta tracking by multiplication with the continuation probability. Mathematically, we substitute a binary random variable by its expectation. The variance can be further reduced by employing a control variate [10]. Alternatively, biased or unbiased versions of ray marching can also be used [9, 105].

4 | Differentiable Rendering

With the necessary background on Monte Carlo methods and rendering algorithms established, we now move on to inverse problems. Inverse rendering techniques solve a general minimization problem of the form

$$\hat{\boldsymbol{\pi}} = \arg\min_{\boldsymbol{\pi}} g(I(\boldsymbol{\pi})). \tag{4.1}$$

The vector of *scene parameters* $\boldsymbol{\pi}$ encodes a representation of the virtual scene, which could include parameters of scene geometry, participating media, BSDFs, virtual cameras and light sources. At this point, we do not assume a specific representation. Some parameters might be stored on grids (e.g., 2D textures), but they could also be scalars or even the weights of a neural network.

The function $I(\boldsymbol{\pi})$ renders an image given the scene parameters. The objective function g is a differentiable function that takes the full image as input. In the simplest case, g computes the mean squared error between $I(\boldsymbol{\pi})$ and a reference image. In more complex cases, it could be an arbitrary black-box nonlinear function. Ultimately, our goal is to minimize this function over a selection of scene parameters. We will write all derivations assuming a single rendered image, but they trivially generalize to multi-view optimizations, where several reference images are provided.

We cannot expect to find a closed-form solution for the general minimization problem in Equation 4.1. Moreover, the dimensionality of the optimization space is vast. Even a simple scene with a low-resolution material (e.g., 768×768 RGB texels) has over 1.7 million parameters that must all be optimized. The optimization algorithms we use therefore need to scale to millions of parameters. Gradient-based optimization methods are commonly used to tackle such high-dimensional problems. The gradient of the objective function provides the steepest descent direction that can guide the optimization algorithms. A central challenge in inverse rendering is efficiently computing the gradient of the objective function. Due to the high-dimensional parameter space and the structure of physically-based rendering algorithms, this is a non-trivial problem.

We first review gradient-based optimization in Section 4.1 and general-purpose computational differentiation methods in Section 4.2, followed by a description of the Mitsuba system. We then discuss the fundamentals of physically-based differentiable rendering. This will also introduce some of the challenges we then address in Chapter 5 and Chapter 6. The chapter concludes with an overview of some of the applications of differentiable rendering.

This chapter is based in part on the related work and background sections of the publications covered in the later chapters [45, 46, 47]. The discussion of AD methods is inspired by the Dr.Jit paper [49].

4.1 Gradient-based optimization

The large number of parameters in inverse rendering problems and the non-linear nature of the objective function can most effectively be tackled using gradient-based optimizers. Starting from an initialization $\boldsymbol{\pi}_0$, *gradient descent* [108] iteratively updates the parameters using the gradient of the objective function:

$$\boldsymbol{\pi}_i = \boldsymbol{\pi}_{i-1} - \lambda \cdot \partial_{\boldsymbol{\pi}} g(I(\boldsymbol{\pi}_{i-1})), \tag{4.2}$$

where $\partial_\pi := \partial/\partial\pi$ is the derivative operator and λ is the step size (or *learning rate*). The step size is oftentimes gradually decreased as the optimization progresses. Given a small enough step size, this algorithm will decrease the objective function value in each iteration until converging to a local minimum. Unless the problem is convex, there is no guarantee for the optimization to reach a global minimum. The optimization might terminate in an undesirable local minium that is far from the optimum. Consistently avoiding such local minima remains a largely open problem in physically-based differentiable rendering.

Adam optimizer. In many applications, optimized parameters are represented in different units and their derivatives exhibit different levels of variance. For example, the gradient of the albedo texture of a material will have significantly less variance than the gradient of the roughness texture. In practice, this requires careful per-parameter tuning of the gradient descent step size. This is difficult to do manually, which has led to the development of several improved gradient-based optimizers [109, 110, 111]. In particular, the *Adam* optimizer [111] has been shown to robustly handle a wide range of optimization problems. It computes a per-variable step size and can deal with extremely noisy gradients. In each iteration, it first updates running estimates of gradient mean m_i and second moment v_i:

$$\begin{aligned} g_i &= \partial_{\boldsymbol{\pi}} g(I(\boldsymbol{\pi}_{i-1})) \\ m_i &= \beta_1 m_{i-1} + (1-\beta_1) g_i \\ v_i &= \beta_2 v_{i-1} + (1-\beta_2) g_i^2 \end{aligned}$$

Then, these estimators are used to compute the actual per-variable updates:

$$\begin{aligned}\widehat{m}_i &= m_i/(1-\beta_1^i)\\ \widehat{v}_i &= v_i/(1-\beta_2^i)\\ \boldsymbol{\pi}_i &= \boldsymbol{\pi}_{i-1} - \lambda \cdot \widehat{m}_i/(\widehat{v}_i + \varepsilon)\end{aligned}$$

Both m_0 and v_0 are set to 0 in the beginning. The parameters β_1 and β_2 control the influence of past gradients. The authors propose to use $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The constant $\varepsilon = 10^{-8}$ prevents division by zero. We use the Adam optimizer for all our optimization results in the following chapters.

4.2 Differentiation methods

While we will later discuss the theoretical implications of differentiating a rendering algorithm, we first focus on the purely computational aspect of the problem. How can we best obtain the derivatives of a complex computation such as path tracing? Conceptually, the simplest approach is to differentiate the computation by hand and implement code for the resulting analytic expressions. This is certainly feasible and has been done for example in the *Redner* differentiable renderer [112] and for the project presented in Chapter 7[1]. However, as the complexity of the rendering algorithm increases, this becomes very tedious and error-prone. It makes adding new types of BSDFs or phase functions laborious and increases the barrier to experimenting with new algorithms. It is therefore desirable to use automated methods, which we review in this section.

4.2.1 Finite differences

The simplest gradient computation method is finite differences (FD). It numerically approximates the derivative of a scalar function $f : \mathbb{R} \to \mathbb{R}$ as:

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}, \tag{4.3}$$

where h is a small step size. A commonly used variant of FD are central differences:

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}. \tag{4.4}$$

[1]This project was developed before the ones presented in Chapter 5 and Chapter 6.

Finite differences are biased since they evaluate a blurred version of the true derivative [53]:

$$\frac{f(x+h)-f(x)}{h} = \frac{1}{h}\int_x^{x+h} f'(t)\,\mathrm{d}t = \int_{-\infty}^{\infty} K(t-x)f'(t)\,\mathrm{d}t, \tag{4.5}$$

where $K(x) = \frac{1}{h}\mathbb{1}_{[0,h]}(x)$ is a box kernel. The effect of this blur kernel becomes negligible as h decreases, and FD is commonly used as a reference to validate other gradient computation methods. Misso et al. [53] further show that by progressively reducing the step size h one can construct an unbiased gradient estimator. However, this comes at the significant cost of evaluating the FD estimator many times.

It is straightforward to apply finite differences to a renderer by generating the image once with the original and once with the offset parameter. When using a Monte Carlo renderer, the evaluation of f will be noisy. If $f(x+h)$ and $f(x)$ are evaluated independently, the FD estimator requires an enormous number of Monte Carlo samples to converge. The issue can easily be resolved by using the same random number generator seed for both evaluations. The correlation of the two estimators then causes a significant part of the variance to cancel out.

Fundamentally, the main problem of finite differences is that they cannot scale to functions with many input parameters. For inverse rendering, we would need to render the image twice for *each* parameter. This is completely impractical for most real use cases. An alternative is *simultaneous perturbation* [113], which is a stochastic estimator that estimates high-dimensional gradients by simultaneously offsetting all parameters. For $f\colon \mathbb{R}^n \to \mathbb{R}$, the i'th component of the gradient vector can be estimated as:

$$\partial_{\mathbf{x}_i} f(\mathbf{x}) \approx \frac{f(\mathbf{x}+h\Delta)-f(\mathbf{x}-h\Delta)}{2h\Delta_i}, \tag{4.6}$$

where Δ is a randomly sampled vector and Δ_i is its i'th component. This approach replaces the scalability issue with additional variance, which harms optimization performance. Unfortunately, this and related *derivative-free* optimization methods cannot compete with gradient descent using the true infinitesimal gradient.

4.2.2 Automatic differentiation

Instead of finite differences or analytic gradients, we typically want to use some form of *automatic differentiation* (AD). These methods compute gradients automatically by leveraging the chain rule to decompose the derivative of a computation into a Jacobian product:

$$\partial_{\mathbf{x}}\left[g(f(\mathbf{x}))\right] = \mathbf{J}_g(f(\mathbf{x}))\,\mathbf{J}_f(\mathbf{x}), \tag{4.7}$$

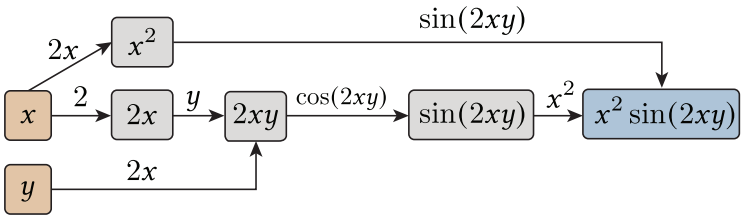


Figure 4.1: Example computation graph corresponding to the expression $x^2 \sin(2xy)$. The edge weights are the derivative of the operation applied to the input node.

where $\mathbf{J}_g(f(\mathbf{x}))$ is the Jacobian of g evaluated at $f(\mathbf{x})$.

This principle enables scalable derivative computation that supports an arbitrary number of input variables. Automatic differentiation was initially introduced between the 1950s and 1970s [114, 115, 116] and then later gained significant traction thanks to its application to training neural networks [117]. The following overview of AD discusses the basic principles and outlines some of the main constraints posed by the inverse rendering problem. A more in-depth discussion of AD can be found in the book by Griewank and Walther [118].

Computation graphs. The central idea is to think of a given computation as *graph* of operations. The individual operations are nodes and the derivatives of individual steps are assigned to the graph's edges. For example, consider the following expression:

$$x^2 \sin(2xy). \tag{4.8}$$

In a computer program, we could implement the evaluation of this expression as a sequence of steps:

```
a = 2 * x
b = a * y
c = sin(b)
d = x * x
e = c * d
```

The corresponding computation graph is shown in Figure 4.1. The weight of an edge $a \to b$ between nodes a and b is the derivative $\partial b / \partial a$. An implementation of AD can compute these edge weights during the forward computation. The forward computation will henceforth also be referred to as *primal* computation. Many quantities used in the edge weights are redundant with the primal computations. The stored weights then allow to efficiently compute variable gradients. The stored graph of operations is sometimes also referred to as *tape* or *Wengert tape* [115].

Forward-mode differentiation. A key choice in AD algorithms is the *directionality* of the gradient computation. The stored computation graph can be traversed either in forward or reverse direction. If the computation has a single differentiable input variable, but many outputs, it is efficient to evaluate gradients from the variable to the output in *forward* direction. Mathematically, the differentiation turns into a series of *Jacobian-vector products* (JVP). For a function $\mathbf{y} = f(\mathbf{x})$, forward-mode AD computes the output gradient $\delta_\mathbf{y}$ as the product of the Jacobian $\mathbf{J}_f$ with the input gradient $\delta_\mathbf{x}$:

$$\delta_\mathbf{y} = \mathbf{J}_f \delta_\mathbf{x}. \tag{4.9}$$

Here and in the following, we use δ to denote vectors that are inputs and outputs of Jacobian products.

We can apply forward-mode AD to differentiate the output of Equation 4.8 with respect to x. We first initialize a variable $\delta x = 1$ and then traverse the graph in Figure 4.1 from left to right, in each step multiplying the derivative value by the stored edge weights. This results in the following sequence of operations:

```
δx = 1
δa = δx * 2
δb = δa * y
δc = δb * cos(2 * x * y)
δd = δx * 2 * x
δe = δc * x * x + δd * sin(2 * x * y)
```

In the end, the variable δe contains the derivative $\partial f / \partial x$. Using the computation graph, we can differentiate an arbitrary sequence of elementary operations, without explicitly computing and storing the full Jacobian matrix of the program.

Forward-mode differentiation can be formalized by using *dual numbers*. Similar to a complex number, a dual number $a + \epsilon b$ consists of a real part a and a dual part b. The symbol ϵ satisfies $\epsilon^2 = 0$ and hence the product of two dual numbers is $(a + \epsilon b)(c + \epsilon d) = ac + (ad + bc)\epsilon$. For a function f, we can use a Taylor expansion around a to see that $f(a + \epsilon b) = f(a) + \epsilon b f'(a)$. All the higher-order terms contain a factor ϵ^2 and vanish. Therefore, the derivative $f'(a)$ can be obtained by extracting the dual part after evaluating $f(a+\epsilon)$. In an implementation, forward-mode AD can be realized using *operator overloading*. We can compute gradients by implementing a dual number type, where basic operators are overloaded to compute the corresponding derivatives. This means that we compute both the used edge weights and their application to the gradient variables alongside the primal computation, without storing a computation graph.

The main issue with forward-mode differentiation is that the entire derivative computation needs to be carried out separately for *each* input variable. Similar to finite differences, this does not scale to the large number of parameters encountered in inverse

rendering. The use of forward-mode AD is therefore restricted to debugging and validation of differentiable rendering algorithms, where it can be used to visualize derivatives of individual pixels with respect to a single input variable. Researchers working on rendering algorithms often debug methods by inspecting rendered images. Some of that intuition can be applied to derivatives by analyzing such forward-mode gradient images.

Reverse-mode differentiation. The solution to this limitation is to traverse the computation graph in *reverse* order. Given a sequence of operations, reverse-mode AD will start by evaluating the chain rule for the last operation and proceed toward the input of the algorithm. Mathematically, reverse-mode computation evaluates *vector-Jacobian products* (VJP) from the output end of the computation. For a function f, it evaluates

$$\delta_{\mathbf{x}} = \delta_{\mathbf{y}}^T J_f. \tag{4.10}$$

The advantage of this evaluation order is that the gradient computation no longer needs to be duplicated for each input variable. For the previous example, reverse-mode AD can simultaneously compute the gradient with respect to x *and* y:

```
δe = 1 # Initialize the output gradient
δd = δe * sin(2 * x * y)
δc = δe * x * x
δb = δc * cos(2 * x * y)
δa = δb * y
# Accumulate x gradient
δx = δd * 2 * x + δa * 2
# Accumulate y gradient
δy = δa * 2 * x
```

The x gradient is identical to the forward-mode version, but we now got the y gradient at almost no extra cost. This reverse-mode gradient computation is also called *backpropagation*.

While simple for our example program, reverse-mode AD is generally more difficult to implement than forward-mode. Since it propagates gradients *opposite* to the primal program's computation order, it requires storing some of the edge weights of the computation graph in memory to be able to run efficiently. If we naïvely implemented reverse-mode AD without storing any of the edge weights, each gradient computation step would require re-running the primal computation up to the current node in the graph. The complexity would be quadratic in the number of computation steps, which makes such an approach unusable in practice. On the other hand, storing all edge weights of the graph would easily exceed the available system memory for most complex computations.

The standard remedy for this issue is to only store the program state at a sparse set of *checkpoints* [119, 120]. Between checkpoints, derivative terms are recomputed during the graph traversal. The number and placement of checkpoints control the tradeoff between memory use and recomputation. This can be done by manually defining checkpoints or using some heuristics in the AD system. Another aspect of this problem is the granularity of the AD graph. For commonly used complex operations, e.g., convolution layers in a neural network, it can be worthwhile to implement the vector-Jacobian product manually, rather than relying on AD to construct an efficient implementation from elementary operations. In the context of physically-based differentiable rendering, we will see that using checkpointing is insufficient and propose a rendering-specific solution in Chapter 5.

Tracing AD. There are a large number of design choices when building a reverse-mode AD implementation. A common pattern is to *trace* computations using operator overloading. Every operation in the traced program will add a node to the AD graph. A tracing framework is generally not aware of higher-level control structures such as loops, if-conditions and function calls. Any loops will be unrolled and function calls will be inlined.

Popular machine learning frameworks such as TensorFlow [121] and PyTorch [122] employ this approach and are designed for programs consisting of a relatively small number of neural network layers (usually less than 100) without complex control flow. The individual neural network operations are implemented on top of highly optimized GPU kernels (e.g., using cuDNN [123]). Each operation is equipped with a custom forward and backward gradient propagation implementation. The frameworks then simply connect up a few dozen of these individual operations. By default, the result of each operation is written to memory and cached for the reverse-mode gradient computation. This uses a significant amount of memory, but since the number of operations is low this scales reasonably well to machine learning workloads.

However, complex programs such as a renderer consist of thousands of elementary operations that need to be evaluated and differentiated efficiently. In that case, writing out all intermediate results is infeasible and we need to use checkpointing. High-performance reverse-mode AD of large programs further requires some form of a compiler to *fuse* several operations into a single program or GPU kernel. By fusing operations into a single kernel, temporary variables can use registers and we can reduce costly reads and writes to (large) temporary memory buffers. For example, JAX [124] builds on the XLA compiler [125] to convert a program trace into a set of efficient kernels.

Source transformation. While tracing is commonly used, it has the disadvantage of not supporting control flow such as loops. The trace of a program can be much larger than the original program, which makes the compilation expensive and prevents the use of sophisticated optimizations. The alternative is to use a *source transformation* approach, in which a derivative program is compiled by analyzing the control flow of the primal program. Tapenade [126] compiles derivative code for C programs, while Zygote [127] works on programs in single-static assignment form. Similarly, Enzyme [128] computes derivatives of optimized LLVM IR [129]. Stalin∇ [130] enables reverse-mode AD support in a functional programming context. Source transformation and tracing approaches are not mutually exclusive: several systems combine high-level tracing with a limited version of source transformation AD.

Differentiable programming in computer graphics. All our algorithms are implemented on top of Mitsuba [50] and Dr.Jit [49], which we discuss in Section 4.3. Several other AD systems tailored to computer graphics applications have been proposed. The recent TensorRay [131] system implements a JIT compiler supporting reverse-mode AD of differentiable renderers. The Taichi [132, 133] language enables writing efficient, differentiable physics simulations in Python. It is particularly efficient at handling computation on sparse grids, as commonly used for fluid simulation and can fuse a large number of operations into a single GPU kernel. Differentiable Halide [134, 135] allows backpropagating gradients through high-performance image processing code. Halide is a domain-specific language for image processing that decouples the specification of algorithms from their execution schedule, which enables writing high-performance, differentiable CPU and GPU code. DiffVG [136] is a framework for differentiable rasterization of vector graphics.

4.2.3 Reversible computation

Instead of relying on automatic differentiation, we could try to exploit domain knowledge to come up with more efficient gradient estimators. At its core, a large part of differentiable rendering research, including some of the methods presented in this thesis, falls into this category. There is a rich history of custom differentiation methods for a range of applications.

A key difficulty for reverse-mode differentiation are long-running loops that require trading off storage and expensive recomputation. The simplest solution is to store the state of the loop variables after each iteration, but this has a very high memory use and

performs poorly if the number of loop iterations is unbounded.

However, there is a way around this problem for certain special cases: If the loop iteration is *reversible*, the required intermediate state can be reconstructed during the reverse-mode gradient accumulation. The gradient computation traverses the loop state from iteration N to iteration 1. At step i, we can recover the state at step $i-1$ by applying the inverse loop iteration. The adjoint sensitivity method [137] from the area of optimal control is the classical example of this idea: its primal phase integrates an ordinary differential equation up to a certain point in time. Differentiation then follows the same trajectory in reverse by taking negative timesteps starting from the endpoint. This idea has been used to reduce the costs of differentiation in fluid dynamics [138], robotics [139], and reversible residual networks in machine learning [140, 141]. Invertible neural network architectures can further be employed to learn parametric sampling methods [142, 143].

In Chapter 5, we introduce a novel differentiation method that is similar to reversible programming. Instead of explicitly running an inverse program, it relies on the invertibility of the (low-dimensional) Jacobian relating adjacent scattering interactions. In Chapter 7 we use invertibility to differentiate through a novel transmittance model, similar to concurrent work on differentiable volume rendering for scientific visualization [144].

4.3 Mitsuba and Dr.Jit

The efficient implementation of reverse-mode AD for large programs is also a compilation problem. In this section, we briefly describe Mitsuba 3 [50], which is the differentiable rendering system developed in our lab alongside the research on algorithms. It is implemented on top of Dr.Jit [49], a just-in-time compiler for differentiable rendering. This stack represents the evolution of the Mitsuba 2 renderer [48, 145]. The aim of this section is to outline some of the requirements for physically-based differentiable rendering systems.

Mitsuba offers a variety of plugins implementing different types of shapes, BSDFs, light sources, sensors and phase functions. Scenes are described as XML files or Python dictionaries, specifying a combination of different plugins and their parameters. For a detailed description of the system features, see Mitsuba 3's documentation[2]. The system is designed to support the rapid exploration of physically-based differentiable rendering

[2] `https://mitsuba.readthedocs.io/`

algorithms. This is enabled by a just-in-time (JIT) compiler called Dr.Jit which compiles algorithms implemented in a Python frontend. The compilation by Dr.Jit is essential to achieve competitive performance, and all functionality in Mitsuba is implemented on top of it.

Dr.Jit compiler. At the lowest level, the Dr.Jit compiler is built around 1D arrays that process independent elements in parallel (e.g., often each element corresponds to a separate Monte Carlo sample). These arrays support various mathematical operations and can be combined to form structures of arrays (e.g., a 3D vector). Any operation on these arrays will be added to a trace of operations. This trace is unrelated to AD and is simply a sequence of operations. The tracing is *uninterrupted* and no actual computation is carried out until the user requests it. Crucially, Dr.Jit supports tracing virtual function calls, loops and calls to ray tracing acceleration libraries (concretely, Embree [146] and OptiX [147]). These operations are preserved in the trace (e.g., loops remain loops and are not unrolled). The uninterrupted tracing allows fusing the entire rendering process into a single GPU or CPU kernel. This *megakernel* stores intermediate values in registers and only accesses system memory for inputs and outputs. This is not only useful for AD, but also makes it possible to implement a fast, memory-efficient GPU path tracer in Python. On the contrary, a *wavefront* implementation stores temporary results in large buffers in system memory. We found that megakernels are often faster than their wavefront counterparts, especially when compiling for CPUs, where the system memory's throughput is much lower than on a GPU.

The compiler implements standard optimizations like constant propagation and common subexpression elimination. It further performs optimizations specific to virtual function calls, e.g., it propagates constants across virtual function call boundaries. Since tracing happens every time we render an image, it needs to be fast and the compiler does not feature any complex multi-pass optimizations. The trace of operations is then handed off to OptiX [147] for GPU compilation or LLVM [129] CPU compilation. These frameworks take care of challenging low-level problems like register allocation.

Tracing AD. This JIT compiler naturally combines well with a second level of tracing that constructs an AD graph. Traversing the AD graph in forward or reverse direction then simply emits instructions to the JIT compiler. Dr.Jit supports fine-grained control over how gradients are propagated in the AD graph. For example, it allows temporarily suspending gradient computation for all variables not in a user-defined group. Virtual function calls support gradient computation as well. The gradient propagation

then simply performs a virtual function call on derivative functions (instead of inlining everything). Differentiating through polymorphism is important, as rendering mostly accesses scene parameters through virtual function calls (e.g., when evaluating different BSDFs). Note that some problems are deliberately left to the user: Dr.Jit does not differentiate loops or ray tracing calls. However, it facilitates the implementation of custom gradient computation algorithms. The differentiation approach that we use in rendering is a combination of specialized algorithms, hand-written derivatives, automatic differentiation and source transformation. The design of Dr.Jit was partially driven by the needs of the path replay backpropagation algorithms that we introduce in Chapter 5. That algorithm differentiates a rendering algorithm by only using AD locally inside of the path tracer loop, without the need to build an AD graph across iterations.

4.4 Differentiable Monte Carlo rendering

With this, we now turn to the problem of differentiating Monte Carlo rendering algorithms. As stated earlier, our goal is to solve problems of the form:

$$\hat{\boldsymbol{\pi}} = \arg\min_{\boldsymbol{\pi}} g(I(\boldsymbol{\pi})), \tag{4.11}$$

where g is an image-based objective function. In the following, we will simplify the notation by considering only the intensity I of a single pixel j and one differentiable parameter π. The derivations generalize to differentiable rendering of RGB images and multiple parameters. The theoretical framework also applies to more general objective functions $g(I(\boldsymbol{\pi}), \boldsymbol{\pi})$ that contain additional loss terms defined on the parameters (e.g., regularization or priors). We will focus on the gradients related to the rendered image, as additional terms can be handled by standard AD.

Objective function gradient. Using the simplified notation, our goal is to compute the derivative $\partial_\pi g(I(\pi))$. The chain rule allows writing this term as:

$$\partial_\pi g(I(\pi)) = g'(I(\pi))\ \ \partial_\pi I(\pi), \tag{4.12}$$

where g' is the derivative of the objective function. We further declutter the notation by dropping the explicit dependency of I on π from now on. As explained in Chapter 3, we will use Monte Carlo integration to estimate I. If we replace I with a Monte Carlo estimator $\hat{I}$ in the equation above and take the expected value we get

$$\begin{aligned}\mathbb{E}\left[\partial_\pi g(\hat{I})\right] = \mathbb{E}\left[g'(\hat{I})\ \partial_\pi \hat{I}\right] &= \mathbb{E}\left[g'(\hat{I})\right]\mathbb{E}\left[\partial_\pi \hat{I}\right] + \operatorname{Cov}\left[g'(\hat{I}), \partial_\pi \hat{I}\right] \\ &= \mathbb{E}\left[g'(\hat{I})\right]\partial_\pi \hat{I} + \operatorname{Cov}\left[g'(\hat{I}), \partial_\pi \hat{I}\right] \neq \partial_\pi g(I).\end{aligned} \tag{4.13}$$

Generally, this will not result in an unbiased estimator of the true objective function gradient. Without any problem-specific knowledge, we can assume gradient descent to work best if our estimate of $\partial_\pi g(I)$ is as accurate as possible.

The bias arises from two sources and can partially be eliminated. First, $g'(\hat{I})$ and $\partial_\pi \hat{I}$ are correlated, which produces the covariance term. We can get rid of this bias by using two independent estimators: a primal estimator $\hat{I}^p$ used to evaluate g' and a separate gradient estimator $\partial_\pi I^a$ [148, 149]. Using these decorrelated estimators the covariance term disappears:

$$\mathbb{E}\left[g'(\hat{I}^p)\ \partial_\pi \hat{I}^a\right] = \mathbb{E}\left[g'(\hat{I}^p)\right] \partial_\pi \hat{I}^a. \tag{4.14}$$

We will in the following always use a separate set of samples for primal and gradient estimators. In practice, this means we render two images using different random number seeds.

The second source of bias is the estimation of $g'(I)$. We can observe that if $g(I) = (I - I^{Ref})^2$ is the squared difference to a reference value I^{Ref}, the derivative function is linear: $g'(I) = 2(I - I^{Ref})$. Thus, any unbiased estimator of the pixel color I produces an unbiased estimator of $g'(I)$.

Unfortunately, this does not hold for arbitrary objective functions. For example, the gradient of the L_1 loss cannot easily be estimated in an unbiased way. The gradient of $g(I) = |I - I^{Ref}|$ is $g'(I) = \text{sign}(I - I^{Ref})$. Naïvely estimating this gradient yields the following expected value:

$$\mathbb{E}\left[g'(\hat{I})\right] = \mathbb{E}\left[\text{sign}(\hat{I} - I^{Ref})\right] = P(\hat{I} - I^{Ref} \geq 0) - P(\hat{I} - I^{Ref} < 0). \tag{4.15}$$

This will be a value in $[-1, 1]$ and typically does not equal the true sign, which is either -1 or +1. However, the sign of the expected value will be correct and the gradient will therefore in expectation point in the right direction. The situation is less clear when using more complex loss functions that might mix information from different pixels (e.g., a perceptual neural loss [150]). In this thesis, we use variants of L_1 and L_2 losses and did not experiment with more complex loss functions. Fortunately, even with a more complex objective function, the gradient estimator remains consistent and the bias decreases as the number of samples is increased.

Detached estimator. The remaining challenge is to estimate $\partial_\pi I$ itself. Mathematically, we need to differentiate a parameter-dependent, high-dimensional integral over light paths:

$$\partial_\pi I = \partial_\pi \int_{\mathcal{P}} f(\mathbf{x}, \pi)\, \mathrm{d}\mathbf{x}, \tag{4.16}$$

where f is the parameter-dependent image contribution function. We omit its usual pixel index subscript, as all the following derivations focus on a single pixel. If f does not contain parameter-dependent discontinuities, we can directly estimate this derivative using Monte Carlo integration. The derivative operator can be moved into the integral:

$$\partial_\pi \int_{\mathcal{P}} f(\mathbf{x}, \pi)\, \mathrm{d}\mathbf{x} = \int_{\mathcal{P}} \partial_\pi f(\mathbf{x}, \pi)\, \mathrm{d}\mathbf{x} \approx \frac{1}{N} \sum_{i=1}^{N} \frac{\partial_\pi f(\mathbf{x}_i, \pi)}{p(\mathbf{x}_i, \pi)}. \tag{4.17}$$

For this estimator, we need to differentiate the evaluation of f. We do not have to differentiate the sampling process that produces $\mathbf{x}_i$ or the corresponding PDF $p(\mathbf{x}_i)$. We call this estimator *detached* since both sampling and PDF evaluation are detached from the differentiation process. This is the most commonly used estimator in differentiable rendering [45, 48, 148, 151]. If f contains π-dependent discontinuities, additional precautions are required (see Section 4.5). Similarly, if the path space $\mathcal{P}$ is parameter-dependent, we need to account for changes in its geometry [152] or switch to a parameterization of the integration domain that is independent of π. Since the path space is defined as the cartesian product of scene surfaces, any parameter influencing the shape or position of surfaces will modify it.

The derivation here only considers a single scene parameter, but we are usually concerned with evaluating the gradient of many parameters at once. For efficiency, the same set of sampled light paths can be used to evaluate all parameter derivatives simultaneously. The mathematical formulation here does not address practical considerations of evaluating all these derivative integrals. In Chapter 5 we will further investigate how to efficiently compute derivatives of f, in particular in the presence of long light paths.

Attached estimator. While conceptually simple, the detached estimator does not handle all potential use cases. In particular, it does not support perfectly specular BSDFs. Such BSDFs are delta functions, which do not yield valid derivatives. The solution to this problem is to also differentiate the BSDF sampling process. By doing so, we switch from differentiating the integrand by itself to differentiating the ratio of integrand to PDF. This avoids having to differentiate the delta function of the specular BSDF, as it cancels out with the sampling density.

Differentiating the sampling process can be interesting beyond perfectly specular surfaces. Many of the sampling steps in a Monte Carlo renderer are highly scene-dependent. For example, the roughness parameter of a microfacet BSDF will affect the sampling of the scattered direction. This and other sampling methods usually transform a set of uniformly distributed random numbers to the desired target distribution,

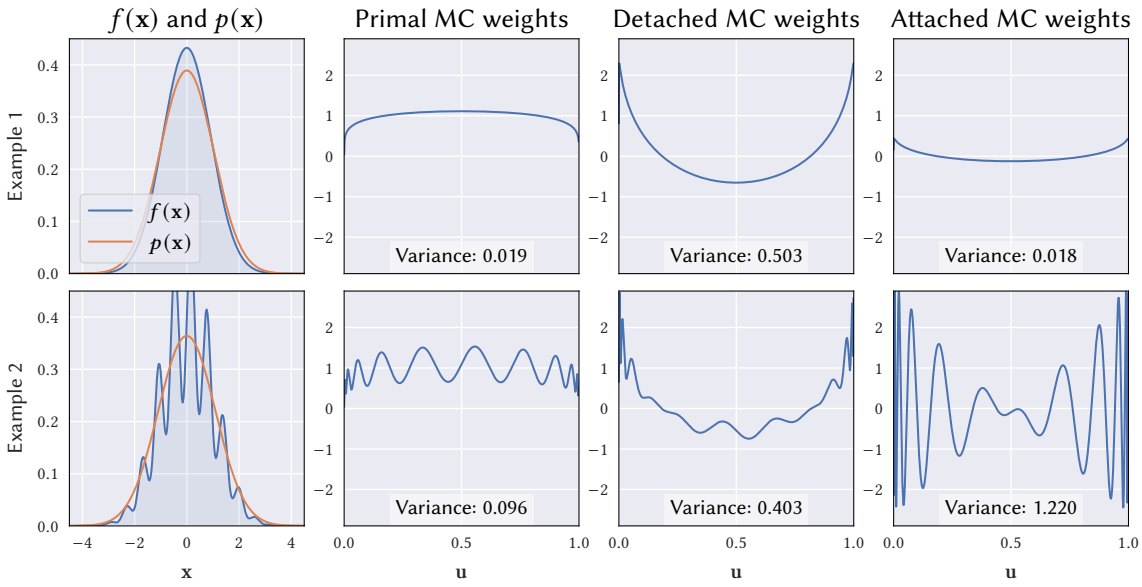


Figure 4.2: We plot the Monte Carlo sample weights for the primal, detached and attached estimators for two example problems. In both problems, integrand and sampling density are based on a Gaussian function and differentiated with respect to its variance. The sampling weights are plotted over the primary sample space $[0, 1]$. **Top row:** the sampling density is close to proportional to the integrand. In that case, the attached estimator achieves a variance close to the primal estimator. **Bottom row:** the integrand is multiplied by a parameter-independent sine wave. The additional motion due to the attached sampling strategy causes additional variance, while the detached estimator performs similarly to the first example.

e.g., using inverse transform sampling [71]. We can interpret this transformation as a reparameterization of the original integral. This poses the question of when we should differentiate before or after applying this reparameterization of the integrand. Because the sampling strategy may potentially produce different distributions depending on the parameter π, samples from the latter approach can be understood to smoothly follow the motion of the underlying sampling strategy with respect to perturbations in π. This influences the variance properties of the resulting estimators.

Formally, sampling strategies can be understood as a change of variables to new coordinates $\mathbf{u} \in \mathcal{U}$ parameterizing the integration domain $\mathcal{P}$ via a mapping $T\colon \mathcal{U} \to \mathcal{P}$, where $\mathcal{U} = [0, 1]^n$ is a unit-sized hypercube of suitable dimension. The space $\mathcal{U}$ is called the *primary sample space* [153]. The mapping $\mathbf{x} = T(\mathbf{u})$ is constructed from a target density $p(\mathbf{x})$ so that its Jacobian determinant satisfies $|\mathbf{J}_T(\mathbf{u})| = p(\mathbf{x})^{-1}$. The reparameterized integral then takes the form

$$I = \int_{\mathcal{P}} f(\mathbf{x}, \pi)\, \mathrm{d}\mathbf{x} = \int_{\mathcal{U}} f(T(\mathbf{u}, \pi), \pi) |\mathbf{J}_T(\mathbf{u}, \pi)|\, \mathrm{d}\mathbf{u} = \int_{\mathcal{U}} \frac{f(T(\mathbf{u}, \pi), \pi)}{p(T(\mathbf{u}, \pi), \pi)}\, \mathrm{d}\mathbf{u}. \tag{4.18}$$

This formulation is called *attached*, since samples geometrically follow the motion of

$T(\mathbf{u}, \pi)$ with respect to perturbations of π. Similar to before, we can build an estimator of the derivative by applying Monte Carlo integration:

$$\partial_\pi I = \int_{\mathcal{U}} \partial_\pi \left[\frac{f(T(\mathbf{u}, \pi), \pi)}{p(T(\mathbf{u}, \pi), \pi)} \right] \mathrm{d}\mathbf{u} \approx \frac{1}{N} \sum_{i=1}^{N} \partial_\pi \left[\frac{f(T(\mathbf{u}_i, \pi), \pi)}{p(T(\mathbf{u}_i, \pi), \pi)} \right]. \tag{4.19}$$

The attached estimator is primarily useful for perfectly specular surfaces, but Zeltner et al. [68] also showed that it can produce lower variance than the detached version for derivatives of BSDFs with low roughness. On the other hand, the additional motion of the samples might introduce more variance in the evaluation of other terms in the integrand. Figure 4.2 shows two example integrands on which we compare attached and detached estimators. In the first example, the sampling distribution closely matches the integrand and the attached estimator can outperform the detached version. However, in the second example, the differentiated parameter does not affect the high-frequency variation and hence the motion of the samples causes additional variance for the attached estimator.

Finally, the attached estimator is more difficult to use as in practice it requires handling discontinuities in the sampling function T. Examples of such discontinuities are discrete sampling decisions such as in delta tracking or discontinuities due to sampled rays hitting different objects as π changes [48, 68, 154]. Zeltner et al. [68] propose to adaptively switch between attached and detached estimators to circumvent some of these issues.

Related formulations. As mentioned before, differentiable Monte Carlo estimators have been used in other disciplines as well. We show how several formulations from other fields can be reduced to our detached and attached estimators.

Differentiable Monte Carlo has for example been used to train certain generative neural network models [155, 156]. During training, the goal is to optimize the parameters of the neural network to reproduce the distribution of samples in a data set. This requires differentiating an expected value with respect to the parameters π of a distribution with PDF $p(X, \pi)$. Paisley et al. [155] proposed to reformulate the derivative of an expected value as:

$$\begin{aligned} \partial_\pi \, \mathbb{E}_{X \sim p(X,\pi)} \left[f(X) \right] &= \partial_\pi \int_{\mathcal{X}} f(\mathbf{x}) p(\mathbf{x}, \pi) \, \mathrm{d}\mathbf{x} = \int_{\mathcal{X}} f(\mathbf{x}) p(\mathbf{x}, \pi) \partial_\pi \log p(\mathbf{x}, \pi) \, \mathrm{d}\mathbf{x} \\ &= \mathbb{E}_{X \sim p(X,\pi)} \left[f(X) \partial_\pi \log(p(X, \pi)) \right], \end{aligned} \tag{4.20}$$

where the second equality moves the derivative operator into the integral and uses $\partial_\pi \log(p(\mathbf{x}, \pi)) = \partial_\pi p(\mathbf{x}, \pi) / p(\mathbf{x}, \pi)$. The second expected value can then be estimated by

sampling X according to p. It is easy to see that this is nothing else than the previously introduced detached estimator applied to the integrand $f(x)p(x,\pi)$:

$$\mathbb{E}_{X\sim p(X,\pi)}\left[f(X)\partial_\pi \log(p(X,\pi))\right] \approx \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}_i)\partial_\pi \log(p(\mathbf{x}_i,\pi)) = \frac{1}{N}\sum_{i=1}^{N}\frac{f(\mathbf{x}_i)\partial_\pi p(\mathbf{x}_i,\pi)}{p(\mathbf{x}_i,\pi)}. \tag{4.21}$$

Similar to rendering, this detached estimator can suffer from high variance. In variational autoencoder [156] training, this issue is mitigated using the so-called *reparameterization trick*:

$$\partial_\pi \mathbb{E}_{X\sim p(X,\pi)}\left[f(X)\right] = \partial_\pi \mathbb{E}_{U\sim q(U)}\left[f(T(U,\pi))\right] = \mathbb{E}_{U\sim q(U)}\left[\partial_\pi f(T(U,\pi))\right], \tag{4.22}$$

where q is a suitable parameter-independent distribution and T transforms a sample U into a sample of the distribution $p(X,\pi)$. This reduces the variance of the gradient estimator and corresponds to using an attached estimator. Similar ideas were also present in early work on reinforcement learning [157].

In physics, de Lataillade et al. [158] discussed tradeoffs of different estimators, similar to the work by Zeltner et al. [68]. They suggest an alternative detached estimator, that works if we can only access the sample weight $w(\mathbf{x}_i,\pi) \coloneqq f(\mathbf{x}_i,\pi)/p(\mathbf{x}_i,\pi)$, but not the integrand f itself:

$$\partial_\pi \int_{\mathcal{X}} f(\mathbf{x})\,\mathrm{d}\mathbf{x} \approx \frac{1}{N}\sum_{i=1}^{N} \partial_\pi w(\mathbf{x}_i,\pi) + w(\mathbf{x}_i,\pi)\frac{\partial_\pi p(\mathbf{x}_i,\pi)}{p(\mathbf{x}_i,\pi)}. \tag{4.23}$$

They motivate this estimator by physical simulations that might not provide an explicit integral formulation. An example of this used to be delta tracking, where an integral formulation was only proposed a few years ago [97]. Mathematically, the formulation is equivalent to the detached estimator, since substituting the definition of w yields:

$$\begin{aligned}\partial_\pi w(\mathbf{x},\pi) + w(\mathbf{x},\pi)\frac{\partial_\pi p(\mathbf{x},\pi)}{p(\mathbf{x},\pi)} &= \frac{\partial_\pi f(\mathbf{x},\pi)p(\mathbf{x},\pi) - f(\mathbf{x},\pi)\partial_\pi p(\mathbf{x},\pi)}{p(\mathbf{x},\pi)^2} + \frac{f(\mathbf{x},\pi)\partial_\pi p(\mathbf{x},\pi)}{p(\mathbf{x},\pi)^2} \\ &= \frac{\partial_\pi f(\mathbf{x},\pi)}{p(\mathbf{x},\pi)}.\end{aligned} \tag{4.24}$$

In summary, it seems that the main classes of differentiable Monte Carlo estimators are indeed detached and attached, and various alternative methods can be reduced to these.

4.5 Discontinuities

So far, all derivations assumed f to be free of parameter-dependent discontinuities. This allowed moving the derivative operator inside the integral. However, if the integrand contains parameter-dependent discontinuities, we cannot swap derivative and integral operator:

$$\partial_\pi \int_{\mathcal{P}} f(\bar{\mathbf{x}}, \pi)\, \mathrm{d}\bar{\mathbf{x}} \neq \int_{\mathcal{P}} \partial_\pi f(\bar{\mathbf{x}}, \pi)\, \mathrm{d}\bar{\mathbf{x}}. \tag{4.25}$$

For example, if π controls the position of an object, it will affect the position of discontinuities in the visibility function. Discontinuities are a primary concern when reconstructing object shape or pose, which are important use cases for inverse rendering methods. Note that only parameter-dependent discontinuities are problematic. Discontinuities in the integrand that are unaffected by π do not cause any issues. Even in the presence of parameter-dependent discontinuities, the image formation process inherently remains differentiable. It simply becomes more difficult to estimate its derivative using Monte Carlo integration. In this section, we will discuss methods to handle these geometric discontinuities in physically-based differentiable rendering.

4.5.1 Reynolds transport theorem

We first consider the simple problem of differentiating a one-dimensional integral. In that case, the *Leibniz integral rule* states:

$$\partial_\pi \int_{a(\pi)}^{b(\pi)} f(x, \pi)\, \mathrm{d}x = \int_{a(\pi)}^{b(\pi)} \partial_\pi f(x, \pi)\, \mathrm{d}x + \partial_\pi b(\pi) f(b(\pi), \pi) - \partial_\pi a(\pi) f(a(\pi), \pi), \tag{4.26}$$

where $a(\pi)$ and $b(\pi)$ are parameter-dependent integration boundaries and f is a scalar differentiable function with continuous partial derivative $\partial_\pi f$. If the integration bounds are not parameter-dependent, we simply move the derivative operator into the integral and all other terms become zero. However, if the boundary moves with π, we need to specifically evaluate the two boundary terms.

This idea can be generalized to discontinuous integrands in higher dimensions. The *Reynolds transport theorem* [159, 160] implies that the derivative of an integral containing discontinuities can be decomposed into an *interior* and a *boundary* integral:

$$\partial_\pi \int_{\mathcal{X}(\pi)} f(\mathbf{x}, \pi)\, \mathrm{d}\mathbf{x} = \underbrace{\int_{\mathcal{X}(\pi)} \partial_\pi f(\mathbf{x}, \pi)\, \mathrm{d}\mathbf{x}}_{\text{Interior integral}} + \underbrace{\oint_{\Gamma(\pi)} \Delta f(\mathbf{x}, \pi) \langle \partial_\pi \mathbf{x}, \mathbf{n} \rangle\, \mathrm{d}\mathbf{x}}_{\text{Boundary integral}}, \tag{4.27}$$

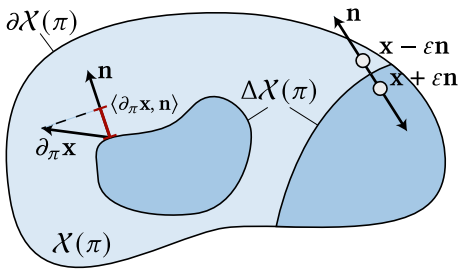


Figure 4.3: This figure illustrates the terms used in the Reynolds transport theorem (Equation 4.27).

where $\mathcal{X}(\pi)$ is the integration domain and $\Gamma(\pi) \subset \mathcal{X}(\pi)$ is the union of the domain boundary $\partial\mathcal{X}(\pi)$ and the set of (parameter-dependent) interior discontinuities $\Delta\mathcal{X}(\pi)$. The boundary integral integrates the dot product of the normal direction $\mathbf{n}$ at $\mathbf{x}$ and the motion of the boundary $\partial_\pi \mathbf{x}$ multiplied by Δf. The function Δf is defined as

$$\Delta f(\mathbf{x}, \pi) = \begin{cases} \lim_{\varepsilon \to 0} f(\mathbf{x} + \varepsilon\mathbf{n}, \pi) - f(\mathbf{x} - \varepsilon\mathbf{n}, \pi) & \text{if } \mathbf{x} \in \Delta\mathcal{X}(\pi), \\ f(\mathbf{x}, \pi) & \text{if } \mathbf{x} \in \partial\mathcal{X}(\pi) \end{cases} \tag{4.28}$$

The two-sided limit computes the difference between function values on either side of the discontinuity. The terms used in the theorem are illustrated in Figure 4.3.

The set $\Gamma(\pi)$ implicitly depends on the integrand f and the parameter being differentiated. For example, if $\mathcal{X}(\pi)$ is the support of a pixel, $\Gamma(\pi)$ would contain the silhouette edges of visible objects in that pixel. The notation here assumes that potentially $\mathcal{X}$ itself is parameter dependent. The theorem also applies for integrals over (subsets of) manifolds, e.g., the unit sphere [160]. However, this formulation does not work if $\mathcal{X}$ is a moving manifold, e.g., the surface of a shape that depends on π. It is possible to consider these cases as well, but this requires introducing additional derivative terms along normals and tangents of manifolds [152]. These terms must be considered if we want to rigorously differentiate a path space integral with respect to shape parameters. We will not go into this here, as we do not rely on that formulation for any of our algorithms.

4.5.2 Edge sampling

The interior term in Equation 4.27 can be evaluated using the previously described differentiable Monte Carlo estimators, but the boundary integral is more difficult to estimate. The derivatives caused by discontinuities need to be explicitly integrated over silhouette edges.

Li et al. [112] were the first to systematically investigate discontinuities in differentiable rendering. They proposed to use Monte Carlo integration to evaluate the boundary

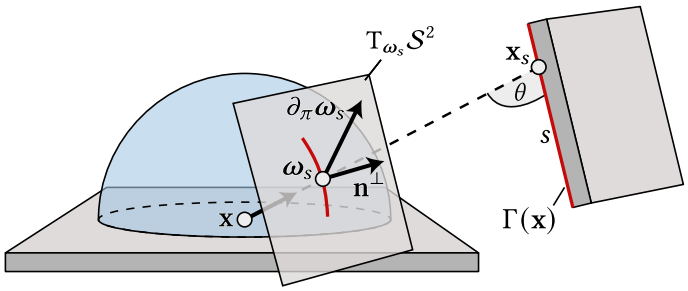


Figure 4.4: Visualization of some of the terms evaluated during edge sampling. The motion $\partial_\pi \boldsymbol{\omega}_s$ and the normal $\mathbf{n}^\perp$ are both in the local tangent space $T_{\omega_s}\mathcal{S}^2$.

term. For scenes consisting of triangle meshes, this can be done by explicitly sampling the set of edges causing discontinuities. At a shading point $\mathbf{x}$, the boundary integral over incident illumination becomes [160]:

$$\int_{\Gamma(\mathbf{x})} \Delta L_i(\mathbf{x}, \boldsymbol{\omega}_s) f_s(\mathbf{x}, \boldsymbol{\omega}_s, \boldsymbol{\omega}_o) \langle \mathbf{n}^\perp, \partial_\pi \boldsymbol{\omega}_s \rangle V(\mathbf{x}, \mathbf{x}_s) \frac{\sin\theta}{\|\mathbf{x} - \mathbf{x}_s\|} \, \mathrm{d}s, \tag{4.29}$$

where $\Gamma(\mathbf{x}) = \bigcup_{i=1}^{N} E_i$ is the set of silhouette edges E_i in 3D and s parameterizes the edge. The direction $\boldsymbol{\omega}_s$ is the normalized vector from $\mathbf{x}$ to $\mathbf{x}_s$: $\boldsymbol{\omega}_s := {}^{\mathbf{x}_s - \mathbf{x}}/_{\|\mathbf{x}_s - \mathbf{x}\|}$. The dot product $\langle \mathbf{n}^\perp, \partial_\pi \boldsymbol{\omega}_s \rangle$ computes the velocity of $\boldsymbol{\omega}_s$ against the normal of the discontinuity in the tangent space of the unit sphere. The division by the distance accounts for the projection of points on an edge to the unit sphere of directions. The angle θ is the angle between $\boldsymbol{\omega}_s$ and edge itself. Similar to the cosine term for surfaces, it attenuates contributions at grazing angles. The radiance difference $\Delta L_i(\mathbf{x}, \boldsymbol{\omega}_s)$ requires estimating the radiance on both sides of the edge (i.e., by recursively tracing two light paths). Some of the terms in this integral are shown in Figure 4.4.

Naïvely, one could uniformly sample all the triangle edges in the scene and then check if the sampled edge is a silhouette. However, this would scale extremely poorly to complex scenes. Therefore, Li et al. propose a hierarchical data structure that allows sampling discontinuities during path tracing. They first sample an edge using this hierarchical structure and then sample a point on the edge by using a linearly transformed cosine distribution to approximately sample the edge according to the BSDF [161, 162]. Zhang et al. [160] extend this method to handle scenes containing participating media.

Sampling silhouette edges in this way is difficult and the acceleration data structure cannot always guarantee efficient sampling. For example, it only poorly samples edges of complex meshes that are visible in reflections. A more robust edge sampling approach are path-space methods proposed by Zhang et al. [152, 163]. These first sample a point and direction on a silhouette edge and then connect to subpaths sampled from the sensor

and light sources. While complex to implement, this can result in high-quality edge gradients in scenarios with challenging lighting conditions. The performance can be improved further by employing a KD-tree to guide the initial sampling step [164].

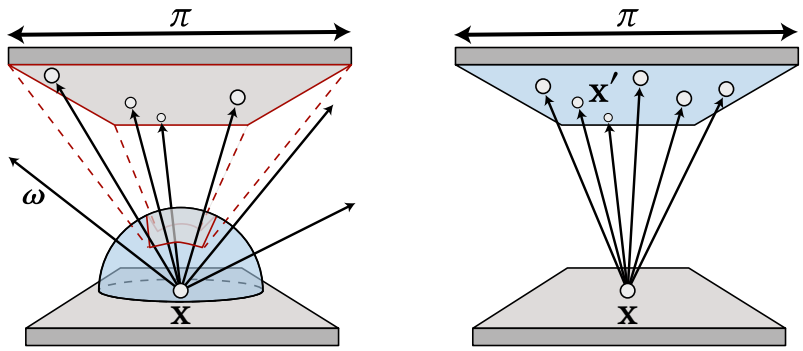
The edge sampling problem is closely related to next event estimation in the presence of a large number of light sources. If the number of lights is large, simple uniform sampling is insufficient for efficient next event estimation. It then becomes necessary to use importance sampling to decide which light source to evaluate. Similar to the discontinuities, the set of important light sources depends on the current shading point. Several works have investigated strategies to reduce variance for scenes with many light sources [165, 166, 167, 168]. Another closely related problem is drawing contours for non-photorealistic rendering [169, 170, 171]. To imitate a drawing using a rendering algorithm, we need to render outlines around objects, which also requires a way of detecting silhouette edges.

4.5.3 Reparameterization

Directly sampling discontinuity edges might not always work efficiently. It further seems difficult to generalize this idea to implicit surface representations, where the surface is given as the zero-level set of a function. In that case, the set of discontinuities to consider is not just a discrete list of triangle mesh edges.

As we have already seen in the discussion of detached and attached estimators, reparameterizing the integral is a powerful tool to construct new estimators. Roger et al. [25] provided an early example of such an idea applied to discontinuities, which we illustrate in Figure 4.5. Consider the problem of estimating the derivative of the area subtended by a rectangle on the hemisphere. There are two primal estimators we can construct for this problem: either we integrate over the hemisphere by sampling outgoing directions (Figure 4.5a) or we directly integrate over the area of the rectangle (Figure 4.5b). The first strategy results in a discontinuous integrand, as some rays might miss the rectangle. On the other hand, if we use area sampling and an attached derivative estimator, we circumvent the problem and can directly estimate the gradient by differentiating the Monte Carlo estimator. Zhang et al. [152] used this idea to compute derivatives of light sources and shapes with discontinuous surface normals.

Loubet et al. [154] proposed to design general reparameterizations that follow geometric discontinuities. The idea is that this reparameterized integrand should then be free of parameter-dependent discontinuities. Following this step, it is legal to move the derivative operator inside the integral and estimate parameter derivatives by sampling



(a) Solid angle integration (b) Area integration

Figure 4.5: The parameterization of the integrand affects the types of discontinuities we encounter. In this example, the parameter π controls the scale of the top rectangle. Our goal is to estimate the derivative of the area subtended by this rectangle in the hemisphere of directions at $\mathbf{x}$. We can do this by either integrating over the hemisphere itself **(a)** or over the area of the rectangle **(b)**. The integration domain is marked in light blue. When integrating over the hemisphere, the integrand is discontinuous on the border of the rectangle (red). By directly sampling the rectangle's surface, this issue vanishes and we can correctly estimate the derivative of the subtended area.

light paths using standard path tracing.

1D example. We can further illustrate this idea on a simple 1D example. Consider the following integral derivative:

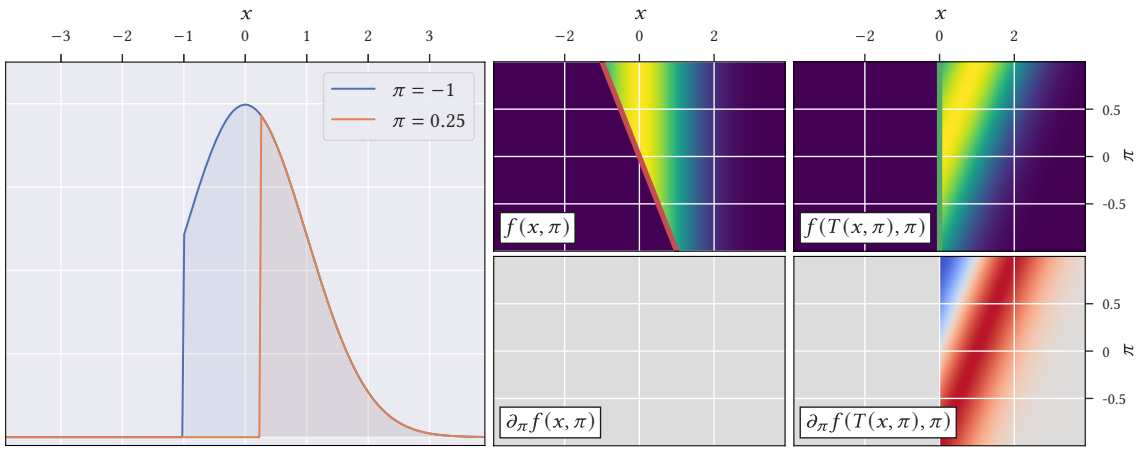
$$\partial_\pi \int_{\mathbb{R}} \mathbb{1}_{[\pi,\infty]}(x)\; g(x)\,\mathrm{d}x. \tag{4.30}$$

The integrand consists of a step function at position π multiplied by a smooth function g. Due to the discontinuity, we cannot simply move the derivative operator inside the integral. Following Loubet et al., we can introduce a simple reparameterization $\mathcal{T}(x,\pi) = x + \pi$, which turns the step function into a parameter-independent discontinuity and enables swapping gradient and integration operators:

$$\begin{aligned}\partial_\pi \int_{\mathbb{R}} \mathbb{1}_{[\pi,\infty]}(x)\; g(x)\,\mathrm{d}x = \partial_\pi \int_{\mathbb{R}} \mathbb{1}_{[\pi,\infty]}(x+\pi)\; g(x+\pi)\,\mathrm{d}x = \partial_\pi \int_{\mathbb{R}} \mathbb{1}_{[0,\infty]}(x)\; g(x+\pi)\,\mathrm{d}x \\ = \int_{\mathbb{R}} \mathbb{1}_{[0,\infty]}(x)\; \partial_\pi g(x+\pi)\,\mathrm{d}y.\end{aligned} \tag{4.31}$$

This example is illustrated in Figure 4.6. Intuitively, the reparameterization of the integrand modifies the integrand to factor in the effect of discontinuity, while decoupling the discontinuity itself from the parameter.

Divergence theorem. While the two previous examples provide intuitive understanding, Bangaru et al. [69] showed that the reparameterization idea can be motivated and



(a) Integrand $\mathbb{1}_{[\pi,\infty]}(x)\ g(x)$ (b) Original integrand (c) Reparam. integrand

Figure 4.6: In this example, we consider a 1D integrand **(a)** that is the product of a step function at position π and a smooth function g. **(b)** The derivative of the original integrand with respect to π is zero for all combinations of x and π. **(c)** By using a simple reparameterization, we can eliminate the dependence of the step function on π. The new derivative integrand is non-zero and can be integrated using Monte Carlo to estimate the derivative of the integral.

formally justified by considering the *divergence theorem*. The divergence theorem states that

$$\oint_{\partial A} \langle \mathbf{f}, \mathbf{n} \rangle \, \mathrm{d}\mathbf{s} = \int_{A-\partial A} \nabla_{\boldsymbol{\omega}} \cdot \mathbf{f} \, \mathrm{d}\boldsymbol{\omega}, \tag{4.32}$$

where $\mathbf{f}$ is a vector field, $\mathbf{n}$ is a normal vector and $\nabla_{\boldsymbol{\omega}}$ is the divergence operator. This equality relates an integral over the domain boundary ∂A to an integral over the interior $A - \partial A$. Bangaru et al. applied the divergence theorem to convert an integral over discontinuity boundaries into an area integral:

$$\begin{aligned} \oint_{\Gamma(\pi)} f(\mathbf{x}, \pi) \langle \partial_\pi \mathbf{x}, \mathbf{n} \rangle \, \mathrm{d}\mathbf{s} &= \int_{\mathcal{X}} \nabla_{\mathbf{x}} \cdot (f(\mathbf{x}, \pi) \mathcal{V}(\mathbf{x}, \pi)) \, \mathrm{d}\mathbf{x} \\ &= \int_{\mathcal{X}} \partial_{\mathbf{x}} f(\mathbf{x}, \pi) \cdot \mathcal{V}(\mathbf{x}, \pi) \, \mathrm{d}\mathbf{x} + \int_{\mathcal{X}} f(\mathbf{x}, \pi) \, \nabla_{\mathbf{x}} \cdot \mathcal{V}(\mathbf{x}, \pi) \, \mathrm{d}\mathbf{x}. \end{aligned} \tag{4.33}$$

Here, $\mathcal{V}$ is a so-called *warp field*, which can be thought of as an interpolated vector field of the velocity of the discontinuities. The warp field needs to satisfy the following criteria:

1. **Continuity:** The warp field $\mathcal{V}$ itself has to be continuous.

2. **Boundary consistency:** For $\mathbf{x}_b$ on the boundary, we need $\mathcal{V}(\mathbf{x}_b, \pi) = \partial_\pi \mathbf{x}_b$

As long as the warp field continuously interpolates the boundary velocities, the area form of the boundary term is equivalent to the original boundary integral. The area integral can then be estimated using regular Monte Carlo integration, without explicitly sampling edges.

Relation to reparameterization. Following Bangaru et al. [69], we can connect this back to the idea of reparameterization. By change of variables, a reparameterization $\mathcal{T}$ can be used to rewrite an integral as:

$$\int_{\mathcal{T}(\mathcal{X})} f(\mathbf{x}, \pi)\, \mathrm{d}\mathbf{x} = \int_{\mathcal{X}} f(\mathcal{T}(\mathbf{x}, \pi), \pi) |\mathbf{J}_{\mathcal{T}}(\mathbf{x}, \pi)|\, \mathrm{d}\mathbf{x}. \tag{4.34}$$

For now, this assumes the integration $\mathcal{X}$ to be flat, e.g., this derivation does not apply for integration over the unit sphere of directions. We will discuss this important use case shortly.

For the differentiable rendering problem, we can construct $\mathcal{T}$ such that at the current parameter π it is simply the identity: $\mathcal{T}(\mathbf{x}, \pi) = \mathbf{x}$. However, we do this such that it depends on π and $\partial_\pi \mathcal{T} \neq 0$. We can therefore also assume that the integration domain remains unchanged: $\mathcal{T}(\mathcal{X}) = \mathcal{X}$. The idea is that the reparameterization should only affect derivative computation, but not the primal integrals [154]. The derivative of the reparameterized integral is then:

$$\begin{aligned} \partial_\pi \int_{\mathcal{X}} f(\mathcal{T}(\mathbf{x}, \pi), \pi) |\mathbf{J}_{\mathcal{T}}(\mathbf{x}, \pi)|\, \mathrm{d}\mathbf{x} = &\int_{\mathcal{X}} \partial_\pi f(\mathbf{x}, \pi)\, \mathrm{d}\mathbf{x} \\ &+ \int_{\mathcal{X}} \partial_{\mathbf{x}} f(\mathbf{x}, \pi) \cdot \partial_\pi T(\mathbf{x}, \pi)\, \mathrm{d}\mathbf{x} \\ &+ \int_{\mathcal{X}} f(\mathbf{x}, \pi) \partial_\pi |\mathbf{J}_{\mathcal{T}}(\mathbf{x}, \pi)|\, \mathrm{d}\mathbf{x}, \end{aligned} \tag{4.35}$$

where we used that $\mathcal{T}$ is a π-dependent identity map with unit Jacobian determinant. We can contrast this with the result of the interior term and the boundary term after applying the divergence theorem:

$$\begin{aligned} \partial_\pi \int_{\mathcal{X}} f(\mathbf{x}, \pi)\, \mathrm{d}\mathbf{x} = &\int_{\mathcal{X}} \partial_\pi f(\mathbf{x}, \pi)\, \mathrm{d}\mathbf{x} \\ &+ \int_{\mathcal{X}} \partial_{\mathbf{x}} f(\mathbf{x}, \pi) \cdot \mathcal{V}(\mathbf{x}, \pi)\, \mathrm{d}\mathbf{x} \\ &+ \int_{\mathcal{X}} f(\mathbf{x}, \pi) \nabla_{\mathbf{x}} \cdot \mathcal{V}(\mathbf{x}, \pi)\, \mathrm{d}\mathbf{x}. \end{aligned} \tag{4.36}$$

This shows that both approaches are in fact equivalent, as long as $\mathcal{V}(\mathbf{x}, \pi) = \partial_\pi \mathcal{T}(\mathbf{x}, \pi)$ and $\nabla_{\mathbf{x}} \cdot \mathcal{V}(\mathbf{x}, \pi) = \partial_\pi |\mathbf{J}_{\mathcal{T}}(\mathbf{x}, \pi)|$. The first equality implies that the reparameterization

$\mathcal{T}$ should be constructed such that $\partial_\pi \mathcal{T}(\mathbf{x}_b, \pi) = \partial_\pi \mathbf{x}_b$ for points $\mathbf{x}_b$ on the boundary. In other words, differentiating the reparameterization should result in a differential motion that perfectly matches the motion of the discontinuity. The second equality can be shown by using Jacobi's formula [69]: $\partial_\pi \left|\mathbf{J}_\mathcal{T}(\mathbf{x}, \pi)\right| = \operatorname{tr}\left(\operatorname{adj}(\mathbf{J}_\mathcal{T}(\mathbf{x}, \pi))\partial_\pi \mathbf{J}_\mathcal{T}(\mathbf{x}, \pi)\right)$, where tr(...) is the trace of a matrix and adj(...) is the *adjugate* matrix. Since $\mathbf{J}_\mathcal{T}(\mathbf{x}, \pi)$ is the identity matrix, its adjugate is the identity as well. Therefore, we get the desired equality between the derivative of the Jacobian determinant and the divergence of the warp field:

$$\partial_\pi \left|\mathbf{J}_\mathcal{T}(\mathbf{x}, \pi)\right| = \operatorname{tr}\left(\partial_\pi \mathbf{J}_\mathcal{T}(\mathbf{x}, \pi)\right) = \nabla_\mathbf{x} \cdot \partial_\pi \mathcal{T}(\mathbf{x}, \pi) = \nabla_\mathbf{x} \cdot \mathcal{V}(\mathbf{x}, \pi). \tag{4.37}$$

In summary, this shows that we can approach the problem either as a reparameterization or using the divergence theorem. We will in the following adopt the reparameterization perspective. The main reason for this is that it naturally leads to mathematical expressions compatible with reverse-mode AD. The reparameterization and its Jacobian can depend on an arbitrary number of differentiable parameters.

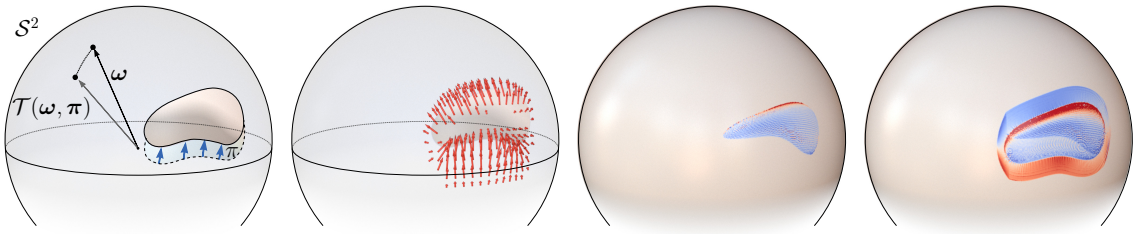
Spherical integrals. In the following, we will establish the necessary theory to apply reparameterizations to spherical integrals. This perspective on reparameterizations is novel and was published as part of the article we discuss in Chapter 6. We provide these theoretical insights here, as they are general and not specific to the representation used in that later chapter.

In rendering, we commonly integrate quantities over the set of directions, or equivalently, over the surface of the unit sphere. Unidirectional rendering algorithms can be expressed as the recursive solution of spherical integrals. Therefore, our goal is to differentiate integrals of the form:

$$\partial_\pi I(\pi) = \partial_\pi \int_{\mathcal{S}^2} f(\boldsymbol{\omega}, \pi)\, \mathrm{d}\boldsymbol{\omega}. \tag{4.38}$$

This formulation, for now, does not handle recursive integration, i.e., to account for interreflections, but that will be sufficient for most derivations. We will not handle the very challenging special case of discontinuities observed through perfectly specular interactions (e.g., geometry observed through a water surface).

Given this spherical integral, we can reformulate the problem using a change of variables $\mathcal{T} : \mathcal{S}^2 \to \mathcal{S}^2$ that maps the unit sphere onto itself. The reparameterization $\mathcal{T}$ has to be chosen so that the resulting integrand is free of discontinuities with respect to the



(a) Integrand (b) Vector field $\partial_\pi \mathcal{T}$ (c) Original gradient (d) Reparam. gradient

Figure 4.7: Illustration of a discontinuous integrand on the unit sphere $\mathcal{S}^2$. **(a)** We integrate the color of a shaded shape over a constant-colored background. The scene parameter π controls the translation of the object. **(b)** We then introduce a reparameterization $\mathcal{T}$, which is designed so that the vector field $\partial_\pi \mathcal{T}$ follows the motion of the object boundary and falls off continuously to 0 away from the object. **(c)** We further visualize the gradient of the original integrand and **(d)** the gradient after reparameterizing, including the area change. Blue and red colors indicate negative and positive values, respectively.

scene parameters, which then allows moving the derivative operator into the integral:

$$\begin{aligned}
\partial_\pi I(\pi) &= \partial_\pi \int_{\mathcal{S}^2} f(\boldsymbol{\omega}, \pi)\, \mathrm{d}\boldsymbol{\omega} \\
&= \partial_\pi \int_{\mathcal{S}^2} f\left(\mathcal{T}(\boldsymbol{\omega}, \pi), \pi\right) \left\| \mathrm{D}\mathcal{T}_{\boldsymbol{\omega},\pi}(\mathbf{s}) \times \mathrm{D}\mathcal{T}_{\boldsymbol{\omega},\pi}(\mathbf{t}) \right\| \mathrm{d}\boldsymbol{\omega} \\
&= \int_{\mathcal{S}^2} \partial_\pi \left[f\left(\mathcal{T}(\boldsymbol{\omega}, \pi), \pi\right) \left\| \mathrm{D}\mathcal{T}_{\boldsymbol{\omega},\pi}(\mathbf{s}) \times \mathrm{D}\mathcal{T}_{\boldsymbol{\omega},\pi}(\mathbf{t}) \right\| \right] \mathrm{d}\boldsymbol{\omega},
\end{aligned} \tag{4.39}$$

where $\mathbf{s}$ and $\mathbf{t}$ are orthonormal tangent vectors of the unit sphere at $\boldsymbol{\omega}$ and $\mathrm{D}\mathcal{T}_{\boldsymbol{\omega},\pi}$ is the differential of $\mathcal{T}$ with respect to the vector $\boldsymbol{\omega}$. The norm of the cross product of transformed tangent vectors accounts for the distortion in the integration domain, similar to the Jacobian determinant for a change of variables in ambient space. If we were to reparameterize the 3D ambient space, we would simply use the Jacobian determinant of the mapping. However, here this would be incorrect as the reparameterization works on a manifold. We instead need to use the norm of the cross product of the tangent vectors, mapped through the differential of the reparameterization.

This formulation of the reparameterized integral using the cross product of the transformed tangent vectors is more explicit than what has been described in previous work. We use this framework in Chapter 6 to define reparameterizations for implicit surfaces. In Appendix A, we draw the connection between this formulation as a reparameterization over the unit sphere and the divergence formulation by Bangaru et al. [69]. We show that we can evaluate either the cross-product term or the divergence of the mapping and both will give the same results when differentiated, and correctly account for the fact that we reparameterize an integral over a manifold.

For correctness, we need the reparameterization to satisfy $\partial_\pi \mathcal{T}(\boldsymbol{\omega}_b, \pi) = \partial_\pi \boldsymbol{\omega}_b$ for all directions $\boldsymbol{\omega}_b$ that lie on the set of discontinuities of the original integrand. Note that now $\partial_\pi \mathcal{T}(\boldsymbol{\omega}_b, \pi)$ and $\partial_\pi \boldsymbol{\omega}_b$ are vectors that lie in the tangent space of the unit sphere. Aside from that, we need $\partial_\pi \mathcal{T}(\boldsymbol{\omega}, \pi)$ itself to be continuous for the reparameterization to be valid.

Figure 4.7 illustrates a reparameterization of an integral over the unit sphere. We visualize the original integrand, the vector field $\partial_\pi \mathcal{T}(\boldsymbol{\omega}, \pi)$, and the derivatives of the original and reparameterized integrand. The gradient of the original integrand does not contain any terms related to the occlusion change on the silhouette of the moving object.

Constructing a reparameterization. The key challenge in using the reparameterization approach is constructing a suitable map $\mathcal{T}$, or equivalently, a warp field $\mathcal{V}$. For triangle meshes, a suitable reparameterization can be obtained by first constructing an auxiliary reparameterization, which attains the correct motion at triangle boundaries. This reparameterization is then convolved with a filter kernel in the spherical domain, that removes discontinuities from the reparameterization itself [69]. Since this blurring only affects the reparameterization, it can be used to construct an unbiased gradient estimator of the original integral. Concretely, Bangaru et al. [69] suggest the following warp field:

$$\mathcal{V}(\boldsymbol{\omega}, \pi) = \frac{\int_{S^2} w(\boldsymbol{\omega}, \boldsymbol{\omega}') \mathcal{V}^{\text{direct}}(\boldsymbol{\omega}', \pi) \, \mathrm{d}\boldsymbol{\omega}'}{\int_{S^2} w(\boldsymbol{\omega}, \boldsymbol{\omega}') \, \mathrm{d}\boldsymbol{\omega}'}, \tag{4.40}$$

where $w(\boldsymbol{\omega}, \boldsymbol{\omega}')$ is a weighting kernel and $\mathcal{V}^{\text{direct}}(\boldsymbol{\omega}', \pi)$ is a naïve, discontinuous warp field that simply follows the triangle motion. It is computed by evaluating the surface motion at the ray intersection location. The weighting kernel is defined as

$$w(\boldsymbol{\omega}, \boldsymbol{\omega}') = \frac{1}{\mathcal{D}(\boldsymbol{\omega}, \boldsymbol{\omega}') + \mathcal{B}(\boldsymbol{\omega}')}. \tag{4.41}$$

Here, $\mathcal{D}(\boldsymbol{\omega}, \boldsymbol{\omega}') = \exp(\kappa(1 - \langle \boldsymbol{\omega}, \boldsymbol{\omega}' \rangle)) - 1$ measures the distance between two directions and the scalar concentration parameter κ controls the width of the kernel. The *boundary test* $\mathcal{B}(\boldsymbol{\omega}')$ measures the distance to the triangle mesh boundary. As $\boldsymbol{\omega}'$ approaches a boundary, $\mathcal{B}$ goes to zero. The combination of these two distance measures ensures that the warp field attains the correct boundary motion. The convolution is evaluated using Monte Carlo integration and requires tracing a number of auxiliary rays to sample the space around $\boldsymbol{\omega}$. Moreover, unbiased estimation of $1/\int_{S^2} w(\boldsymbol{\omega}, \boldsymbol{\omega}') \, \mathrm{d}\boldsymbol{\omega}'$ requires the use of a de-biasing technique, which introduces additional variance. This method has been

designed for triangle meshes, but can also be generalized to other representations. In Chapter 6, we use this as a baseline when proposing a new reparameterization method.

4.5.4 Alternative methods

Edge sampling and reparameterization are the main methods that address discontinuities that occur in physically-based differentiable rendering. The problem of differentiating discontinuities also occurs in a few related contexts, which we will briefly mention here.

Domain-specific languages. There has been some work on incorporating support for discontinuities directly into AD frameworks or a domain-specific language (DSL). An automated, general-purpose solution to differentiate discontinuous programs would be useful for a wide range of tasks. Discontinuities for example also occur in a rigid body simulation, where objects might collide and instantaneously change directions. TEG [172] is a DSL that uses local invertibility to localize discontinuities symbolically. However, the current system is too constrained to handle the complexity of differentiable Monte Carlo rendering. Yang et al. [173] build support for discontinuities into shader programs. They demonstrate that this enables the optimization of parameters used in procedural shaders. Their method does not generalize to arbitrary programs and is therefore not directly applicable to physically-based differentiable rendering.

While it is interesting to consider a compilation approach, it is unlikely that the problem of discontinuities can efficiently be solved entirely by the compiler. If anything, a compiler might be able to reduce the implementation work, similar to the Aether DSL [174], which aids the implementation of rendering algorithms by automatically computing Jacobians and PDFs.

Rasterization. Several works [175, 176, 177, 178, 179] have proposed differentiable versions of triangle mesh *rasterization*. A rasterizer renders triangle meshes to the screen by iterating over all triangles and then filling in the pixels occupied by the visible parts of each of them. The color of each pixel will be determined by the triangle overlapping its center. This discrete assignment of triangles to pixels is inherently not differentiable. Differentiable rasterization methods either introduce a form of soft blending of triangles [177], approximate analytic visibility [178] or differentiable splatting of surface samples to the image buffer [179]. Zhou et al. [180] used a data structure to compute differentiable analytic visibility for primary rays. Rasterization is computationally efficient and commonly used in real-time rendering applications (e.g., games), but difficult

to use to (differentiably) render soft shadows, reflections or indirect illumination. For many inverse rendering applications, differentiating these physical effects is important and therefore rasterization is less useful.

4.6 Applications of differentiable rendering

In the following, we provide a high-level overview and survey of recent work on applications of physically-based differentiable rendering.

4.6.1 Surface reconstruction

Differentiable rendering facilitates the joint reconstruction of surface geometry, material properties and lighting conditions. We focus on works that consider some form of physically-plausible light transport. Nayar et al. [181] showed 30 years ago that object reconstruction can benefit from accounting for multiple scattering. Their method is restricted to mostly flat, textured Lambertian surfaces and does not consider occlusion change. Similarly, Chandraker et al. [182] demonstrated that considering interreflections resolves depth ambiguities. Hauagge et al. [183] improved surface reconstruction by estimating the average local occlusion of incident ambient illumination. All these works consider the reconstruction of objects using a single camera view, but multiple lighting conditions. Recently, several papers suggest using differentiable path tracing to estimate BSDF and lighting parameters of indoor scenes [149, 184]. The scene parameters obtained using these methods enable applications such as relighting or realistic object insertion for augmented reality. It is also possible to jointly reconstruct object geometry and BSDF parameters [185, 186, 187]. In these applications, there is generally a correlation between reconstruction accuracy and how controlled the capture setup is. For example, we most likely cannot hope to obtain fully relightable assets only using a single static illumination condition.

Objects made out of glass or refractive materials are another example that requires considering indirect illumination in the reconstruction process. Differentiable rendering has recently been used to reconstruct glass objects [188, 189]. The complexity of this problem requires either controlled illumination [188] or pre-training of a neural network to guide the reconstruction process [189]. Bemana et al. [190] reconstruct transparent objects by optimizing a volumetric representation with a spatially-varying index of refraction. They exploit the reversibility of the specular light paths to efficiently compute gradients. Kassubeck et al. [191] reconstruct transparent objects from a single image of

the refractive caustic they produce.

Edge sampling methods have also been applied to differentiate *time-resolved* rendering [192, 193]. Time-of-flight (ToF) sensors not only capture photon intensities, but also information about photon arrival times. By simulating and differentiating the ToF capture process, additional information about a scene can be retrieved. This can for example be useful for non-line of sight imaging [194], where an unknown scene is reconstructed by observing its diffuse reflectance (e.g., on a white wall). This problem is severely ill-posed and time-of-flight sensors are necessary to reduce ambiguities.

4.6.2 Implicit surfaces

Surface reconstruction and the problem of discontinuities are closely related to the underlying choice of geometry representation. Triangle meshes are extremely popular for forward rendering, but their fixed topology can be a burden for inverse problems. Inverse problems have therefore considered a broader range of possible geometry representations. For general 3D geometry, an *implicit* representation models the surface as a zero-level set: $\mathcal{S} = \{f(\mathbf{x}) = 0 \colon \mathbf{x} \in \mathbb{R}^3\}$. Using an implicit surface for 3D reconstruction is a form of a *level set method* [195]. A common implicit representation are signed distance functions (SDFs). A signed distance function ϕ measures the distance to the 3D surface. The distance is positive for points outside an object and negative for points inside. Many works have used SDFs or general level set methods for 3D object reconstruction using laser scanners [196, 197] or RGB-D sensors [198, 199]. Level set methods have also been extended to account for the motion of occlusion boundaries [200, 201].

Recently, there has been some work on using SDFs as a geometry representation for differentiable rendering. Concurrently, Jiang et al. [202] and Liu et al. [203] proposed to use SDFs for scene reconstruction using differentiable rendering. These works use *sphere tracing* [204, 205, 206] to intersect rays with an SDF. To deal with visibility discontinuities, Liu et al. [203] introduced a differentiable version of a silhouette loss, similar to later work by Yariv et al. [207]. Niemeyer et al. [208] optimize implicit surfaces by adding a 3D loss based on the visual hull [209]. Mehta et al. [210] present a level-set approach to update the parameters of SDFs represented by neural networks.

Relying on a silhouette loss can be problematic. Such approaches are difficult to generalize to shadows and higher-order light interactions. Additionally, they cannot work in cases where silhouette information is not applicable (e.g., reconstructing a room from the inside). Cole et al.'s [179] differentiable splatting method does not require silhouette information but cannot generalize to shadows and interreflections. An alternative

approach is to convert the SDF to a triangle mesh using marching cubes [211] or marching tetrahedra [212], to then fall back to using differentiable triangle mesh rendering methods [213, 214]. These methods can work well [186, 187], but do not directly differentiate the process of SDF rendering. In Chapter 6, we present an algorithm that avoids meshing and leverages the SDF's global structure to obtain high-quality gradients using a reparameterization. A concurrent article proposes a similar approach for neural SDFs [215].

4.6.3 Volume rendering

Inverse multiple scattering. Recovering the parameters of a participating medium is an important use case for physically-based differentiable rendering and requires accounting for potentially long light paths with many bounces. Unlike surface reconstruction, we cannot hope to get physically-plausible parameters when ignoring multiple scattering. In computer graphics, differentiable volume rendering was pioneered by Gkioulekas et al. [18], who recovered the parameters of homogeneous media from multiple scattering. Khungurn et al. [151] optimized the parameters of fabric appearance models. Later work used image-based acquisition to reconstruct volumes (e.g., smoke) under multiple scattering [148]. Zhao et al. [216] applied differentiable volume rendering to the problem of downsampling volume rendering parameters. Velinov et al. [217] fit the scattering parameters of human teeth using gradient descent. Differentiable rendering has also been combined with the diffusion approximation [11] to recover the subsurface scattering parameters of real objects [218]. Che et al. [219] use a differentiable renderer to train a neural network that infers scattering parameters from images. They formulate an autoencoder architecture, where images are encoded into medium parameters using a neural network. The parameters are then rendered using a differentiable volume renderer, which allows training the entire encoder-decoder architecture end-to-end. The complex, non-linear relation between volume parameters and appearance has also led to inverse rendering methods being developed for user-guided appearance editing [220, 221]. In our work on the Mitsuba 2 renderer, we showed that one can use automatic differentiation to optimize the parameters of heterogeneous volumes while accounting for multiple scattering [48].

Scene representation and neural rendering. While surface-based representations enable efficient rendering and are the standard for forward rendering, image-based optimization of surface geometry can be challenging due to the inherent non-convexity

of such an optimization. As an alternative, volumetric representations have recently seen widespread use as a differentiable, general-purpose scene representation. Volumetric approaches are particularly useful for scenes containing aggregate geometry such as hair or foliage. These are generally poorly approximated by triangle mesh geometry and a volumetric representation is easier to optimize. They are often used in combination with neural rendering, where (part of) the rendering process is approximated by a neural network instead of physically-based rendering. For example, Lombardi et al. [222] used a neural volumetric representation to model facial performances. In their case, the volumetric model is used to represent hair, which is difficult to reconstruct as a triangle mesh.

The most popular volumetric representation are *neural radiance fields (NeRFs)* [223]. NeRFs are emissive volumes represented by a neural network. Given a 3D point and direction, a small fully-connected neural network predicts emitted radiance and a volume density value. This representation is rendered by accumulating emitted radiance along camera rays using ray marching. The neural network is optimized to reproduce the appearance of a set of reference views. Several works have suggested replacing the neural network by grid-based spherical harmonics [224, 225, 226], which can achieve similar performance. Related volumetric representations have also been extended to allow re-rendering under novel illumination conditions [227, 228]. The work we discuss in Chapter 7 proposes a new relightable volumetric representation for general scenes.

Coordinate-based neural networks like NeRFs are also called *neural fields* and can be used as a representation for any coordinate-based data (e.g., 2D images). Since they do not suffer from the curse of dimensionality, they allow pushing beyond the resolution limits of discretized grids and generalize to higher-dimensional signals like the directional emission used by NeRF. Recent works have also suggested combining neural fields with adaptive data structures [229, 230] or hash grids [231] to further improve evaluation performance. We refer to the state-of-the-art report by Xie et al. [232] for an overview.

4.6.4 Alternative differentiable representations

Aside from purely surface-based or volumetric representations, there has also been some work on alternative representations. Recent hybrid methods [233, 234] define a volume density based on an underlying SDF to improve the convexity of the geometry reconstruction problem. Unlike the method we discuss in Chapter 6, these approaches rely on differentiable volume rendering and do not directly differentiate the surface rendering

process.

Alternatively, point-based shape representations have also been shown to produce high-quality scene reconstructions [235, 236]. PointNeRF [237] uses a set of points to control an emissive volumetric representation. The Pulsar renderer [238] represents objects as collections of colored spheres.

The design space for scene representations is large: we can combine points, explicit surfaces, implicit surfaces and volumes. The representation can further be compressed by using adaptive data structures or coordinate-based neural networks. Depending on the representation, the inverse rendering problem can become significantly easier. A disadvantage of many hybrid or neural representations is that they are difficult to reconcile with physically-based light transport, e.g., to account for interreflection. In this thesis, we focus on representations that admit a physically-based light transport model.

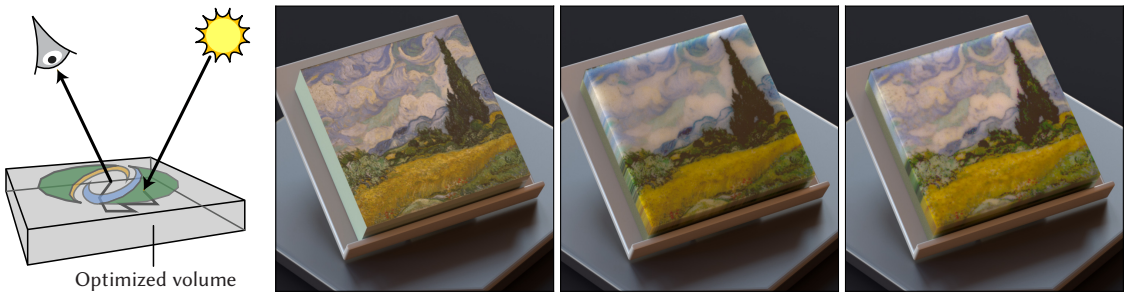
4.6.5 Optics and digital fabrication

Differentiable rendering can be used to optimize the design of optical system or fabricated objects.

Lens design. An interesting application of differentiable rendering is optimizing lenses and camera systems to satisfy certain quality criteria. Li et al. [239] optimize a single lens element using differentiable ray tracing. Sun et al. [23] optimize an entire lens system by tracing light paths through a lens consisting of multiple elements. Tseng et al. [240] optimize compound optical systems by training a neural network to approximate a commercial black-box optics simulator. The neural network approximation is then differentiated to optimize optical parameters.

Caustic design. A closely related problem is *caustic design*. The idea is to design a reflective or refractive surface that focuses the light into a caustic that approximates a given target picture [241, 242, 243]. This problem can be addressed using optimal transport [243] or by solving a series of Poisson problems [242]. We showed in [48] that good results can also be achieved by simply differentiating a suitable Monte Carlo rendering algorithm.

This idea can further be extended to *gradient index optics*, which are lenses with a spatially-varying index of refraction (IOR). By differentiating a forward simulation [244] of the photon trajectories through the material, the IOR values can be optimized for a certain target caustic. Accurately manufacturing such lenses is still an open problem [245].



(a) Scene setup (b) Target appearance (c) Naïve (d) Ours (optimized)

Figure 4.8: Colored 3D printer inks can be highly translucent, leading to internal scattering **(a)**. If we naïvely attempt to replicate the appearance of a textured solid object **(b)**, we end up with undesirable blurring due to scattering **(c)**. We optimize the volume's albedo coefficients to better reproduce the reference appearance **(d)**. This can recover some of the contrast that is lost due to scattering. All images are rendered, as we did not attempt to fabricate these objects.

Recently, Teh et al. [246] exploited reversibility of iteration steps to optimize gradient-index optics without storing a computation graph. They show that their approach can for example be used to optimize the design of optical fibers.

3D printing. Differentiable volume rendering has been employed to fabricate 3D objects with a certain target appearance. Papas et al. [22] optimized mixing proportions of colored pigments to produce silicone objects replicating the scattering properties of a reference material. Another application is 3D color printing, where mixtures of colored plastics are deposited to fabricate colored objects. The used inks are highly translucent, which leads to volumetric scattering inside the printed object. The scattering reduces the contrast of printed surface textures. Several specialized heuristics for this problem exist [247, 248], but we showed in [48] that the surface appearance can be optimized directly using differentiable volumetric path tracing. In Figure 4.8 we show an example result of this optimization. The differentiable rendering optimization optimizes volume albedo values stored on a voxel grid. The setting here is slightly simplified, in that we do not consider density variation between voxels. We consider up to 64 scattering events inside the medium. Nindel et al. [249] build an optimization pipeline that considers the full complexity of the problem, including measured scattering properties of the actual 3D printer inks. A differentiable rendering approach has also been proposed to optimize the transmittance of a 3D printed light field display [250].

4.7 Differential transport in other fields

Similar differentiable Monte Carlo estimators have been used in a range of adjacent fields. Depending on the context, these methods are also referred to as *perturbation Monte Carlo* or *sensitivity analysis*. We have already discussed a few of these in Section 1.3.

In nuclear engineering, differentiable estimators of neutron scattering are useful to analyze the criticality of nuclear reactors [31, 32] and to optimize radiation shielding [29, 30]. Differentiable Monte Carlo simulations have found use in improving the design of combustion chambers [25] and the reconstruction of of transparent gas flows encountered during fuel injection [246].

The reconstruction of the scattering properties of participating media has a range of interesting medical applications. Differentiable transport has been used to recover the scattering parameters of organic tissue [251, 252]. An application of this is cancer diagnosis, where it has been observed that the phase function parameters differ between healthy liver tissue and liver tumors [253]. It is also possible to simulate scattering in a *virtual waveguide* produced by applying ultrasound to tissue. This can be used to guide radiation for cancer therapy [254].

In physics, differentiable delta tracking [96, 255] has been investigated to compute derivatives of atmospheric scattering. Previous work however is restricted to forward-mode differentiation, which only allows considering a small number of parameters. In Chapter 5, we show how to efficiently differentiate delta tracking with respect to an arbitrary number of variables. Differentiable Monte Carlo has also been used to recover parameters of the aggregate scattering due to plant canopy [36]

Lastly, Monte Carlo integration and its derivatives also find use in computing pricing of financial options [256]. The derivatives estimate the sensitivity of option prices with respect to parameters such as interest rates.

4.8 Derivatives in forward rendering

Aside from inverse rendering, derivative computations have also found use in forward rendering applications. Various prior works in computer graphics have differentiated light paths in forward mode, propagating an infinitesimal perturbation of a scene or ray parameter through the calculation to determine how it influences the rendered image or outgoing ray. These derivatives then find use in texture filtering of objects observed

through specular reflection [257], when interpolating precomputed diffuse [258] or specular illumination [259], and for adaptive sampling and reconstruction [260]. They can also be used to determine valid specular path configurations to render effects such as caustics or glints [99, 261, 262]. Derivatives have also been used in image space, for example to render gradient images that are subsequently reconstructed [65, 66, 263]. Li et al. [264] used forward-mode AD to compute first and second-order gradients to improve exploration of the path space for Markov chain Monte Carlo methods. Later work combined first-order gradients with insights from gradient-based optimization to further improve on this idea [265].

5 | Path Replay Backpropagation

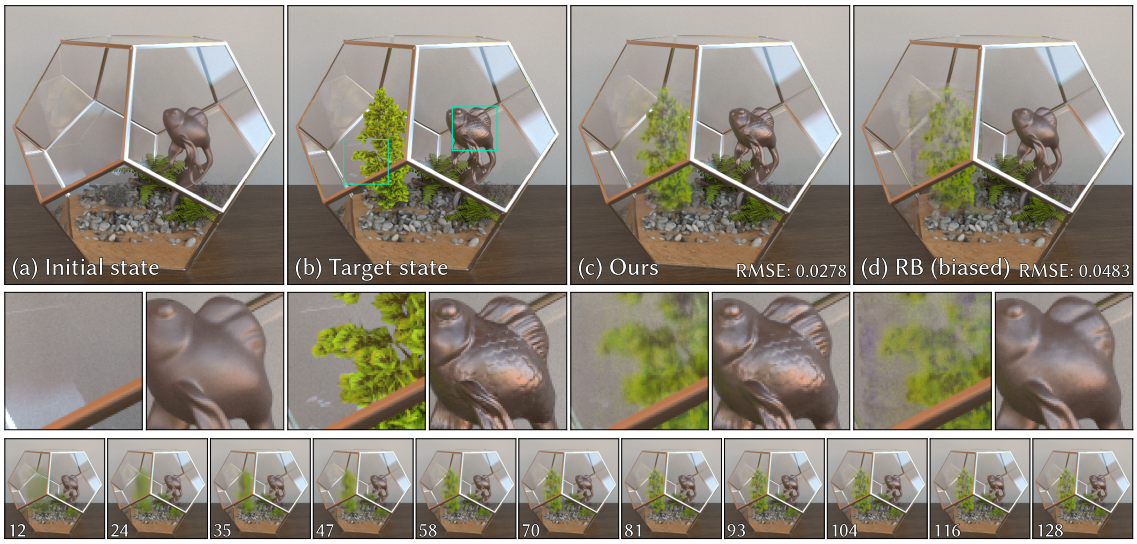


Figure 5.1: Inverse reconstruction of a scene with complex lighting and heterogeneous structure. Given the initialization **(a)**, we seek to reconstruct the target **(b)** involving normal-mapped surface variation and roughness changes on the fish sculpture, and the addition of a plant based on triangular geometry. Using three rendered views of the target, we apply our proposed *path replay backpropagation* (PRB) **(c)** and a linear-time version of *radiative backpropagation* (RB) [266] **(d)** to reconstruct the modified sculpture and a heterogeneous medium approximating the plant. Our method computes unbiased gradients and is able to converge to a higher-quality solution at equal time. The second and third rows show insets and PRB's convergence over time.

With the necessary background established, we now move on to discuss the first publication. In this chapter, we introduce *path replay backpropagation*, an algorithm that is designed to efficiently differentiate physically-based rendering algorithms with a high number of scattering events. In principle, we can differentiate physically-based light transport simulations using automatic differentiation. Mathematically, the estimators introduced in Section 4.4 are compatible with using a standard AD implementation. However, simulations featuring complex light transport phenomena pose severe challenges: effects like global interreflection, subsurface scattering, and heterogeneous participating media produce immense amounts of unstructured arithmetic, whose effect on derivatives must be carefully tracked. The computation of derivatives must proceed in reverse mode to efficiently convert a small number of derivatives at the rendering algorithm's output end into a large number of derivatives at the input end. The output end typically involves a scalar objective computed from the rendered image, while the input refers to the high-dimensional scene parameter space.

The inevitable predicament faced by reverse-mode differentiation is that the differential version of an algorithm requires access to certain quantities of the original computation, and these accesses occur in an order that is reversed when compared to the primal calculation. As discussed in Chapter 4, the standard remedy to this problem entails discarding most primal variables except for a sparse set of checkpoints [120] placed at strategic locations like function calls or the beginning of each loop iteration. A checkpoint captures the full program state at an instant of time, enabling the recovery of discarded information via re-computation in a classic computation versus memory trade-off. However, even individual checkpoints tend to be large in the context of rendering, and accessing them incurs significant storage costs and memory access latencies. Allocating gigabytes of memory for checkpoints limits the use of differentiable rendering to simple scenes, where the memory needed for the scene parameters is relatively small. This is undesirable, as in most practical use cases we want to allocate as much memory as possible to a high-resolution scene representation (e.g., a large voxel grid storing volume parameters). The memory use of large scenes is already a constraint for forward rendering [267], and differentiation must not exacerbate this problem.

Nimier-David et al. [266] observed that the gradient of the rendering process can be formulated as the solution of a *differential* rendering equation. Instead of relying on automatic differentiation, their *radiative backpropagation* (RB) method computes gradients by tracing light paths. This algorithm accumulates gradients directly without memorizing intermediate state, which removes the memory footprint issue and also improves efficiency. At the time of its publication, the RB method achieved impressive speedups over Mitsuba 2 [48], which relied entirely on automatic differentiation.

However, radiative backpropagation has two main limitations: the algorithm is either severely biased or requires a recursive radiance estimate at each scattering interaction in addition to its own recursion, making its computation time *quadratic* in the number of scattering events along a light path. The method also cannot differentiate interactions involving perfectly specular BSDFs, such as smooth conductors or dielectrics.

We propose path replay backpropagation (PRB) to address both limitations. Our method splits gradient evaluation into two separate passes: in a first step, we sample light paths as usual and record a small amount of information about each complete light path: in the simplest case, this is just the path's contribution to the (hypothetical) image. Then, we reset the pseudorandom number generator and recompute the *same light paths* once more, while backpropagating derivatives to scene parameters. The key idea of our approach is that certain steps of the computation can be inverted. By taking advantage

this invertibility and the data stored by the preceding pass, we are able to recover all information required by the adjoint phase on the fly.

The time complexity of the resulting algorithm has a linear dependence on path length, while computing the same unbiased derivatives as conventional AD with dramatically reduced memory usage. The two-pass scheme can furthermore generalize to derivatives of specular materials, requiring only a small amount of extra storage (a 4×4 matrix per sample) that remains independent of path length. Unlike conventional AD, PRB does not have any issues differentiating light paths of arbitrary length. This for the first time enables efficient computation of reverse-mode derivatives of unbiased volume rendering using delta tracking. In summary, our contributions are:

- A linear time, unbiased method for differentiable path tracing.
- A generalization of that algorithm to specular materials.
- The ability to efficiently differentiate unbiased volume transport based on delta tracking.

While originally motivated by the differentiable rendering problem, our path replay method can be applied to other problems of similar structure. To the best of our knowledge, this is a new differentiation approach that has not been proposed previously. We will discuss the fundamental mathematical idea behind path replay and also provide an outlook on its application outside of rendering.

In this chapter, we will first introduce the predecessor method (RB), as it is one of the main baselines against which we evaluate PRB. Both methods replace conventional automatic differentiation with a specialized path tracing loop that uses AD only for certain local quantities. We then describe our method in depth and present results and validations.

5.1 Background

5.1.1 Problem setting

Our goal is to efficiently compute derivatives of an image-based objective function g with respect to scene parameters. As seen in Section 4.4, by the chain rule, this derivative decomposes into a product of the objective function and pixel gradient:

$$\partial_\pi g(I(\pi)) = g'(I(\pi)) \;\; \partial_\pi I(\pi). \tag{5.1}$$

We would like to evaluate this product for a large number of scene parameters π. For notational convenience, we will again focus on such gradients with respect to a single scene parameter π, though all techniques developed will also support simultaneous optimization involving vast numbers of parameters.

For any method to scale to an arbitrary number of parameters, it needs to evaluate this derivative in reverse mode. Mathematically, we seek to implement the vector-Jacobian product of the derivative of the objective function and the Jacobian of the image-formation process $\mathbf{J}_I$:

$$\delta_\pi = \delta_g^T \mathbf{J}_I. \tag{5.2}$$

The derivative of the objective function with respect to the image can be computed using automatic differentiation. Conceptually, the term δ_g is an image, where each pixel value is the gradient of the objective function with respect to that pixel. For example, if the objective function is the mean squared error, then the derivative of the objective in each pixel of δ_g will simply be $2/M \cdot (I_j - I_j^{ref})$, where M is the number of pixels. The Jacobian J_I of the rendering process is of size $n \times p$, where p is the number of parameters. This Jacobian is oftentimes dense: due to the global nature of light transport, a single parameter might affect all pixels in the image. At their core, both the prior work and ours evaluate this Jacobian product without explicitly computing the Jacobian. However, reverse-mode AD still leads to extremely high memory usage due to the need to store checkpoints that are later accessed during reverse-mode differentiation.

In this chapter, we focus on efficient reverse-mode differentiation and disregard discontinuity handling via reparameterization or edge sampling. The combination of custom adjoints such as RB and PRB with a reparameterization of the visibility discontinuities was explored by Zeltner et al. [68]. We will showcase PRB for scenes and optimization problems that do not contain parameter-dependent discontinuities. Such examples are the reconstruction of participating media or the material properties of surfaces, while leaving the geometric representation of the scene fixed.

5.1.2 Radiative backpropagation

Since using conventional AD suffers from the aforementioned limitations, Nimier-David et al. [266] proposed exploiting properties unique to light transport and turn reverse-mode derivative propagation into an independent simulation that transports derivative radiation from sensors to objects with differentiable parameters. We briefly review how this works and point out issues of this transformation addressed by our work.

If we assume the use of a box pixel filter and only a single parameter π, the Jacobian product can be written as a sum over pixels:

$$\delta_\pi = \delta_g^T J_I = \sum_{j=1}^{M} \delta_{g,j}\ \partial_\pi I_j = \sum_{j=1}^{M} \frac{1}{N} \sum_{i=1}^{N} \delta_{g,j} \frac{\partial_\pi f_j(\mathbf{x}_i, \pi)}{p_j(\mathbf{x}_i)}, \tag{5.3}$$

where M is the number of pixels and $\delta_{g,j}$ is the component j of the image gradient (i.e., the value of pixel j in the "gradient image"). The integer N denotes the number of samples. For each Monte Carlo sample $\mathbf{x}_i$, we evaluate the ratio of the derivative of f_j and the PDF. This formulation computes a detached estimator, but we will later show how our method can also be extended to attached sampling.

This equation illustrates one of the key ideas of RB: we can compute gradients by emitting the adjoint quantity $\delta_{g,j}$ from each pixel independently. This already reduces the complexity of the problem, as we now just need to figure out how to compute the product of $\delta_{g,j}$ and the derivative of a single light path's contribution. Most importantly, this product has to be evaluated in a way that can scale to an arbitrary number of parameters.

At this point, the remaining issue is to differentiate the image-contribution function itself. At its core, this requires differentiating the radiance arriving at the sensor. We focus on the surface-only case for simplicity, where the light transport in a scene is characterized by the rendering equation [3]:

$$L_o(\mathbf{x}, \boldsymbol{\omega}_o) = L_e(\mathbf{x}, \boldsymbol{\omega}_o) + \int_{S^2} L_i(\mathbf{x}, \boldsymbol{\omega}_i) f_s(\mathbf{x}, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i)\, \mathrm{d}\boldsymbol{\omega}_i^\perp. \tag{5.4}$$

Here, L_o, L_e, and L_i refer to the outgoing, emitted, and incident radiance, and f_s is the BSDF. The product of incident radiance and BSDF is integrated over projected solid angles $\boldsymbol{\omega}_i^\perp$. Nimier-David et al. then apply the derivative operator ∂_π to both sides of this equation, which produces several terms via the product rule:

$$\begin{aligned} \partial_\pi L_o(\mathbf{x}, \boldsymbol{\omega}_o) = \partial_\pi L_e(\mathbf{x}, \boldsymbol{\omega}_o) + \int_{S^2} & \partial_\pi L_i(\mathbf{x}, \boldsymbol{\omega}_i)\ f_s(\mathbf{x}, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i) \\ & + L_i(\mathbf{x}, \boldsymbol{\omega}_i)\ \partial_\pi f_s(\mathbf{x}, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i)\, \mathrm{d}\boldsymbol{\omega}_i^\perp. \end{aligned} \tag{5.5}$$

This equation can be interpreted as another kind of energy balance that now involves *differential radiance* $\partial_\pi L_i$, $\partial_\pi L_e$, and $\partial_\pi L_o$. Inspecting the various terms, this differential radiance can then be seen to satisfy the following properties:

1. Differential radiance is emitted when the primal emission L_e depends on π.

2. Ordinary radiance generates differential radiance when it interacts with objects, whose BSDF depends on π.

3. Finally, once created, differential radiance scatters like ordinary light (i.e., involving the BSDF of scene objects).

A difficulty that might not be apparent from the above equation is that a separate set of differential radiance functions exist for *each* scene parameter. Starting from the gradient image δ_g, radiative backpropagation solves equation (5.5) by constructing light paths from the sensor, using the same sampling strategy as path tracing. Using only a single simulation, gradients for all scene parameters are accumulated simultaneously. In a sense, RB exploits the reciprocity of the scattering and transport operators [55]. Unlike reverse-mode AD, the gradient computation pass runs in the same direction as the original computation.

Limitations. Two issues become apparent when further scrutinizing this approach. First, The differential scattering equation (5.5) references the *primal incident radiance* L_i, coupling the primal and differential light transport problems so that both must now be solved at the same time. The standard approach for evaluating integral equations using recursive Monte Carlo sampling then requires recursion to handle both ∂L_i *and* L_i. This double recursion leads to a quadratic growth in the amount of computation as the maximum path length increases, which can be very costly when the scene involves significant multiple scattering. A similar quadratic complexity would be attained using AD if the checkpoints at each iteration were replaced by full recomputation of the required state. Avoiding this quadratic complexity was the motivation to store the intermediate state in the first place. While the recursion could be restricted by probabilistically choosing only one of the two terms, there is no obvious sampling scheme to do so. Finally, variance in such a stochastic scheme would increase exponentially due to this repeated sub-optimal choice.

Second, the method of Nimier-David et al. does not support materials containing Dirac delta functions (e.g. ideal specular BSDFs like smooth glass or metal surfaces). This case involves additional challenges that arise from coupling within *specular chains*, which refer to uninterrupted sequences of vertices along a light path involving such ideal specular materials.

Biased RB Nimier-David et al. suggest that to avoid quadratic time complexity, one can simply set the incident radiance L_i to 1 during differentiation. They argue that the resulting bias is innocuous because gradients will still have the correct component-wise *sign* and can thus be expected to converge to a local minimum when combined with

robust gradient-based optimization techniques. However, we find that both of these claims are generally incorrect: the sign may not match, and convergence is then also not guaranteed. In particular, we found that the sign is only correct in a very local sense for exactly the term $L_i(\mathbf{x}, \boldsymbol{\omega}_i)\, \partial_\pi f_s(\mathbf{x}, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i)$, in which the incident radiance specifies a positive multiplicative factor. This means that the sign of this product is preserved if the radiance is set to a value of one. In practice, many such local gradients are accumulated due to the global nature of light transport and scattering, and the combination of these various signed quantities is not guaranteed to result in a sum that still has the correct sign.

Figure 5.2 demonstrates this issue by optimizing an enclosed object. In this case, next event estimation is not available and the method must rely on the incident radiance term that was assumed to equal 1. This experiment shows that bias due to this approximation breaks the convergence even in a simple unimodal 1D optimization problem. Our method produces unbiased estimates and converges as expected while retaining the linear time complexity of biased RB.

Note that we use the color map of Figure 5.2 throughout this chapter: gray, blue, and red denote zero, positive, and negative-valued regions, respectively.

5.2 Method

We now explain the principles of our method, starting with the basic idea that we then expand into a more general principle to address the limitations noted in Section 5.1. We first focus on the *detached* case and postpone the more complex *attached* case to Section 5.2.2.

5.2.1 Replaying light paths to estimate derivatives

While Monte Carlo estimators are in principle random, real-world usage relies on deterministic pseudorandom number generators that can be rewound or reinitialized to produce an identical stream of variates. This turns out to be surprisingly useful because it enables running two variants of an algorithm that will perform an identical random walk. At the same time, these two algorithms can then use the generated path vertices for different purposes. In rendering, this approach has for example been used to generate correlated light paths in gradient-domain rendering [66, 268] or, more recently, to re-use light paths in interactive path tracing [269].

We build on this idea to run a regular forward path tracing pass that is followed by

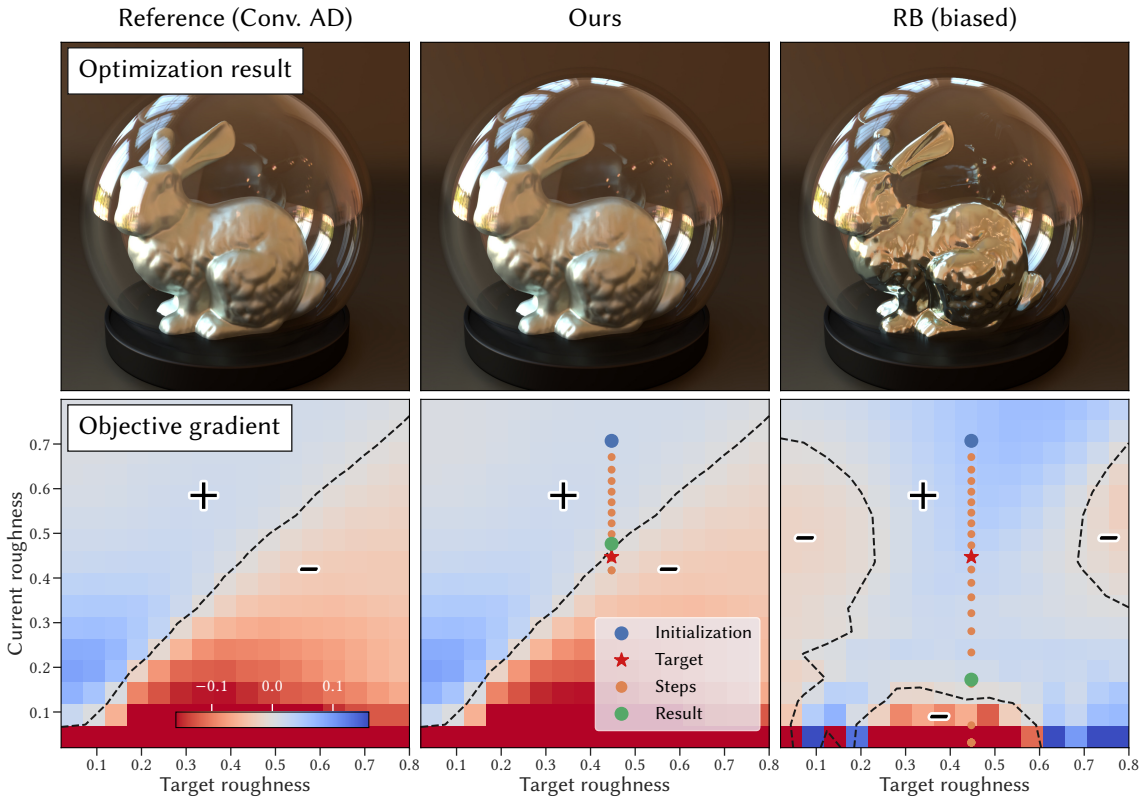


Figure 5.2: Analysis of Bias in radiative backpropagation (RB) [266]. We optimize the scalar roughness of an indirectly illuminated metal bunny contained inside a glass container. The top row shows the reference configuration and optimization results obtained by our method and biased RB. Not only did RB not converge here: it *cannot* converge regardless of initialization. Due to the scalar objective and 1D parameter space, we can exhaustively plot gradients for a range of initial and target roughness values (bottom row). A zero-valued gradient is expected on the diagonal, where initial and target roughness coincide. We also plot intermediate optimization steps for the optimization task of the top row. These visualizations show that the gradients computed by our method match the result of conventional AD. Biased RB is not only biased but also produces incorrect signs throughout the parameter space. Optimization will therefore fail even when initialized with the correct solution.

an adjoint pass visiting the same sequence of vertices. The first phase will determine the total amount of radiance accumulated by the path, and the second "random" walk can then exploit this information to precisely reconstruct the incident illumination at each path vertex. Our method produces the same result as automatic differentiation, while using a constant amount of storage and retaining the linear time complexity of regular forward path tracing. The results do not just match conventional AD in expectation: they produce the same noise pattern, with only small numerical differences related to the different evaluation in terms of IEEE-754 floating point arithmetic.

Our method is best explained using a few lines of pseudocode. To reduce the idea to its core, we will assume that emitters are static, and we will also work without standard optimizations like MIS or next event estimation. The following code fragment specifies a standard unidirectional path tracer to clarify all notation. It takes a ray as input and returns the resulting accumulated radiance. The code is heavily simplified and focuses on what is relevant for differentiation: changes to the path throughput β and the total radiance L accumulated along the path.

```
def sample_path(ray):
    L = 0, β = 1
    for i in range(N):
        L += β * L_e(...)
        ω_i, bsdf_value, bsdf_pdf = sample_bsdf(...)
        β *= bsdf_value / bsdf_pdf
        ray = spawn_ray(ω_i, ...)
    return L
```

Algorithm 5.1: Path tracer sketch, where β and L denote throughput and accumulated radiance. L is also an input of the adjoint pass discussed next.

The subsequent adjoint phase has the purpose of back-propagating *adjoint radiance* δ_L along the same light path: this is the derivative of radiance with respect to the optimization's objective function, which captures how the radiance along this specific path should change to improve the objective's current value. In the case of a box pixel filter, this adjoint radiance is simply $\delta_L = \delta_{g,j}/N$. We now must evaluate the terms of the differential transport problem outlined in Equation 5.5. Due to the simplifying assumption of static emitters, this equation further reduces to

$$\begin{aligned}\partial_\pi L_o(\mathbf{x}, \boldsymbol{\omega}_o) = \int_{S^2} &\partial_\pi L_i(\mathbf{x}, \boldsymbol{\omega}_i)\ f_s(\mathbf{x}, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i) \\ &+ L_i(\mathbf{x}, \boldsymbol{\omega}_i)\ \partial_\pi f_s(\mathbf{x}, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i)\, \mathrm{d}\boldsymbol{\omega}_i^\perp. \end{aligned} \tag{5.6}$$

To recapitulate, the first term of this equation states that differential radiation propagates according to a standard random walk, and the adjoint phase simply handles this part using a loop. The second term must be handled specially and requires backpropagating the product of adjoint and incident radiance into the BSDF f_s. For our algorithm, Equation 5.6 is helpful to describe which terms we need to evaluate. However, all we do is differentiate the contribution of the path that was already sampled in the first pass. Replaying the path is simply a convenient way of accessing the required terms, but all the sampling decisions are simply repeated from the first pass.

The following code fragment implements the path tracer's adjoint phase. The function `backward_grad(expr, grad_out)` evaluates the reverse-mode derivative of the expression `expr` to propagate a gradient with respect to the expression's output (`grad_out`) towards the scene parameters, returning another gradient resulting from this step.

```
1 def sample_path_adjoint(ray, L, δ_L):
2     β = 1
3     for i in range(N):
4         L -= β * L_e(...)
5         ω_i, bsdf_value, bsdf_pdf = sample_bsdf(...)
6         δ_π += backward_grad(bsdf_value, δ_L * L / bsdf_value)
7         β *= bsdf_value / bsdf_pdf
8         ray = spawn_ray(ω_i, ...)
9     return δ_π
```

Algorithm 5.2: The adjoint phase takes L as input and reverses certain operations in `sample_path()` to propagate adjoint radiance δ_L to scene parameters.

Line 4 of the adjoint pass reconstructs the incident illumination at the current vertex by *subtracting* emission, if present. This is the inverse of line 4 in Listing 5.1. The resulting modified value L still includes the `bsdf_value` / `bsdf_pdf` ratio that was applied in line 6 of Listing 5.1.

Our goal is now to evaluate the term $L_i(\ldots)\ \partial f_s(\ldots)$ of the differential scattering equation (5.6), for which we reuse the existing sample ω_i with density `bsdf_pdf`. This entails dividing out the BSDF value but retaining the reciprocal density and back-propagating the product of this quantity with the adjoint radiance δL into the scene parameters corresponding to the current BSDF f_s.

Several aspects of this algorithm are remarkable: it is mathematically equivalent to reverse-mode AD that normally requires a complete program reversal including ample storage of intermediate values. Yet, the loop of our adjoint pass retains its order, exploit-

ing the repeatability of the random walk along with a minimal use of extra information (L) computed in a prior phase.

We note that Listing 5.2 involves a subtraction, which can lead to floating point cancellation errors. This case could arise when a path vertex has an extremely high contribution that dominates that of other path vertices. When processing this vertex in the adjoint pass, cancellation could cause L to round to zero. Arguably, this corresponds to a situation where the numerical accuracy of the primary simulation is also suspect.

Next event estimation can be integrated into this framework by carefully reconstructing the direct illumination at each vertex and performing the inverse of the primal operations that are needed to accomplish this. The pseudocode in Listing 5.3 shows an implementation considering next event estimation and surface emission derivatives. The code assumes that we have a function `sample_emitter` that samples an emitter contribution L_{NEE}. Adding MIS to this algorithm is straightforward: since this estimator is detached, MIS weights are simply constant factors multiplied into emitter contributions. We omit the MIS weights here for brevity.

```
def sample_path_adjoint_nee(ray, L, δ_L):
    β = 1
    for i in range(N):
        L_NEE = sample_emitter(...)
        δ_π += backward_grad(L_e(...) + L_NEE, δ_L * β)
        L -= β * (L_e(...) + L_NEE)

        ω_i, bsdf_value, bsdf_pdf = sample_bsdf(...)
        δ_π += backward_grad(bsdf_value, δ_L * L / bsdf_value)
        β *= bsdf_value / bsdf_pdf
    return δ_π
```

Algorithm 5.3: The adjoint phase if we also consider emission and next event estimation derivatives.

5.2.2 Attached sampling strategies

While the previously discussed method works well in a wide range of situations, a detached estimator cannot compute parameter derivatives related to perfectly specular surfaces, as this would entail backpropagating gradients through Dirac delta functions. This prevents optimizing index of refraction, geometry, or surface normals of smooth conductors and dielectrics, including interesting applications like caustic design or the

inverse reconstruction of transparent objects [188]. The detached estimator might also result in high variance for nearly specular surfaces with low roughness.

In this section, we show how our path replay method can be extended to handle attached estimators. As first introduced in Section 4.4, attached estimators differentiate the integral *after* transforming to primary sample space:

$$\partial_\pi I_j(\pi) = \partial_\pi \int_{\mathcal{P}} f_j(\mathbf{x}, \pi)\, \mathrm{d}\mathbf{x} = \int_{\mathcal{U}} \partial_\pi \left[\frac{f_j(T(\mathbf{u}, \pi), \pi)}{p(T(\mathbf{u}, \pi), \pi)} \right] \mathrm{d}\mathbf{u}, \tag{5.7}$$

where $T\colon \mathcal{U} \to \mathcal{P}$ maps from the unit hypercube $\mathcal{U} = [0, 1]^n$ to the original space $\mathcal{P}$. The mapping $\mathbf{x} = T(\mathbf{u})$ is constructed from a target density $p(\mathbf{x})$ so that its Jacobian determinant satisfies $|\mathbf{J}_T(\mathbf{u})| = p(\mathbf{x})^{-1}$. Our aim is now to differentiate not only the integrand itself, but also the parameter-dependent mapping T and PDF.

Applying the ideas of Section 5.2.1 poses additional challenges compared to the previously discussed detached case. Consider a perturbation of the shading normal or a material parameter at a given surface interaction: this change will further propagate, influencing the geometry of the remainder of this path. This will in turn also change the value of subsequent BSDF and emission terms. It is important to correctly account for these non-local dependencies during differentiation to ensure unbiased gradient estimates. Fortunately, it remains possible to follow the same core idea to develop similar methods for the attached case.

To motivate our approach, let us apply the partial derivative to the various terms in Equation (5.7):

$$\partial_\pi I_j(\pi) = \int_{\mathcal{U}} \frac{\partial_\pi f_j(T(\mathbf{u}, \pi), \pi)}{p(T(\mathbf{u}, \pi), \pi)} - \frac{f_j(T(\mathbf{u}, \pi), \pi)\ \partial_\pi p(T(\mathbf{u}, \pi), \pi)}{p(T(\mathbf{u}, \pi), \pi)^2}\, \mathrm{d}\mathbf{u}, \tag{5.8}$$

where we have applied the quotient rule. Focusing only on the first term, we can apply the chain rule to see that it further expands to:

$$= \int_{\mathcal{U}} \frac{\partial_\pi f_j(\ldots) + \partial_{\mathbf{x}} f_j(\ldots) \cdot \partial_\pi T(\ldots)}{p(\ldots)} - \cdots\, \mathrm{d}\mathbf{u}. \tag{5.9}$$

This derivation shows how the numerator splits into a term that tracks the change of the primal integrand with respect to π, and the following product of two Jacobian matrices expresses how changes in the parameterization influence f_j due to its dependence on the light path vertices $\mathbf{x}$.

We can also derive a recursive energy balance equation for the attached case, similar to Equation 5.5 which produced detached estimators. We start by expressing the render-

ing equation as an integral over primary sample space (ignoring emission for brevity):

$$\begin{aligned} L_o(\mathbf{x}, \boldsymbol{\omega}_o) &= \int_{\mathcal{S}^2} L_i(\mathbf{x}, \boldsymbol{\omega}_i) f_s(\mathbf{x}, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i) \, \mathrm{d}\boldsymbol{\omega}_i^{\perp} \\ &= \int_{\mathcal{U}^2} L_i(\mathbf{x}, T(\mathbf{u}, \pi)) \frac{f_s(\mathbf{x}, \boldsymbol{\omega}_o, T(\mathbf{u}, \pi))}{p(\mathbf{x}, \boldsymbol{\omega}_o, T(\mathbf{u}, \pi))} \, \mathrm{d}\mathbf{u}, \end{aligned} \tag{5.10}$$

where T maps uniformly distributed random variates $\mathbf{u} \in [0, 1]^2$ to sampled directions and p is the solid angle density of the generated samples. Similar to the detached case, we apply the derivative operator ∂_π to this integral:

$$\begin{aligned} \partial_\pi L_o(\mathbf{x}, \boldsymbol{\omega}_o) = \int_{\mathcal{U}^2} & L_i(\mathbf{x}, T(\mathbf{u}, \pi)) \; \partial_\pi \left[\frac{f_s(\mathbf{x}, \boldsymbol{\omega}_o, T(\mathbf{u}, \pi))}{p(\mathbf{x}, \boldsymbol{\omega}_o, T(\mathbf{u}, \pi))} \right] \\ & + \partial_\pi L_i(\mathbf{x}, T(u, \pi)) \; \frac{f_s(\mathbf{x}, \boldsymbol{\omega}_o, T(\mathbf{u}, \pi))}{p(\mathbf{x}, \boldsymbol{\omega}_o, T(\mathbf{u}, \pi))} \\ & + \partial_{\boldsymbol{\omega}_i} L_i(\mathbf{x}, T(\mathbf{u}, \pi)) \cdot \partial_\pi \left[T(\mathbf{u}, \pi) \right] \; \frac{f_s(\mathbf{x}, \boldsymbol{\omega}_o, T(\mathbf{u}, \pi))}{p(\mathbf{x}, \boldsymbol{\omega}_o, T(\mathbf{u}, \pi))} \, \mathrm{d}\mathbf{u}, \end{aligned} \tag{5.11}$$

where we used the product and chain rule to split the derivative into three separate terms. The first two terms involve the primal incident radiance and its parametric derivative. They resemble the detached case and can therefore be evaluated analogously. However, the third term is new and requires the *directional derivative* of the incident radiance.

These directional derivatives constitute the main change due to attached sampling strategies, and our approach will be to similarly reconstruct them on the fly using a constant amount of precomputed information generated by a prior phase. In contrast to the previous algorithm, we therefore not only need to track quantities related to the total accumulated radiance and throughput, but also to the path geometry. In each iteration, we spawn a ray according to the BSDF sampling strategy, and the computed derivatives must then take into account how this new ray will depend on the previous ray.

Specifically, we must compute the Jacobian relating subsequent vertices $(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2)$ in a light path. We assume that $\mathbf{x}_2$ is sampled given an incident ray $\mathbf{x}_0 \rightarrow \mathbf{x}_1$, as illustrated in Figure 5.3. We must then compute the Jacobian capturing the differential relationship $\partial(\mathbf{x}_2, \mathbf{x}_1)/\partial(\mathbf{x}_1, \mathbf{x}_0)$ of these path vertices. Performed naïvely, this would be a 6×6 matrix. However, as we are only interested in rays originating on surfaces (or the camera), and directions are normalized, we can reduce it to a 4×4 matrix by switching to a 2D parametrization using coordinates that leverage the available tangential basis vectors at path vertices.

It is interesting to note that these derivatives can in principle be computed using any kind of parameterization of the space of light paths and rays, and we initially used a position-angle parameterization of rays. In this case, the Jacobian matrices involve

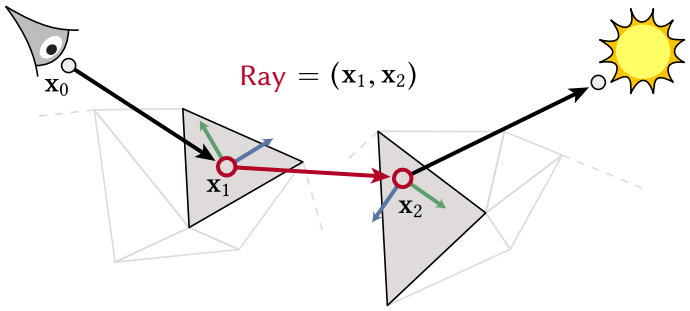


Figure 5.3: The *attached* version of our method must consider how the geometry of a light path changes with respect to infinitesimal perturbations of scene parameters. Our method does so by tracking the differential relationship of a subsequent path vertex $\mathbf{x}_2$ generated from an incident path segment $\mathbf{x}_0 \rightarrow \mathbf{x}_1$ using the combination of BSDF sampling routines and ray tracing.

components with incompatible units and different resulting scales, which can cause poor numerical conditioning. Switching to a position-position parameterization addressed this issue.

Pseudocode Listing 5.4 contains the pseudocode of our attached gradient evaluation technique following the same conventions previously outlined in Listings 5.1 and 5.2. Each loop iteration uses forward-mode differentiation (via a function `forward_grad()`) to capture the differential relationship of adjacent path vertices and reverse-mode differentiation (via a function `backward_grad()`) to backpropagate adjoint radiance into the scene parameter gradient.

Concretely, the function `backward_grad(`$\mathbf{x}$`, `δ`)` propagates the gradient of the function's output to the input parameters by evaluating the Jacobian product $\mathbf{J}_{\mathbf{x}}^{\mathsf{T}}\delta$ using automatic differentiation. On the other hand, the function `forward_grad()` takes an input variable and several output variables and computes the Jacobian matrices of all the output variables with respect to the input. This function returns Jacobian matrices, while `backward_grad()` internally multiplies the Jacobian with vector. By $\mathbf{J}_\circ$ we denote Jacobian matrices of some output variable $\circ$ with respect to the current "ray". Here, a ray is represented by the tuple of UV coordinates at its origin and intersection location. For RGB values, e.g., the accumulated radiance L, these Jacobians have 3 rows and 4 columns. The implementation keeps track of the derivative of the accumulated radiance, path throughput and the next ray. We denote these quantities by $\mathbf{J}_L$, $\mathbf{J}_\beta$ and $\mathbf{J}_{\text{ray}}$. For example, $\mathbf{J}_{\text{ray}}$ contains the derivatives of the UV coordinates at bounce i and $i+1$, $(u_i, v_i, u_{i+1}, v_{i+1})$, with respect to the camera ray, represented by the tuple (u_0, v_0, u_1, v_1). The first pass then returns the accumulated radiance L and its directional derivative $\mathbf{J}_L$ to be used in the adjoint phase.

```
def sample_path(ray):
    L = 0, β = 1
    J_L = 0_{3,4}, J_β = 0_{3,4}, J_ray = I_4
    for i in range(N):
        L += β * L_e(...)
        ω_o, bsdf_value, bsdf_pdf = sample_bsdf(...)
        bsdf_weight = bsdf_value / bsdf_pdf
        ray′ = spawn_ray(ω_o, ...)
        # Accumulate the directional radiance derivative
        J_ray′, J_bsdf, J_{L_e} = forward_grad(ray, {ray′, bsdf_weight, L_e})
        J_bsdf = J_bsdf @ J_ray
        J_{L_e} = J_{L_e} @ J_ray
        J_ray = J_ray′ @ J_ray
        J_L += β * J_{L_e} + L_e * J_β
        J_β = bsdf_weight * J_β + β * J_bsdf
        β *= bsdf_weight
    return L, J_L

def sample_path_adjoint(ray, L, J_L, δ_L):
    β = 1, J_ray = I_4
    for i in range(N):
        L -= β * L_e(...)
        ω_o, bsdf_value, bsdf_pdf = sample_bsdf(...)
        bsdf_weight = bsdf_value / bsdf_pdf
        ray′ = spawn_ray(ω_o, ...)
        J_ray′, J_bsdf, J_{L_e} = forward_grad(ray, {ray′, bsdf_weight, L_e})
        J_bsdf = J_bsdf @ J_ray
        J_{L_e} = J_{L_e} @ J_ray
        J_ray = J_ray′ @ J_ray

        # Update the directional radiance derivative
        J_L -= L / bsdf_weight * J_bsdf + β * J_{L_e}
        J′_L = J_L @ J_ray^{-1}

        # Backpropagate gradients of the current BSDF weight
        δ_π += backward_grad(bsdf_weight, δ_L * L / bsdf_weight)
        # Backpropagate through shading frame and BSDF sampling calculation
        δ_π += backward_grad(ray′, δ_L @ J′_L)
        β *= bsdf_weight
    return δ_π
```

Algorithm 5.4: Pseudocode of our attached backpropagation algorithm.

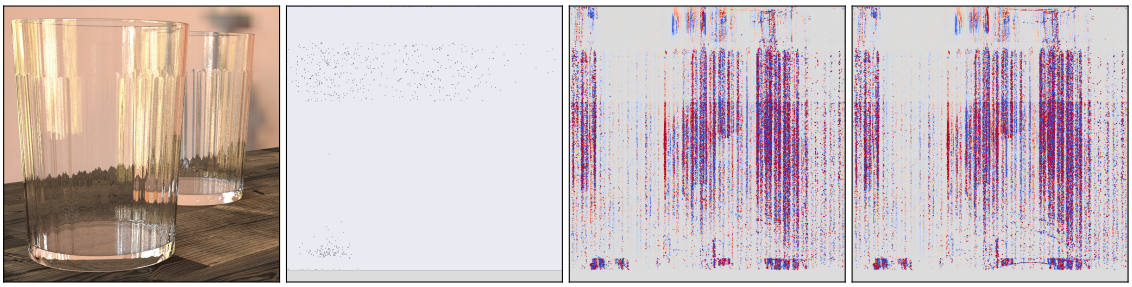
Note that automatic differentiation is only used *within* the loop body, and does not need to build an AD graph over loop iterations. Our explicitly computed Jacobian matrices account for all derivative information that is needed across loop iterations.

Practical considerations We always perform three rendering steps: the first computes an ordinary primal image and uses a pseudorandom number seeding scheme that de-correlates it from the subsequent two gradient-related evaluation steps. Recall from Section 4.4 that we need to decorrelate primal and gradient rendering to eliminate, or at least reduce, gradient bias. The primal image is then used to evaluate the objective function and compute its gradient, which produces the *adjoint image* (i.e., the derivative of the rendered image with respect to the optimization objective) that we then backpropagate from the sensor to objects with differentiable parameters. These two steps rely on the coupled pair of algorithms presented earlier, which are perfectly correlated in the sense that they visit the exact same sequence of path vertices.

Compared to conventional AD or radiative backpropagation, we perform one additional rendering pass. This comes at an extra cost, which means that our linear-time approach may ultimately be slower than existing two-pass methods when the problem to be solved is sufficiently simple so that lack of linear computation time or constant memory usage are not bottlenecks.

Stochastic regularization The Jacobian matrix relating adjacent path segments is singular when the sampling strategy lacks a dependence on the previous vertex. This case for example arises following an interaction with a diffuse material, whose sampling strategy does not depend on the incident direction.

We use the following simple regularization scheme to avoid numerical failure during the inversion process: at each vertex, we add a small amount of statistically independent noise with zero mean, which does not bias the resulting gradients. In particular, we add a diagonal matrix $\lambda \cdot \mathbf{I}_4 \cdot \text{sign}\,(u - 1/2)$ to the ray Jacobian, where $u \sim \mathcal{U}(0, 1)$ is a uniform variate, $\mathbf{I}_4$ the 4×4 identity matrix and $\lambda = 0.01$ denotes a regularization weight. We again make sure to use the exact same random variates in both of our computation passes (i.e., the second and third passes according to the unbiased evaluation scheme discussed above). By using the *same* noise matrix in both passes, we ensure that our method remains unbiased. A detailed proof is provided in Appendix B. The effect of this regularization is illustrated in Figure 5.4.



(a) Reference (b) Ours (no regularization) (c) Ours (d) Conventional AD

Figure 5.4: We show the effect of our random regularization of the ray Jacobian. In this example, we computed the gradient of the surface normals of the ridges on the glass cups. Without stochastic regularization, the diffuse material of the wooden table produces singular Jacobian matrices, and the resulting gradients are mostly invalid (NaN, shown as white). With a small amount of regularization ($\lambda = 0.01$), our gradient estimates are very close to those computed using conventional automatic differentiation.

Moving discontinuities So far we have assumed that discontinuities, when present, do not depend on π and are thus *static* within the integration domain. A serious issue can arise when the π-dependent parameterization introduces an undesirable parameter dependence in previously static discontinuities. Without additional precautions, such a combination can then lead to biased gradients. Zeltner et al. [68] suggest (smoothly) replacing attached estimators with detached versions near discontinuities. This is done by leveraging a convolution to detect occlusion edges, similar to the reparameterization work by Bangaru et al. [69] discussed earlier. This adds some implementation complexity and computational cost. In practice, bias due to naïve attached sampling may be acceptable given the simplicity and efficiency of the uncorrected approach. For example, in the Mitsuba 2 paper [48], we demonstrated a working caustic design application that applies AD to an entire specular chain, which is functionally equivalent to an attached sampler without corrections for moving discontinuities. We will pay no further attention to discontinuities here, as our focus is purely on the time and storage complexity of reverse-mode attached derivatives.

5.2.3 Differentiable delta tracking

Path replay backpropagation unlocks the door to efficient reverse-mode differentiation of volume transport based on unbiased null-collision methods like delta tracking [96], which we introduced in Section 3.5.2.

Differentiating such algorithms in reverse mode previously involved three separate sources of difficulty: first, light paths in participating media are generally much longer and can reach 100-1000s of scattering interactions, exacerbating the difficulty of pro-

gram reversal. Second, null-scattering can introduce a large number of additional *null* interactions that expand the size of the intermediate program state even further. Third, null-scattering makes discrete decisions that require additional precautions during differentiation.

The former two issues are easily addressed by switching to detached or attached PRB with generalizations for volumes (e.g., scattering by a phase function in addition to the BSDF). To resolve the third issue, let's recall the null-scattering integral form of the radiative transfer equation [97, 98]:

$$\begin{aligned} L_i(\mathbf{x}) &= \int_0^\infty \bar{\sigma}\, \bar{\mathrm{T}}(\mathbf{x}, \mathbf{x}_t) \left[\frac{\sigma_a(\mathbf{x}_t)}{\bar{\sigma}} L_e(\mathbf{x}_t) + \frac{\sigma_s(\mathbf{x}_t)}{\bar{\sigma}} L_s(\mathbf{x}_t) + \frac{\sigma_n(\mathbf{x}_t)}{\bar{\sigma}} L_i(\mathbf{x}_t) \right] \mathrm{d}t \\ &= \int_0^\infty p(t) \left[P_a(\mathbf{x}_t) L_e(\mathbf{x}_t) + P_s(\mathbf{x}_t) L_s(\mathbf{x}_t) + P_n(\mathbf{x}_t) L_i(\mathbf{x}_t) \right] \mathrm{d}t, \end{aligned} \tag{5.12}$$

where we have omitted the dependence on $\boldsymbol{\omega}$ and surface-related terms for readability. The majorant transmittance $\bar{\mathrm{T}}(\mathbf{x}, \mathbf{x}_t)$ is the homogeneous transmittance according to the extinction majorant $\bar{\sigma}$. We write the integral in this particular way as it aligns well with the delta tracking algorithm: the term $\bar{\sigma}\, \bar{\mathrm{T}}(\mathbf{x}, \mathbf{x}_t)$ is exactly the free-flight density $p(t)$ implied by the majorant. The radiance terms inside the brackets are all weighted by terms of the form $\sigma_\circ / \bar{\sigma} =: P_\circ$. These conveniently sum to 1, so delta tracking can estimate this integral by evaluating one term at a time, sampled according to the probabilities $P_\circ$:

$$\begin{aligned} L_i(\mathbf{x}) \approx\ & L_e(\mathbf{x}_t) \mathcal{H}[u < P_a(\mathbf{x}_t)] \\ & + L_s(\mathbf{x}_t) \mathcal{H}[P_a(\mathbf{x}_t) \le u < 1 - P_n(\mathbf{x}_t)] \\ & + L_i(\mathbf{x}_t) \mathcal{H}[1 - P_n(\mathbf{x}_t) \le u], \end{aligned} \tag{5.13}$$

where the Heaviside function $\mathcal{H}$ equals 1 if the specified condition is satisfied and 0 otherwise. The variable t is sampled according to $p(t)$ and $u \sim \mathcal{U}(0, 1)$. In the primal estimator, all probabilities simply cancel out with the corresponding terms in the integrand.

We assume the majorant to be a constant and to not participate in the differentiation process. We can form a detached derivative estimator by explicitly only computing the derivative of the integrand, but not of the discrete sampling step:

$$\begin{aligned} \partial_\pi L_i(\mathbf{x}) \approx\ & \frac{\partial_\pi [P_a(\mathbf{x}_t)\, L_e(\mathbf{x}_t)]}{P_a(\mathbf{x}_t)} \mathcal{H}[u < P_a(\mathbf{x}_t)] \\ & + \frac{\partial_\pi [P_s(\mathbf{x}_t)\, L_s(\mathbf{x}_t)]}{P_s(\mathbf{x}_t)} \mathcal{H}[P_a(\mathbf{x}_t) \le u < 1 - P_n(\mathbf{x}_t)] \\ & + \frac{\partial_\pi [P_n(\mathbf{x}_t)\, L_i(\mathbf{x}_t)]}{P_n(\mathbf{x}_t)} \mathcal{H}[1 - P_n(\mathbf{x}_t) \le u], \end{aligned} \tag{5.14}$$

Note that all functions $P_\circ$ and $L_\circ$ can potentially depend on π. The particle proportions $P_\circ$ occur twice in each row, but they must only be differentiated in the numerator, as required by the detached estimator.

Practical implementations of this method will likely also need to make use of next event estimation. One additional challenge that arises here is that transmittance evaluation towards the sampled light position using a method like ratio tracking [10] adds another recursive loop over an unbounded number of scattering events. To correctly differentiate this entire process, we rely on a second nested application of PRB within the volumetric path tracer loop.

We provide pseudocode for the primal delta tracking estimator with next event estimation in Listing 5.5. For simplicity, the pseudocode does not handle volumetric emission. The code assumes that we access medium parameters using a function σ_t`(x)` and a function `albedo(x)` which query medium extinction and albedo, respectively. The function `sampler.rand()` generates a uniformly distributed random variable. The `sample_emitter` function samples a direction towards an emitter and evaluates the emitter radiance weighted by the phase function and sampling PDF. The resulting L_e is then attenuated using the ratio tracking transmittance estimate computed in the following lines. Listing 5.6 shows the pseudocode for a detached delta tracking gradient estimator with next event estimation. Compared to the surface case, it handles the discrete sampling decisions due to delta tracking and contains nested PRB for the ratio tracking loop.

We have not yet experimented with *attached* versions of volume transport estimators but consider them an interesting avenue for future work, especially to handle the directional domain of the in-scattering integral: light paths in forward-scattering media ($g > 0.99$) are highly constrained and reminiscent of specular chains in the surface case. Simple detached sampling strategies can be a poor choice for such a directionally peaked integrand [68]. Our differentiable delta tracking also does not account for density gradients in empty space. This important special case needs to be handled by explicitly sampling medium interactions at points where $\sigma_t = 0$, as done by follow-up work [270]. This improved sampling technique slightly improves reconstruction results on some scenes.

```
def sample_path_delta_tracking(ray):
    L = 0, β = 1, x = ray.o
    while active path:
        # Sample the free-flight distance
        t = -log(1 - sampler.rand()) / σ̄
        x += t * ray.d
        # Continue in next iteration if a null interaction was sampled
        if sampler.rand() < 1 - σ_t(x) / σ̄:
            continue
        β *= albedo(x)

        # Sample emitter direction towards an emitter and emitter contribution
        L_e, ω_e = sample_emitter(...)

        # Attenuate emitter contribution by ratio tracking transmittance estimator
        x_e = x, tr = 1
        while not reached emitter:
            t = -log(1 - sampler.rand()) / σ̄
            x_e += t * ω_e
            tr *= 1 - σ_t(x_e) / σ̄ # ratio tracking transmittance estimate
        L += β * tr * L_e
        ω_i, phase_value, phase_pdf = sample_phase_function(...)
        β *= phase_value / phase_pdf
        ray = spawn_ray(x, ω_i)
    return L
```

Algorithm 5.5: Pseudocode of a volumetric path tracer with delta tracking and ratio tracking next event estimation.

```
def sample_path_delta_tracking_adjoint(ray, L, δ_L):
    β = 1, x = ray.o
    while active path:
        t = -log(1 - sampler.rand()) / σ̄
        x += t * ray.d
        # Backpropagate for null interaction
        if sampler.rand() < 1 - σ_t(x) / σ̄:
            P_n = 1 - σ_t(x) / σ̄
            δ_π += backward_grad(P_n, δ_L * L / P_n)
            continue
        β *= albedo(x)

        # Backpropagate for real interaction
        P_t = σ_t(x) / σ̄
        δ_π += backward_grad(P_t * albedo(x), δ_L * L / (P_t * albedo(x)))

        L_e, ω_e = sample_emitter(...)
        x_e = x, tr = 1
         # Create a copy of the random number generator with its current state
        nee_sampler = sampler.copy()
        while not reached emitter:
            t = -log(1 - nee_sampler.rand()) / σ̄
            x_e += t * ω_e
            tr *= 1 - σ_t(x_e) / σ̄
        x_e = x
        nee_sampler = sampler.copy()
        while not reached emitter:
            t = -log(1 - nee_sampler.rand()) / σ̄
            x_e += t * ω_e
            r = 1 - σ_t(x_e) / σ̄
            δ_π += backward_grad(r, δ_L * β * tr * L_e / r)
        L -= β * tr * L_e
        ω', phase_value, phase_pdf = sample_phase_function(...)
        β *= phase_value / phase_pdf
        # Accumulate phase function gradient
        δ_π += backward_grad(phase_value, δ_L * L / phase_value)
        ray = spawn_ray(x, ω')
    return δ_π
```

Algorithm 5.6: Pseudocode for the adjoint pass of the volumetric path tracer using delta tracking and ratio tracking.

5.2.4 General principles

Iterative jacobian inversion. The previously discussed method performs an iterative reconstruction of the incident illumination along with its directional derivative in the attached case. We can further generalize this idea using an abstract view that interprets sampling of a light path as the repeated composition of a function h. In an implementation, h would represent the body of a `for` loop that takes the previous iteration's state as input to compute an updated set of state variables. In the basic path tracer from Listing 5.1, the relevant loop state is comprised of the value L and throughput β. The loop then evaluates

$$(L(\pi), \beta(\pi)) = \underbrace{h(\pi, h(\pi, ..., h(\pi, L_0, \beta_0))\cdots)}_{N} = h^{(N)}(\pi, L_0, \beta_0), \tag{5.15}$$

where $L_0 = 0$ and $\beta_0 = 1$. In this particular example, the function was defined as $h(L, \beta) = (L + \beta \cdot L_e(...), \beta \cdot f_s(...))$. For notational clarity, we omit the division of the BSDF value $f_s(...)$ by the sampling density. Using the chain rule, we can take the derivative of Equation (5.15) with respect to the scene parameter π:

$$\begin{aligned}\partial_\pi \left(L(\pi), \beta(\pi)\right) &= \partial_\pi \left[h(\pi, h(\pi, ..., h(\pi, L_0, \beta_0))\cdots)\right] \\ &= \sum_{k=1}^{N} \left[\prod_{j=k+1}^{N} \mathbf{J}_h(L_j, \beta_j)\right] \partial_\pi h(\pi, L_{k-1}, \beta_{k-1}),\end{aligned} \tag{5.16}$$

where $(L_k, \beta_k) = h^{(k)}(\pi, L_0, \beta_0)$ and $\mathbf{J}_h(L_j, \beta_j)$ is the Jacobian of h. This expression can be evaluated in different ways. Conventional reverse-mode AD techniques would store the function evaluations after each iteration of the forward loop and then compute the product of the Jacobians in reverse order. This is for example how the backpropagation algorithm for neural networks [117] operates. Alternatively, we could run the computation in reverse and recompute quantities from the output end.

We choose a different approach: instead of inverting the primal calculation, we invert the local Jacobian matrix relating the loop state of adjacent iterations. This is feasible, as the involved quantities are low-dimensional and the computation has a simple structure. The same approach would clearly not scale to neural networks with thousand-dimensional latent representations in intermediate layers. For the detached path tracer, the Jacobian $\mathbf{J}_h$ of the function h is given by

$$\mathbf{J}_h = \begin{pmatrix} 1 & L_e(...) \\ 0 & f_s(...) \end{pmatrix}, \tag{5.17}$$

and we define $\mathbf{J}_{h,k}$ as product of $\mathbf{J}_h$ over the path suffix at vertex k:

$$\mathbf{J}_{h,k} = \prod_{j=k+1}^{N} \mathbf{J}_h(L_j, \beta_j) = \begin{pmatrix} 1 & L_{k,N} \\ 0 & \beta_{k,N} \end{pmatrix}. \tag{5.18}$$

The subscript k, N denotes quantities that are accumulated from path vertex k to N. This shows that the indirect illumination term that needs to be evaluated during backpropagation can be interpreted as an entry of the Jacobian product. Plugging this back into the expression for $\partial_\pi L$ in Equation (5.16), we obtain

$$\begin{aligned} \partial_\pi \left(L(\pi), \beta(\pi)\right) &= \sum_{k=1}^{N} \begin{pmatrix} 1 & L_{k,N} \\ 0 & \beta_{k,N} \end{pmatrix} \partial_\pi h(\pi, L_{k-1}, \beta_{k-1}) \\ &= \sum_{k=1}^{N} \begin{pmatrix} 1 & L_{k,N} \\ 0 & \beta_{k,N} \end{pmatrix} \begin{pmatrix} \beta_{k-1} \partial_\pi L_e \\ \beta_{k-1} \partial_\pi f_s \end{pmatrix} = \sum_{k=1}^{N} \beta_{k-1} \begin{pmatrix} 1 & L_{k,N} \\ 0 & \beta_{k,N} \end{pmatrix} \begin{pmatrix} \partial_\pi L_e \\ \partial_\pi f_s \end{pmatrix}. \end{aligned}$$

In practice, only the first component $\partial_\pi L$ of the gradient is desired, and we do not need the derivative of the path throughput β. Each iteration of the adjoint pass in Listing 5.2 then accumulates one of the elements of the previous sum into the scene parameter gradient. This equation now enables a new interpretation of the method presented in Section 5.2.1: the incident radiance computed in a first forward pass provides the Jacobian product $J_{h,0}$. As we now run our backward pass and subtract the current emitted radiance, we are effectively iteratively applying the inverse Jacobian matrix:

$$\mathbf{J}_h^{-1} = \begin{pmatrix} 1 & -L_e(\ldots)/f_s(\ldots) \\ 0 & 1/f_s(\ldots) \end{pmatrix}. \tag{5.19}$$

The throughput-related computation simplifies to a single division by the current BSDF value.

Fredholm integral equations. Another perspective on path replay backpropagation is to consider the broader class of integral problems this algorithm can be applied to. Both surface and volume rendering equations are *Fredholm integral equations of the second kind*. These equations have the general form:

$$\varphi(\mathbf{x}, \pi) = S(\mathbf{x}, \pi) + \int_{\mathcal{Y}} K(\mathbf{x}, \mathbf{y}, \pi)\, \varphi(\mathbf{y}, \pi)\, \mathrm{d}\mathbf{y}, \tag{5.20}$$

where S is a source term and K is the kernel. Both these terms can potentially be parameter dependent. The function φ is unknown and estimated using recursive Monte Carlo

integration. In rendering, S subsumes emission terms and K any terms related to BSDFs and phase functions. We can apply the derivative operator to the above equation:

$$\partial_\pi \varphi(\mathbf{x}, \pi) = \partial_\pi S(\mathbf{x}, \pi) + \int_y \partial_\pi K(\mathbf{x}, \mathbf{y}, \pi)\ \varphi(\mathbf{y}, \pi) + K(\mathbf{x}, \mathbf{y}, \pi)\ \partial_\pi \varphi(\mathbf{y}, \pi)\, \mathrm{d}\mathbf{y}. \tag{5.21}$$

From this, it seems that path replay, at least in its detached version, should generalize to any problem that can be expressed as such a Fredholm integral equation. For some problems, handling discontinuities might be essential and challenging. The applicability of the attached path replay backpropagation depends on the involved dimensionalities and numerical properties of the problem at hand.

While beyond the scope of this thesis, we provide preliminary results of applying detached PRB to differentiate Monte Carlo PDE solvers [39, 42, 43] in a recent tech report [44]. Monte Carlo PDE methods express the solution to certain PDEs as a Fredholm integral equation, which can be differentiated using path replay to solve inverse problems of PDEs.

5.3 Results

We turn to results and present correctness tests, several applications, and performance evaluations. We implemented our method using an early version of Mitsuba 3 [50] and ran experiments on a NVIDIA TITAN RTX graphics card (24 GB of RAM).

Gradients from detached sampling strategies. In Figure 5.5, we validate the correctness of our method by comparing gradients computed using detached sampling strategies to conventional automatic differentiation and both biased and unbiased RB variants. This figure provides further demonstration that biased RB generates gradients with an incorrect sign. This effect is particularly pronounced when the directional distribution of incident radiance plays a strong role, for instance when optimizing normal or roughness maps.

Gradients from attached sampling strategies. Conversely, Figure 5.6 shows gradients produced by *attached* sampling strategies. The normal map gradients were computed using paths of length 12 using 128 samples per pixel. Specular interreflection along with both reflection and refraction makes it possible to reach each texture-position using various different path configurations. This leads to a relatively high amount of variance compared to the previous case. Radiative backpropagation does not support this type of computation and will return a zero-valued gradient in all three cases.

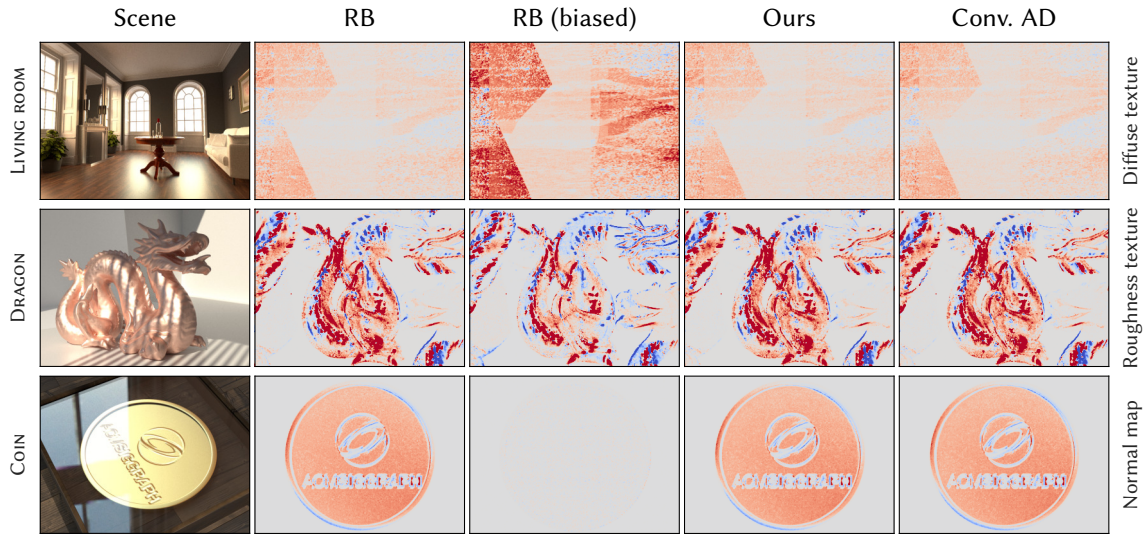


Figure 5.5: Validation of gradients computed using *detached* sampling strategies: we visualize several types of parameter gradients in a somewhat construed validation testcase that involves backpropagating the difference to a blurred version of the primal image. In the first row, we compute the gradient of the diffuse albedo of the wooden floor. Only one dimension is shown in the case of multidimensional parameters like the albedo. The second row shows the gradient of a roughness texture on the dragon. Finally, the last row shows the gradient of the normal map of the embossed coin. Our method and the quadratic time version of radiative backpropagation match reference gradients obtained using conventional automatic differentiation. Biased RB can compute the gradient of a diffuse texture albeit with an incorrect scale, and it produces gradients of the wrong sign for both roughness and normal maps.

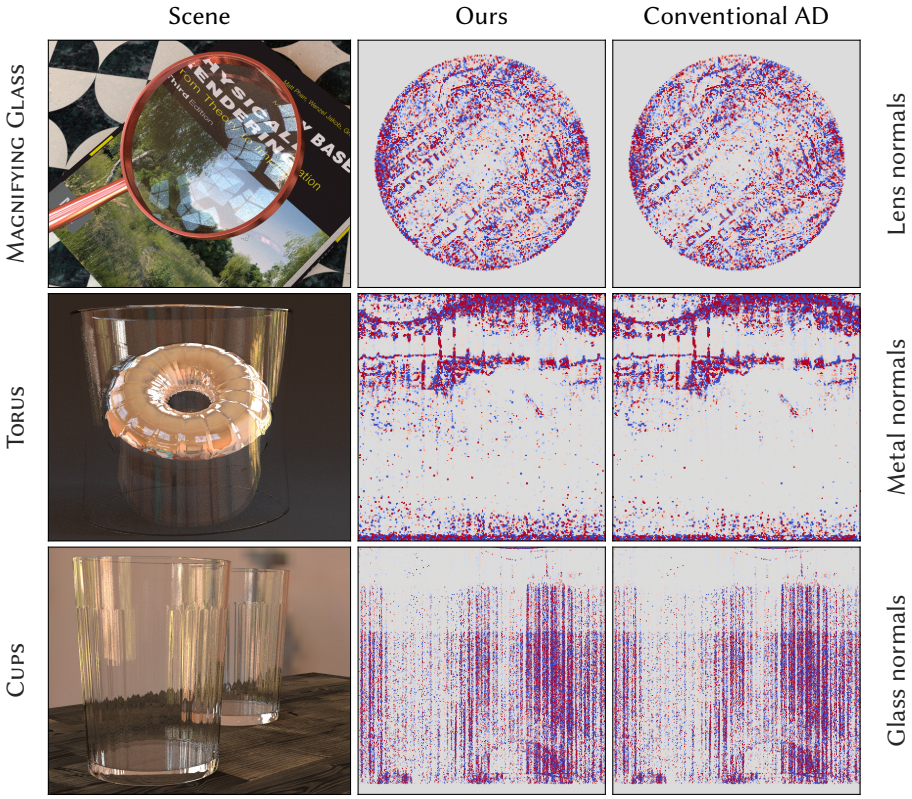
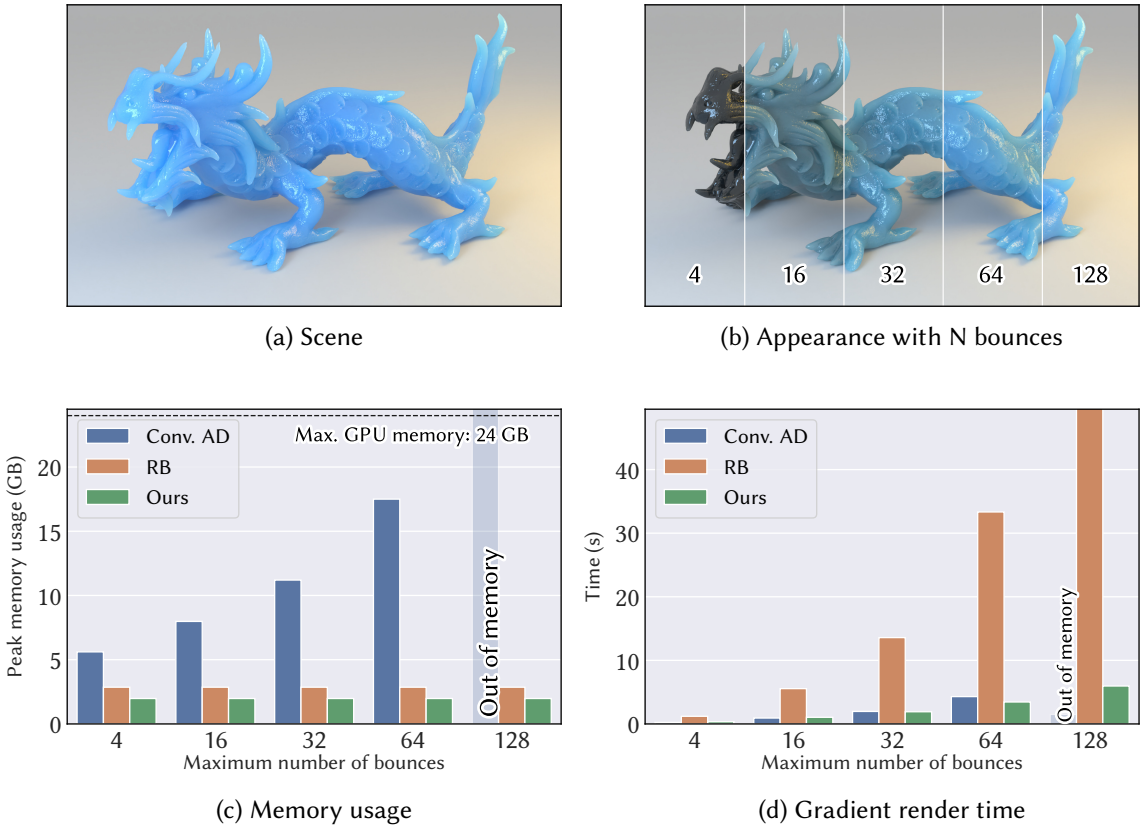


Figure 5.6: Similar to the detached case in Figure 5.5, we now visualize the gradients of *attached* sampling strategies that track differential changes of the path geometry with respect to perturbations of the scene parameters. Parameter optimization of perfectly specular objects requires this approach. The example in the first row shows a magnifying glass with curvature modeled using a normal map. The second scene has normal map on an indirectly observed metallic torus, and the last row uses a normal map to model the profile of the two glass cups.

Subsurface scattering Our method outperforms prior work in scenes that are characterized by light paths with many scattering events. Figure 5.7 showcases the asymptotic behavior of gradient evaluation with respect to the subsurface scattering albedo of a dielectric object with homogeneous internal scattering. The high average path length makes both recursive radiative backpropagation and conventional automatic differentiation approaches unsuitable: the former suffers from quadratic computational cost to recursively estimate the incident illumination, while the latter requires storage of intermediate program state for differentiation that grows linearly with path length and quickly exhausts all available GPU memory.

Heterogeneous volume optimization In Figure 5.8, we use our method to optimize a heterogeneous volume using delta tracking. The large number of null scattering



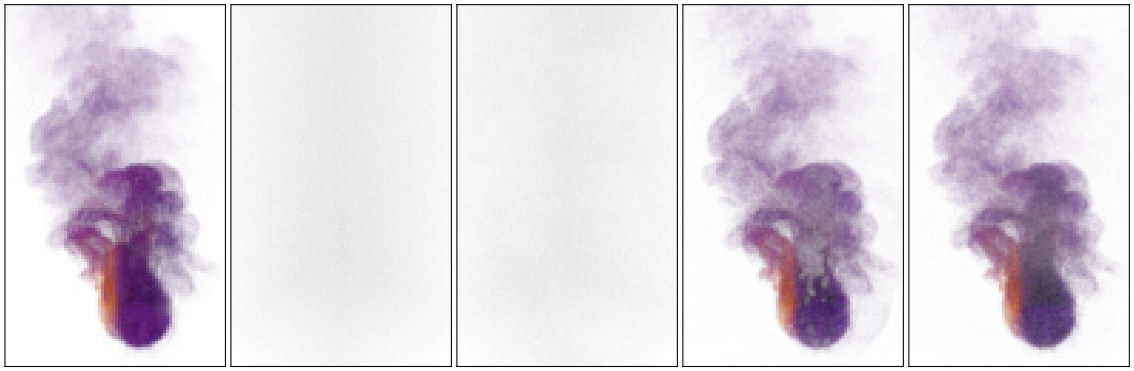
(a) Scene

(b) Appearance with N bounces

(c) Memory usage

(d) Gradient render time

Figure 5.7: **(a)** In this example, we compute gradients with respect to the dragon's subsurface scattering albedo. This represents a typical case where long paths with over a hundred scattering events must be simulated to faithfully capture the material appearance. In **(b)**, we show how the appearance would degrade if we excluded higher order bounces. We then compute the gradient of the albedo using conventional automatic differentiation, unbiased radiative backpropagation (RB), and our new unbiased method. **(c)** Despite rendering the scene only at 640 × 360 pixels and 1 sample per pixel, conventional AD quickly exhausts the available memory of a TITAN RTX GPU. **(d)** Runtime of unbiased RB grows quadratically with path length and becomes prohibitively slow when many scattering events are considered. In contrast, our method performs unbiased estimates using a constant memory footprint and only a linear increase in computation time.



(a) Reference	(b) Conv. AD	(c) RB	(d) RB (biased)	(e) Ours
Time:	45 min	3 min	1 min 47 s	1 min 32 s
Iterations:	0	4	96	64

Figure 5.8: We optimize the density of a heterogeneous volume using delta tracking and multiple methods at roughly equal time. The number of sampled medium interactions makes both conventional automatic differentiation (conv. AD) and unbiased radiative backpropagation (RB) completely infeasible. Conventional AD fails to complete even one iteration within a timespan of 45 minutes. Our method and biased RB are both significantly faster. Our method seems converge slightly more reliably than biased RB.

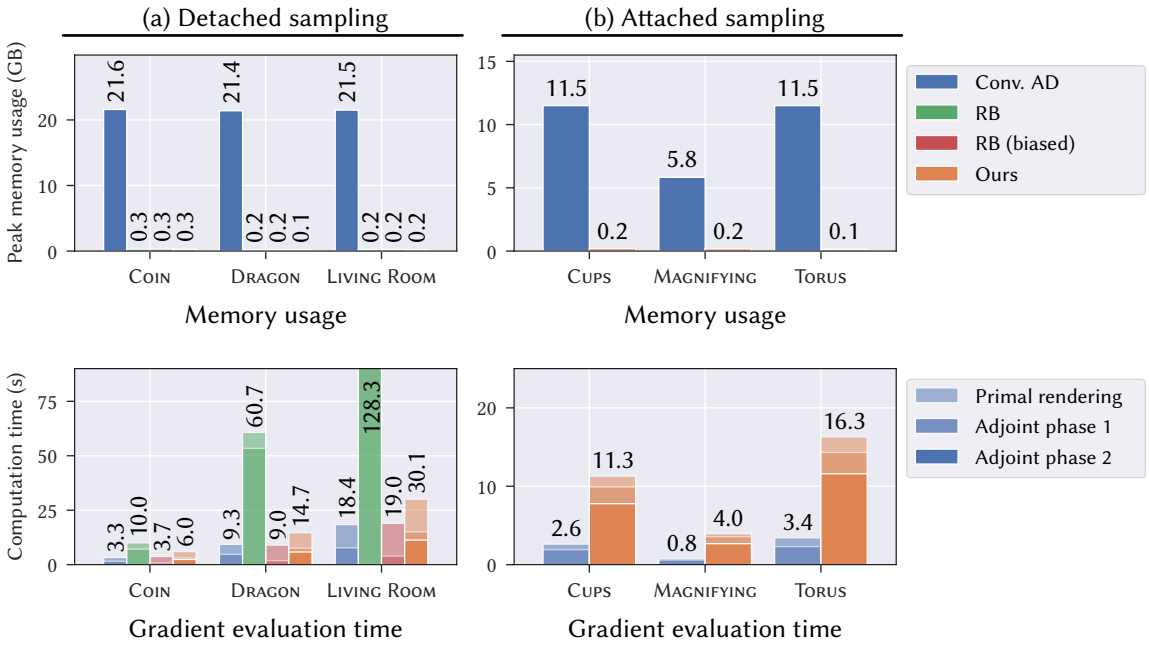


Figure 5.9: We compare the runtime and memory usage of differentiable rendering in multiple example scenes, separating the time spent in the primal and two differentiation-related passes. The primal pass uses a larger number (4×) of samples compared to the adjoint passes, which has a positive effect on gradient variance [149]. In our method, the first adjoint pass refers to the precomputation of temporary information (radiance estimate, ray Jacobians) consumed by the final adjoint pass that accumulates scene parameter derivatives.

(a) Reference (b) Initialization (c) Conventional AD (d) Ours

Figure 5.10: We optimize the normal map of a glass slab so that the refracted view matches a reference image. Neither biased nor unbiased radiative backpropagation can optimize a perfectly specular surface, hence we compare our method to conventional automatic differentiation (at equal iteration count).

events makes any method with a superlinear dependence on path length impractical. Delta tracking maps extremely poorly to wavefront style rendering, so conventional AD, which runs a big wavefront of rays, is unable to complete even a single iteration. Figure 5.1 showcases a similar albedo and density optimization under difficult illumination conditions.

Performance evaluation We evaluate the computation time and memory requirements of both our detached and attached derivatives in Figure 5.9. The results confirm that our method and radiative backpropagation both use a constant amount of memory, which can be substantially lower than memory requirements of conventional AD. While we build on a relatively optimized AD implementation, this approach still uses in excess of 10 GiB of memory to compute a small number of gradient samples in several simple tests. Please see Jakob et al. [50] for a thorough performance evaluation of PRB, including CPU benchmarks. The slower main memory on CPUs results in PRB achieving a 100× speedup over conventional AD.

Optimization using attached sampling. Figure 5.10 shows how attached PRB can be used to optimize the normal map of a glass slab. Our method matches the results achieved using conventional AD. While this is a toy example, similar optimizations are for example required in lens design applications.

5.4 Summary and future work

We presented a new linear-time and constant-memory approach to differentiate the image formation process of physically-based rendering. Our method has a cost that is similar to a biased method presented in previous work, but its unbiased nature makes it more reliable in many types of simulations. Our methods use a constant amount of memory, which we expect to be crucial when optimizing large scenes. We also show that our approach generalizes to the more complex case of attached sampling strategies that track the differential dependence of Monte Carlo importance sampling on scene parameters, enabling differentiation of degenerate BSDFs containing Dirac delta functions. We also expand on previous work targeting volumetric appearance reconstruction and are the first to solve this problem using unbiased delta tracking. The advantages over ray marching match those observed in primal rendering, and we hope that this possibility will inspire inverse rendering to similarly shift towards unbiased volume rendering methods.

We also show that, unlike claimed by prior work, assuming constant incident indirect illumination is generally harmful, as it can result in gradients of the wrong sign. It remains an open question whether it is possible to reduce gradient variance in a meaningful way by using a biased estimator. For such an estimator to be useful, it would need to yield gradients of the correct sign or at least be constructed such that the bias disappears as the optimization converges. However, it is unclear how to construct such biased gradients more efficiently than by post-processing the unbiased estimator (e.g., by using the Adam optimizer [111]). One approach could be to explore the use of biased primal rendering methods (e.g., radiance caching) for differentiable rendering.

Our method is designed for unidirectional path tracing algorithms. More advanced algorithms that need to store full light paths (e.g., bidirectional path tracing or path-space Metropolis light transport) do not benefit from path replay, as then the gradients can be computed directly from stored light paths. However, these algorithms are less commonly used since they become inefficient for long light paths and map less well to GPU rendering than unidirectional path tracing.

Finally, we believe that our method also has the potential to be useful outside of rendering. We recently demonstrated this on the example of Monte Carlo PDE solvers [44]. The ability to differentiate Monte Carlo estimators for Fredholm equations of the second kind opens the possibility to employ differentiable Monte Carlo methods at scale for problems in science and engineering.

6 | Differentiable Signed Distance Function Rendering

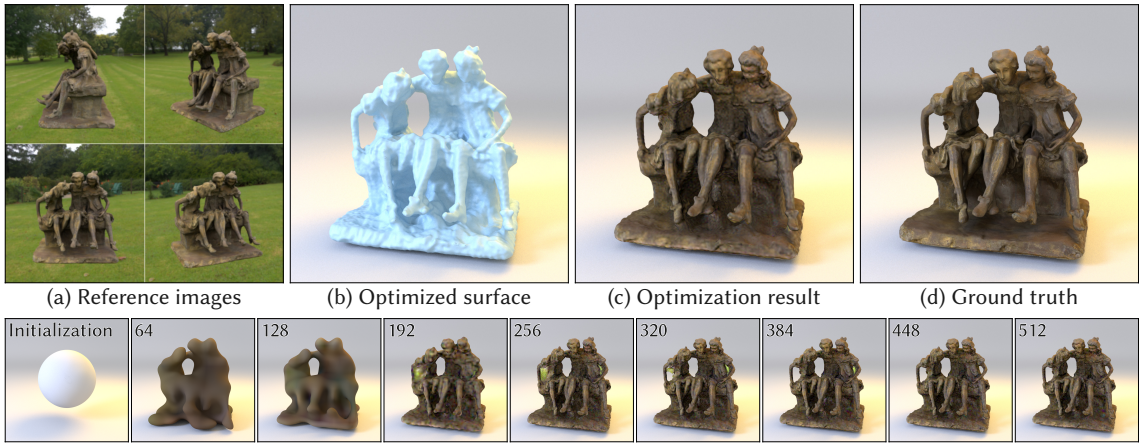


Figure 6.1: Image-based shape and texture reconstruction of a statue given 32 (synthetic) reference images **(a)** and known environment illumination. We use differentiable rendering to jointly optimize a signed distance representation of the geometry and albedo texture by minimizing the L_1 loss between the rendered and the reference images. Our method correctly accounts for discontinuities and we therefore do not require ad-hoc object mask or silhouette supervision. We visualize the reconstructed surface **(b)** and the re-rendered textured object **(c)**. The view and illumination condition in **(b)** and **(c)** are different from the ones used during optimization. In **(d)** we render the ground truth triangle mesh.

The previous chapter introduced a method that can efficiently differentiate long light paths, but ignored the important problem of discontinuities that arise at the silhouette of occluders. Applications such as 3D reconstruction from images require accounting for the instantaneous changes of radiance function on these discontinuities. Otherwise, the derivatives of shape parameters are biased, which impedes the use of gradient descent optimization.

Besides the mathematics of gradient evaluation, a closely related concern is the 3D representation underlying the scene. Certain representations are more amenable for efficient inverse rendering than others. In this chapter, we investigate differentiable rendering of surfaces represented by signed distance functions (SDFs). A signed distance function measures the signed distance to a surface that is defined by its zero-level set. SDFs and general implicit surfaces are a well-established representation, but the recent popularity of inverse rendering has led to a renewed interest in them. We review some recent inverse rendering methods using SDFs in Section 4.6.2. A key advantage of using

SDFs as geometric representation is their ability to easily represent topological changes during the optimization process [195].

While there is a large body of work on using SDFs for inverse rendering, no existing technique can directly differentiate renderings of SDFs with respect to primary, secondary (shadows), or higher-order effects (global interreflections). We generally expect inverse problems to become more useful as their model evaluation and coupled steps like differentiation become increasingly representative of physical reality, hence this is highly desirable. The previously discussed reparameterizations methods [69, 154] can be adapted to SDFs. While originally developed for triangle meshes, these methods are independent of the geometry representation and therefore also apply to SDFs. They expose a bias-versus-computation tradeoff, and to achieve high-quality gradients, require tracing many costly additional rays.

We propose a specialized reparameterization technique for differentiable optimization of SDFs that addresses these drawbacks. Our method augments the sphere tracing technique [204] commonly used for SDF rendering so that it collects a small amount of additional information while stepping through space. We use this information to cheaply instantiate a reparameterization that addresses issues with discontinuous integrals performed by the renderer, so that the remaining calculation can be handled by standard AD techniques. An important difference of our approach compared to prior work is that our parameterization does not need to trace auxiliary rays to sample the surrounding environment in search of occlusion boundaries, since equivalent information is naturally available in the signed distance value of the SDF representation. There are various subtleties that must be considered along the way. We show how careful derivation of gradients and Jacobian determinant leads to an effective and robust method that is both faster and more accurate than prior work.

Finally, we demonstrate the use of our method to reconstruct SDFs of complex objects with gradient-based optimization, eschewing ad-hoc silhouette losses or complex priors. An example result of such an optimization is shown in Figure 6.1. We do not pursue state-of-the-art reconstruction from real-world data in this chapter. Our focus is mainly on efficient derivative evaluation for SDFs. In summary, our contributions are the following:

- We propose a modification of sphere tracing that dynamically constructs reparameterizations enabling accurate differentiable SDF rendering.

- We demonstrate the use of our method for surface reconstruction without the need for complex priors or silhouette loss.

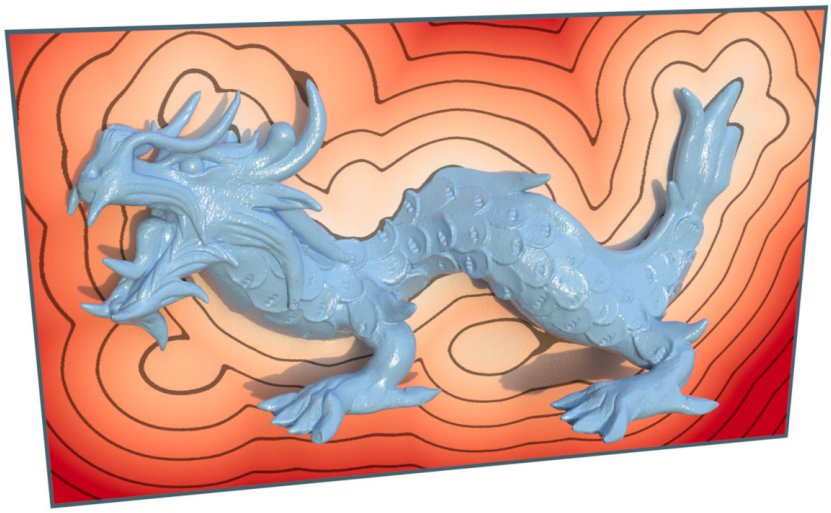


Figure 6.2: A signed distance function is positive outside the object and negative inside. Here we visualize a slice of the SDF and its corresponding isolines. In our implementation, we store the values of the SDF on a regular grid and use B-spline interpolated lookups to ensure smooth normals.

- We provide a rigorous derivation of our reparameterization and the resulting distortion of the integration domain.

Lastly, in the paper we also draw a precise connection between using a reparameterization and applying the divergence theorem [69] for integrals over the unit sphere. This theoretical insight is discussed in Section 4.5.3.

6.1 Method

In the following, we first briefly describe how we store and render SDFs. We then discuss differentiable SDF shading and then finally introduce our reparameterization method to handle visibility discontinuities.

6.1.1 Preliminaries

For a surface $\mathcal{M} \subset \mathbb{R}^3$, the signed distance function $\phi\colon \mathbb{R}^3 \to \mathbb{R}$ measures the distance of a point $\mathbf{x} \in \mathbb{R}^3$ to the surface. We assume the distance to be of negative sign inside and positive outside the surface. Figure 6.2 shows a slice through an example SDF. In the following, we will denote the value of the SDF as $\phi(\mathbf{x}, \boldsymbol{\pi})$, where $\boldsymbol{\pi}$ is the vector of parameters defining the SDF. A key property of SDFs is that they satisfy the eikonal

equation:

$$\|\partial_{\mathbf{x}}\phi(\mathbf{x}, \boldsymbol{\pi})\| = 1, \tag{6.1}$$

i.e., the positional gradient has unit norm. For a point on the surface $\mathcal{M}$, its surface normal equals the positional gradient $\partial_{\mathbf{x}}\phi(\mathbf{x}, \boldsymbol{\pi})$.

SDF representation We store signed distance functions on a voxel grid. With this representation, $\boldsymbol{\pi}$ represents the list of stored values. During rendering, the grid values are interpolated using cubic B-spline basis functions. Sufficiently high-order interpolation is important, since normals are related to the derivative of the SDF. Simple trilinear interpolation would result in discontinuous shading producing an undesirable faceted appearance. We interpolate positional gradient and Hessian using analytic derivatives of the basis functions and leverage the continuity of the SDF's positional gradient to construct a reparameterization. Our method relies on the interpolation smoothing out any potential discontinuities in the positional gradient (including on the SDF's skeleton).

Ray intersection. We use the sphere tracing algorithm [204, 205] to render the SDF representation. Sphere tracing is an iterative procedure that efficiently skips through empty space. In each iteration, the step size is equal to the absolute value of the SDF, i.e., the unsigned distance to the surface, at the current location $\mathbf{x}_i$. This means the algorithm takes a step given the minimum distance to the surface, which ensures that we do not accidentally step over it. Eventually, the ray will either escape into the void, or the evaluated distance will fall below a specified convergence threshold ε, and the ray intersection location is returned. Our method then additionally uses the intermediate SDF evaluations to construct a reparameterization.

Notation As in previous chapters, we simplify the notation by carrying out the derivations use a single parameter π rather than the parameter vector $\boldsymbol{\pi}$. It will further be useful to distinguish between uses of π that are differentiated, or attached to the automatic differentiation graph, and uses that are detached. We will denote detached parameters as π_0.

6.1.2 Shading gradients

Aside from handling discontinuities, differentiable rendering of SDFs also requires the ability to differentiate the evaluation of the surface normal that is later used when evaluating the shape's reflectance model. If a ray intersects the surface defined by the SDF,

the shading normal at the intersection location is given by

$$\mathbf{n}(\pi) = \frac{\partial_{\mathbf{x}}\phi(\mathbf{x}_{t(\pi)}, \pi)}{\left\|\partial_{\mathbf{x}}\phi(\mathbf{x}_{t(\pi)}, \pi)\right\|}, \tag{6.2}$$

where $t(\pi)$ is the intersection distance, $\mathbf{x}_{t(\pi)} \coloneqq \mathbf{x}_\mathrm{o} + t(\pi)\boldsymbol{\omega}$ the intersection location on the surface, with the ray origin $\mathbf{x}_\mathrm{o}$ and the ray direction $\boldsymbol{\omega}$. The normalization is needed, since the grid-interpolated SDF representation cannot guarantee that the eikonal constraint is perfectly satisfied. To differentiate $\mathbf{n}(\pi)$, special care is required, since the intersection distance t depends on π and is the result of sphere tracing, a numerical root finding procedure. Using the inverse function theorem [207, 208], one can show that

$$\partial_\pi t(\pi) = -\frac{\partial_\pi \phi(\mathbf{x}_{t(\pi_0)}, \pi)}{\langle \partial_{\mathbf{x}}\phi(\mathbf{x}_{t(\pi_0)}, \pi_0), \boldsymbol{\omega}\rangle}, \tag{6.3}$$

where $\langle\cdot,\cdot\rangle$ denotes the dot product. Using this expression, we can differentiate the surface normal and correctly account for its dependency on the intersection distance, without the need to track parameter derivatives across sphere tracing iterations. For completeness, we provide the derivation of this expression in Appendix C.1.

6.1.3 Reparameterizing discontinuities

We now turn to our reparameterization for differentiable SDF rendering, decomposing the problem into two steps: first, we define a vector field, whose derivative follows the motion of the SDF surface in 3D. We then show how evaluating this vector field along continuous positions in 3D space enables constructing a reparameterization on the unit sphere, which can correctly handle occlusion and self-occlusion by SDFs. Bangaru et al. [69] followed a similar two-step strategy to define a reparameterization for rendering triangle meshes, but theirs is constructed from a set of auxiliary rays that must be separately traced.

Motion of implicit surfaces. Our eventual goal is to define a reparameterization on the unit sphere. We begin by defining an auxiliary 3D vector field $\mathcal{V}\colon \mathbb{R}^3 \to \mathbb{R}^3$. It is constructed so that the derivative $\partial_\pi \mathcal{V}(\mathbf{x}, \pi) \in \mathbb{R}^3$ matches the infinitesimal surface motion with respect to π when evaluated on the zero level set.

Since tangential motion does not affect discontinuities, we define $\mathcal{V}$ as a scaled multiple of the surface normal, specifically

$$\mathcal{V}(\mathbf{x}, \pi) = -\frac{\partial_{\mathbf{x}}\phi(\mathbf{x}, \pi_0)}{\|\partial_{\mathbf{x}}\phi(\mathbf{x}, \pi_0)\|^2}\phi(\mathbf{x}, \pi). \tag{6.4}$$

Here, only the value of the SDF $\phi(\mathbf{x}, \pi)$ depends on the differentiable parameter π, while the positional gradient and its squared norm are static and hence written using π_0. Similar expressions have been used in prior work on the motion of implicit surfaces [271], neural level sets [272] and differentiable marching cubes [213]. We can verify that the gradient of this vector field matches the surface motion by differentiating with respect to the parameter π:

$$\begin{aligned}
\partial_\pi \mathcal{V}(\mathbf{x}, \pi) &= -\frac{\partial_\mathbf{x}\phi(\mathbf{x}, \pi_0)}{\|\partial_\mathbf{x}\phi(\mathbf{x}, \pi_0)\|^2}\partial_\pi\phi(\mathbf{x}, \pi) \\
&= -\frac{\frac{\partial_\mathbf{x}\phi(\mathbf{x}, \pi_0)}{\|\partial_\mathbf{x}\phi(\mathbf{x}, \pi_0)\|}}{\left\langle \partial_\mathbf{x}\phi(\mathbf{x}, \pi_0), \frac{\partial_\mathbf{x}\phi(\mathbf{x}, \pi_0)}{\|\partial_\mathbf{x}\phi(\mathbf{x}, \pi_0)\|} \right\rangle}\partial_\pi\phi(\mathbf{x}, \pi) \\
&= \frac{-\mathbf{n}}{\langle \partial_\mathbf{x}\phi(\mathbf{x}, \pi_0), \mathbf{n} \rangle}\partial_\pi\phi(\mathbf{x}, \pi) \\
&= \partial_\pi \left[\mathbf{x}_o + t(\pi)\mathbf{n}\right] = \partial_\pi \mathbf{x}(\pi),
\end{aligned} \tag{6.5}$$

where $\mathbf{n} = \frac{\partial_\mathbf{x}\phi(\mathbf{x}, \pi_0)}{\|\partial_\mathbf{x}\phi(\mathbf{x}, \pi_0)\|}$ is the surface normal. The expression on the third line is exactly the motion of a surface point $\mathbf{x}$ that is the result of intersecting a ray in the normal direction $\mathbf{n}$ with the SDF. This follows from plugging $\boldsymbol{\omega} = \mathbf{n}$ into Equation 6.3, which describes the intersection distance gradient. In this case, the ray origin $\mathbf{x}_o$ just needs to be any point along this ray such that the ray intersects the SDF perpendicularly. Note that this is just a construction to prove that $\partial_\pi \mathcal{V}(\mathbf{x}, \pi)$ has the right direction and magnitude, we do not need to actually compute such a ray intersection.

So far, this derivation does not explicitly assume the function ϕ to be an SDF. If ϕ is indeed an SDF, the gradient norm in the denominator in Equation 6.4 equals 1. However, we found that including the normalization by the squared gradient norm makes our method more robust when working with approximate (e.g., grid-interpolated) SDFs. The effect of this is shown in Figure 6.3.

Reparameterization of the unit sphere. We now use this 3D vector field to define a reparameterization in the solid angle domain. Recall the primary requirement of the reparameterization: for a direction $\boldsymbol{\omega}_b$ on a discontinuity on the unit sphere, the reparameterization's gradient $\partial_\pi \mathcal{T}(\boldsymbol{\omega}_b, \pi)$ needs to exactly match the motion of that boundary direction.

The key idea is the following: we define an *evaluation distance function* $t(\boldsymbol{\omega}, \pi_0)$ that, as a boundary is approached, converges to the distance at which the edge of the SDF causing the discontinuity is located in 3D. One important detail is that this distance must itself be continuous in $\boldsymbol{\omega}$, or the requirements on the reparameterization would

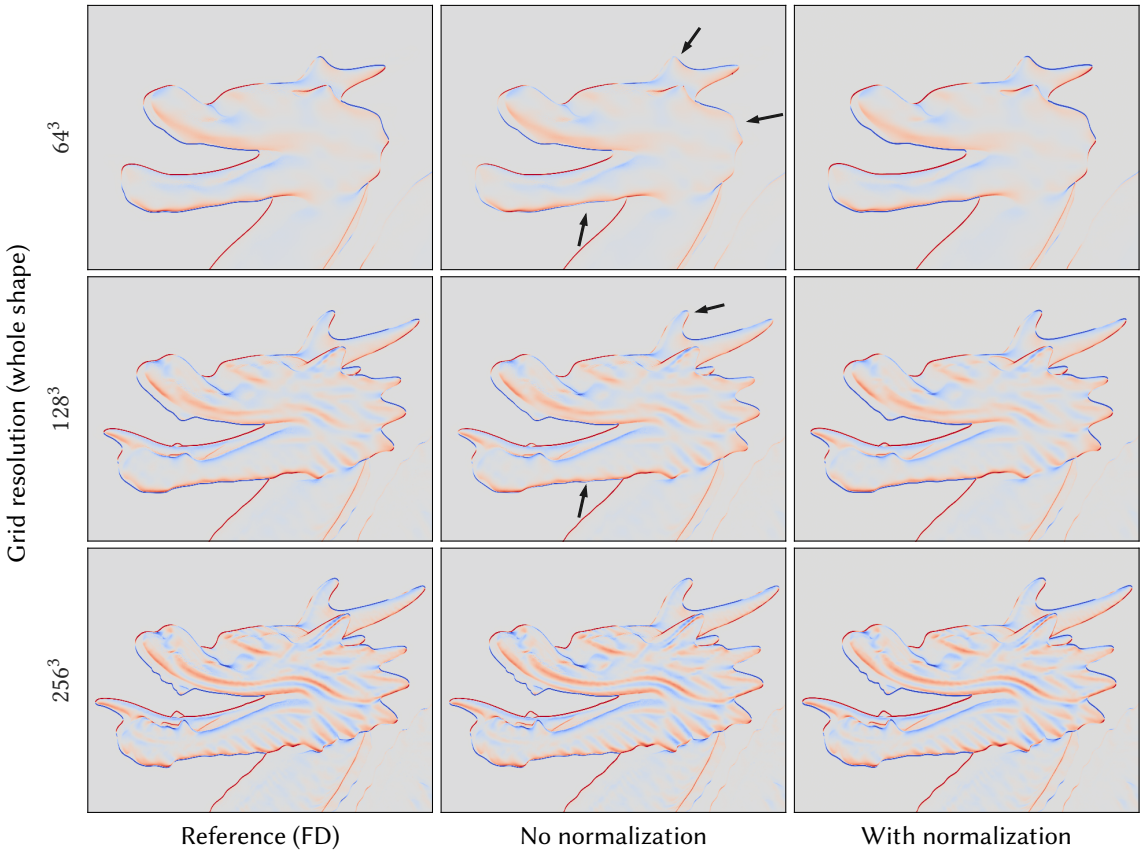


Figure 6.3: We visualize the gradients of a rendered image with respect to a vertical translation of an object represented as an SDF. If we do not include the normalization term described in Equation 6.4, the magnitude of the gradient (**middle** column) does not quite match the reference. Including the normalization improves the accuracy of the estimated silhouette gradients (**right**). As the SDF grid resolution is increased (y-axis), the interpolated SDF more closely matches a true SDF, and the effect of the normalization is less pronounced.

be violated. We compute this distance as a weighted combination of the distances that are encountered during sphere tracing, with weights chosen so that the weighted sum will have the right convergence characteristics as it approaches a boundary. Figure 6.4 illustrates the high-level idea. From an implementation point of view, this corresponds to computing not just the intersection distance during sphere tracing, but one additional distance that we can then use to define our reparameterization. One important property of this distance computation is that it does not depend on the differentiable parameter π. This means that we do not need to compute expensive parameter derivatives $\partial_\pi \phi(\mathbf{x}, \pi)$ within the sphere tracing loop.

For now, we assume this distance t to be given and we will define it precisely later. Given the distance, we construct a reparameterization on the unit sphere by first defining an auxiliary function:

$$\begin{aligned}\bar{\mathcal{T}}(\boldsymbol{\omega}, \pi) &= [\mathbf{x}_t + \mathcal{V}(\mathbf{x}_t, \pi) - \mathcal{V}(\mathbf{x}_t, \pi_0)] - \mathbf{x} \\ &= t\boldsymbol{\omega} + \mathcal{V}(\mathbf{x}_t, \pi) - \mathcal{V}(\mathbf{x}_t, \pi_0),\end{aligned} \tag{6.6}$$

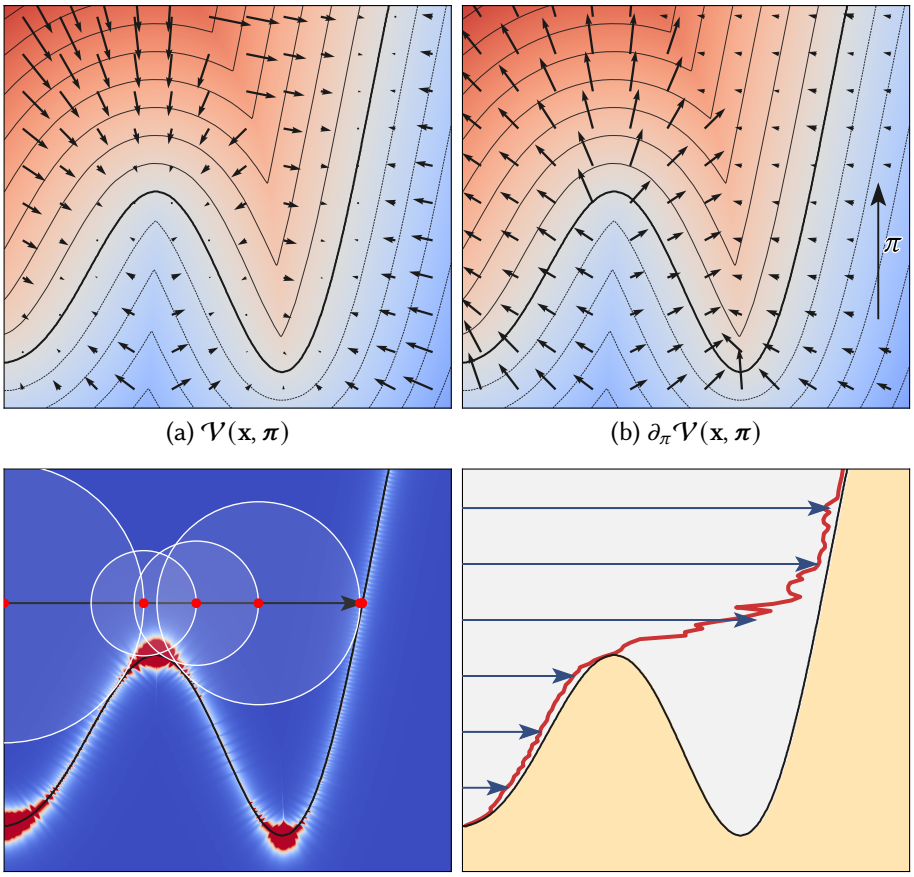
where $\mathbf{x}$ is the ray origin of the ray along $\boldsymbol{\omega}$ and $\mathbf{x}_t \coloneqq \mathbf{x} + t\boldsymbol{\omega}$. The idea here is to take the 3D location $\mathbf{x}_t$ and displace it using our vector field $\mathcal{V}$. We then subtract the ray origin $\mathbf{x}$ to turn this expression into a direction aligned with $\boldsymbol{\omega}$. The subtraction of $\mathcal{V}(\mathbf{x}_t, \pi_0)$ ensures that the map is simply scaling $\boldsymbol{\omega}$ when no derivative is being taken. Since $\bar{\mathcal{T}}(\boldsymbol{\omega}, \pi)$ is not yet a vector of unit length, we normalize to obtain a reparameterization of the unit sphere:

$$\mathcal{T}(\boldsymbol{\omega}, \pi) = \frac{\bar{\mathcal{T}}(\boldsymbol{\omega}, \pi)}{\left\|\bar{\mathcal{T}}(\boldsymbol{\omega}, \pi)\right\|}. \tag{6.7}$$

In the primal domain, this is now an identity map from the unit sphere onto itself. We further need the derivative of this map to follow the motion of the implicit surface over the unit sphere. We can show this by explicitly computing the derivative $\partial_\pi \mathcal{T}$:

$$\partial_\pi \mathcal{T}(\boldsymbol{\omega}, \pi) = \frac{1}{t}\left(\mathbb{I} - \boldsymbol{\omega} \cdot \boldsymbol{\omega}^T\right) \partial_\pi \mathcal{V}(\mathbf{x}_t, \pi), \tag{6.8}$$

where $\mathbb{I}$ is the 3 by 3 identity matrix and $\boldsymbol{\omega} \cdot \boldsymbol{\omega}^T$ is the outer product of $\boldsymbol{\omega}$ with itself (see Appendix C.2 for the expanded derivation). This derivative expression has a simple geometric interpretation: we constructed $\mathcal{V}$ such that its gradient $\partial_\pi \mathcal{V}(\mathbf{x}_t, \pi)$ follows the motion of the surface in 3D space. We then assumed that t was computed such that if $\boldsymbol{\omega}$ approaches a discontinuity, the evaluation distance will converge to the distance to the SDF edge along the current ray. By multiplying the vector field gradient by $\mathbb{I} - \boldsymbol{\omega} \cdot \boldsymbol{\omega}^T$,



(a) $\mathcal{V}(\mathbf{x}, \boldsymbol{\pi})$ (b) $\partial_\pi \mathcal{V}(\mathbf{x}, \boldsymbol{\pi})$

(c) Sphere tracing steps (d) Evaluation distance

Figure 6.4: Our reparameterization builds on a vector field $\mathcal{V}(\mathbf{x}, \pi)$ that is defined everywhere in ambient space **(a)**. Taking the derivative of that vector field with respect to a scalar parameter π will result in a vector field $\partial_\pi \mathcal{V}$ that follows the parameter-dependent motion due to π. In **(b)**, we differentiate the vector field with respect to a global translation in the vertical direction. For a given ray direction, we then compute a weighted combination of positions encountered during sphere tracing to determine where to evaluate the vector field $\partial_\pi \mathcal{V}$. In **(c)** we visualize the intermediate sphere tracing steps and weights that influence the final evaluation point. In **(d)** we draw the computed vector field evaluation locations as a red line as the origin of the incident ray changes. The red line is continuous and coincides with the surface on silhouette edges. In an actual shape optimization, we differentiate $\mathcal{V}(\mathbf{x}, \boldsymbol{\pi})$ with respect to all SDF parameters $\boldsymbol{\pi}$ (i.e. all individual grid values) simultaneously using reverse-mode automatic differentiation.

we project it onto the unit sphere's tangent space. This means any motion in the direction of $\boldsymbol{\omega}$ will be removed. Additionally, the division by t can be interpreted as a form of a geometry term: the motion of the 3D surface over the sphere is decreasing linearly as the evaluation location t moves further away. Overall, this means that our reparameterization attains the right motion on the discontinuities. What remains is to define the evaluation distance t more precisely.

Reparameterization evaluation distance We evaluate our 3D vector field $\mathcal{V}$ at a distance t along the current ray. We obtain a distance function that is both continuous and has the right characteristics on the discontinuities by computing a weighted sum of distances along the ray that are encountered during sphere tracing:

$$t = \frac{1}{\sum_{i=1}^{N} w(i)} \sum_{i=1}^{N} w(i) t_i, \tag{6.9}$$

where t_i are the intermediate distances attained during sphere tracing and w is a weighting function. We define our weighting function as a product of three terms:

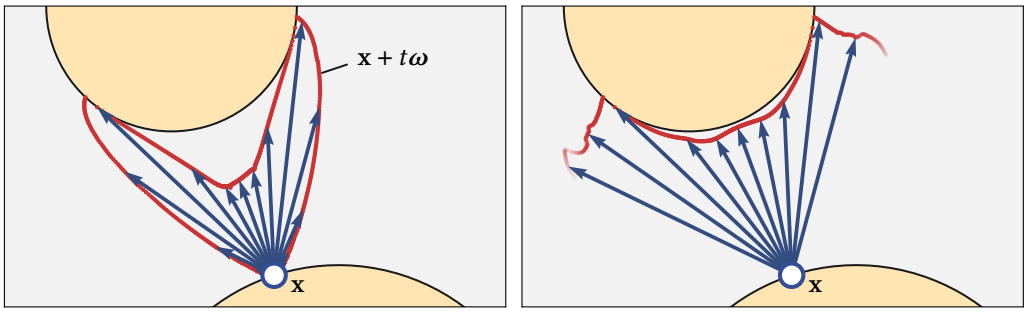
$$w(i) = w_{\text{edge}}(i) w_{\text{dist}}(i) w_{\text{bbox}}(i). \tag{6.10}$$

The first factor detects proximity to discontinuities:

$$w_{\text{edge}}(i) = \left(\varepsilon + |\phi(\mathbf{x}_{t_i}, \pi_0)| + \alpha \left\langle \frac{\partial_{\mathbf{x}} \phi(\mathbf{x}_{t_i}, \pi_0)}{\left\| \partial_{\mathbf{x}} \phi(\mathbf{x}_{t_i}, \pi_0) \right\|}, \boldsymbol{\omega} \right\rangle^2 \right)^{-p}, \tag{6.11}$$

where we use $\varepsilon = 10^{-6}$, $\alpha = 0.1$, $p = 2$ and $\boldsymbol{\omega}$ is the ray direction. This weighting function is designed such that $w \to \infty$ as a surface is approached at a grazing angle (i.e., the sphere tracing reaches an edge causing a discontinuity). The dot product between the ray direction and the (normalized) gradient ensures that the weight only goes to infinity at grazing angles. Without this term, the evaluation distance would coincide with the ray intersection distance and not be continuous. It is crucial for this distance function to be continuous in $\boldsymbol{\omega}$, as otherwise the resulting gradients would be incorrect. This weighting scheme serves a similar purpose as the weights used by Bangaru et al. [69]. In their method, the weights are used for a 2D convolution that has to be evaluated using Monte Carlo integration, whereas our implementation re-uses the locations that are already sampled during sphere tracing.

This first weighting term is not quite sufficient to robustly estimate gradients however. For indirect rays starting on the SDF itself it will likely select an evaluation distance



(a) $\mathbf{x} + t\boldsymbol{\omega}$ without w_{dist} (b) $\mathbf{x} + t\boldsymbol{\omega}$ with w_{dist}

Figure 6.5: We compare the evaluation distance function t with and without the w_{dist} factor. Without this factor, the evaluation distance approaches zero for grazing outgoing ray directions. Including w_{dist} reduces the influence of the surface at the ray origin on the evaluation distance. This also implies that the evaluation distance is undefined for rays that never approach a surface, and we need to make sure our reparameterization continuously goes to zero before reaching this case.

that is very close to the ray origin. This is in particular the case for rays leaving a surface at near grazing angles, causing unnecessary variance without improving gradient estimation. We therefore multiply by a second term:

$$w_{\text{dist}}(i) = \min\left[\sum_{j=1}^{i} \max\left(\frac{\left|\phi(\mathbf{x}_{t_{j-1}}, \pi_0)\right| - \left|\phi(\mathbf{x}_{t_j}, \pi_0)\right|}{\min\left(\beta, \left|\phi(\mathbf{x}_{t_j}, \pi_0)\right|\right)}, 0\right), 1\right], \tag{6.12}$$

where j iterates over sphere tracing steps and $\beta = 0.05$. While the formula appears complicated, the intuition is simple: we only start considering sphere tracing locations as a surface is *approached*. This present formulation ensures that this is done in a way that remains continuous, since we want the final distance t to vary continuously as the ray direction changes. Moreover, the distance term in the denominator ensures that the weight is guaranteed to reach 1 as the surface is reached. There is little extra cost in adding this weighting term, as it just re-uses evaluations of the SDF that are already computed during sphere tracing. The effect of this weighting term is illustrated on a 2D example in Figure 6.5.

Lastly, we need to ensure continuity of the evaluation distance even as the number N of sphere tracing steps changes. Such a change can either be caused by the algorithm traversing beyond the bounding box, or by a different number of iterations being needed to converge to the surface intersection. The first case is handled by the bounding box weight term, which attenuates the influence of positions as the bounding box of the SDF is approached:

$$w_{\text{bbox}}(i) = \min\left(\text{dist}(\mathbf{x}_{t_i}, \text{bbox})/0.01, 1\right). \tag{6.13}$$

To deal with the second case, we additionally multiply each weight by the mean of the previous and current sphere tracing step length. This works because the iteration count increases or decreases due to samples either being right below or newly above the distance threshold used to terminate a ray intersection. In these cases, the distance between subsequent samples will approach zero. Conceptually, this weighting scheme can be interpreted as replacing the summation in Equation 6.9 by integration and applying a trapezoidal quadrature to both integrals:

$$t = \frac{1}{\int w(t)\,\mathrm{d}t} \int w(t) t\,\mathrm{d}t. \tag{6.14}$$

A key insight here is that we do not need these integrals to be evaluated in an unbiased way. We just need the resulting function $t(\boldsymbol{\omega}, \pi_0)$ to satisfy the necessary conditions and the resulting gradient estimator will be unbiased.

While our algorithm here is designed to work with SDFs, the high-level idea could potentially also be applied to more general implicit functions, where, instead of sphere tracing, ray marching and bisection are used to compute ray intersections. Such a generalization would still require the implicit function to be continuous and exhibit sufficient global structure to enable a suitable evaluation distance computation (e.g., an indicator function would not work).

Weighted reparameterization We can further reduce the variance of the gradient estimator by attenuating the effect of the reparameterization for directions that are further away from an actual discontinuity. In general, we can multiply our 3D vector field with any continuous weighting function $w_{\mathcal{V}}(\mathbf{x}, \pi_0)$, as long as its value approaches 1 as it approaches a discontinuity (with respect to the ray direction $\boldsymbol{\omega}$). We write a weighted version of our vector field as:

$$\bar{\mathcal{V}}(\mathbf{x}, \pi) = w_{\mathcal{V}}(\mathbf{x}, \pi_0)\mathcal{V}(\mathbf{x}, \pi). \tag{6.15}$$

The weighting function itself depends on the detached scene parameter, which ensures that the scene parameter gradient remains unchanged. This approach could even be generalized to a weighted sum of vector fields, similar to multiple importance sampling [60]. The weights would only need to sum to 1 on an actual discontinuity. A similar idea has been used by Zeltner et al. [68] to handle discontinuities that occur when computing BSDF derivatives through the BSDF sampling routine. We use the following weighting function to attenuate the vector field $\mathcal{V}(\mathbf{x}, \pi)$:

$$w_{\mathcal{V}}(\mathbf{x}, \pi_0) = \max\left(0, 1 - \frac{\phi(\mathbf{x}_t, \pi_0)}{t \cdot \varepsilon(\mathbf{x})}\right) \tag{6.16}$$

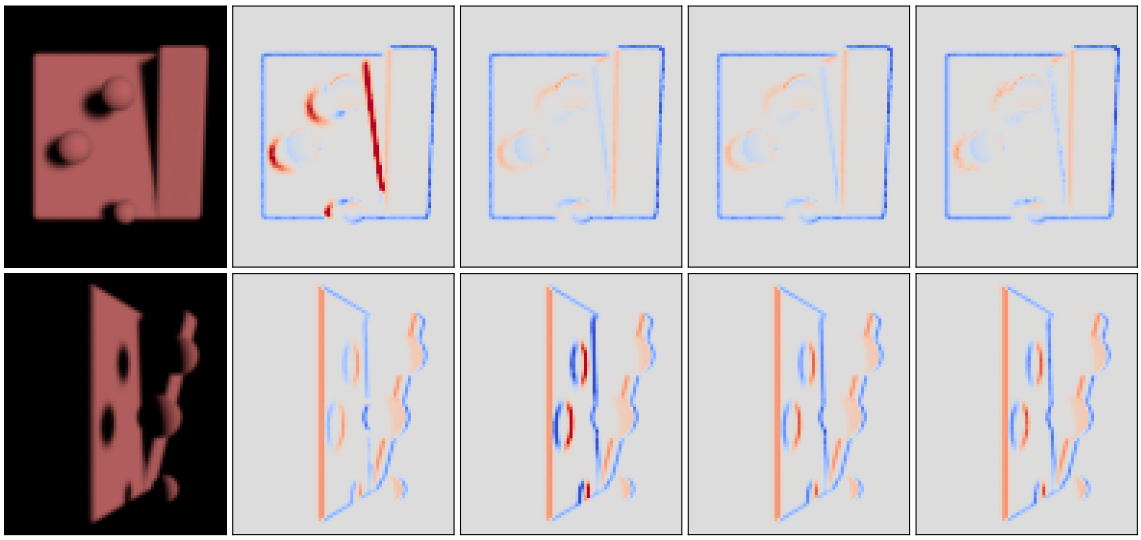
where $\varepsilon(\mathbf{x})$ is the minimum of 0.01 and the distance of $\mathbf{x}$ to the SDF's bounding box. This weighting effectively restricts the reparameterization to only have an impact as $\mathbf{x}$ is approaching a surface and therefore possibly a discontinuity. Additionally, by considering the bounding box it ensures that our reparameterization continuously drops off to zero as the evaluation location approaches the border of the SDF volume. Finally, we also multiply this weight by the sum of sphere tracing weights (clamped to at most 1) to handle the degenerate case of the evaluation location being undefined.

Area element On top of evaluating the reparameterization $\mathcal{T}$ itself, we need to evaluate the area element as defined in Equation 4.39. To do so, we first compute the Jacobian matrix $\partial_{\boldsymbol{\omega}}\mathcal{T}$ analytically. We then simply evaluate the trace of this Jacobian to account for the area change, as it is equivalent to the cross product formulation under differentiation (see Appendix A). The trace requires slightly less computation than explicitly computing the cross product of the transformed tangent vectors.

Since our reparameterization depends on the distance $t(\boldsymbol{\omega}, \pi_0)$, we need to evaluate the derivative $\partial_{\boldsymbol{\omega}} t(\boldsymbol{\omega}, \pi_0) \in \mathbb{R}^3$ to compute the Jacobian. We analytically compute this term during sphere tracing, and return it alongside the distance t and, if applicable, the intersection distance. None of these terms require tracking a differentiable dependency on π through the sphere tracing loop, hence there is no need to build an AD graph over it, which would be an expensive operation. The complete derivation of the Jacobian is laborious and done in Appendix C.3.

Variance reduction Reparameterizing the integral can cause undesirable gradient variance in regions of the image without discontinuities. Based on the observation that the majority of that noise is caused by the differentiable evaluation of the pixel filter, prior work suggested using antithetic sampling and control variates [69, 154]. We find that steps like antithetic sampling and control variates add a significant amount of implementation complexity, and that a similar variance reduction can be achieved by easier means. Renderers usually divide the accumulated radiance in each pixel by the sum of accumulated pixel filter weights to reduce variance [4, Section 13.9]. Interestingly, we found that by tracking derivatives through this normalization step, most of the noise caused by the differentiable pixel filter evaluation can be eliminated.

Nested reparameterization When building a differentiable rendering algorithm, we have to correctly handle the subtleties due to *nested* application of our reparameterization. Related to that, the surface point that results from a ray intersection might be



Scene | (a) Detached $\mathbf{x}$ | (b) Decoupled reparam. | (c) Correct nesting | (d) Reference (FD)

Figure 6.6: This figure shows the subtleties of nesting reparameterizations. We render the same object from two different viewpoints and compute gradients with respect to a translation. If we fully detach the origin $\mathbf{x}$ of the shadow ray **(a)** the result is wrong. If we do not track the effect of the reparameterization of the primary rays, we also get wrong results **(b)**. Only if we carefully account for these effects we get output **(c)** that matches the reference **(d)**.

parameter dependent due to the motion of the surface itself, as explained in Section 6.1.2. Both these factors will have the consequence that the ray origin $\mathbf{x}$, and hence $\mathbf{x}_t$, can differentiably depend on the SDF parameter π. This is relevant for example when applying our reparameterization to a shadow ray. In the derivations so far, we have not considered this potential dependency. Completely ignoring the parameter dependence of $\mathbf{x}_t$ when handling the shadow ray will produce wrong results. If we ignore the dependency on the reparameterization of the primary ray when reparameterizing the shadow ray, the gradient is also not correct. Only if we track the dependency on the primary ray's reparameterization all the way through the secondary reparameterization, we get correct gradients, as shown in Figure 6.6. More precisely, we need to make sure to evaluate $\phi(\mathbf{x}_t, \pi)$ in Equation 6.4 using the parameter-dependent $\mathbf{x}_t(\pi) = \mathbf{x}(\pi) + t\boldsymbol{\omega}$. All other terms of the reparameterization can remain detached as before. This then ensures that our vector field produces the right relative motion between occluder and ray origin. A more detailed discussion and derivation is provided in Appendix C.4.

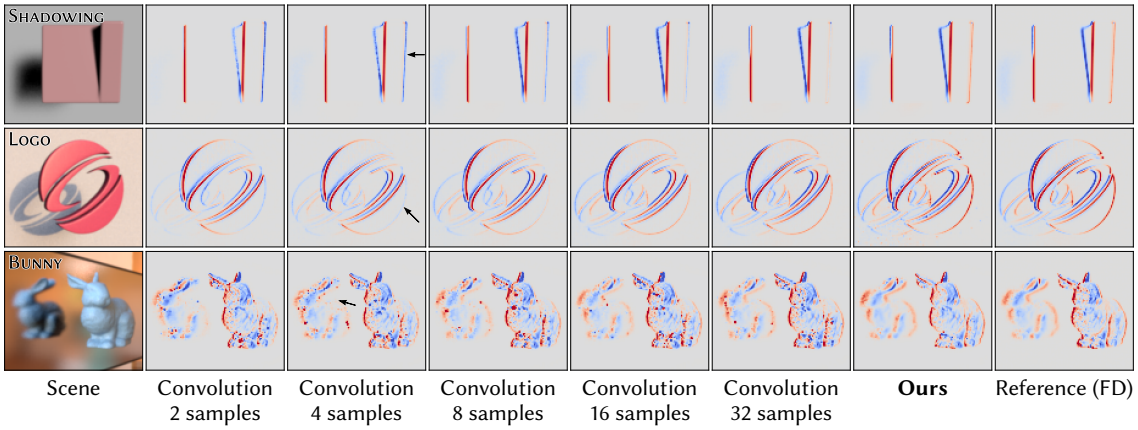


Figure 6.7: We compare the gradients obtained using our method to the ground truth and an SDF version of the convolution method by Bangaru et al. [69]. The gradient images are computed by using forward-mode differentiation with respect to a translation of the entire object. For the convolution method, we show the results using varying numbers of auxiliary rays estimating the convolution integral. Increasing the number of rays improves the accuracy of the gradient estimate, at the cost of increased computation time. All gradient images are rendered using 1024 samples per pixel. The Shadowing and Logo scene are using direct illumination, and the Bunny scene is rendered with one bounce of indirect illumination.

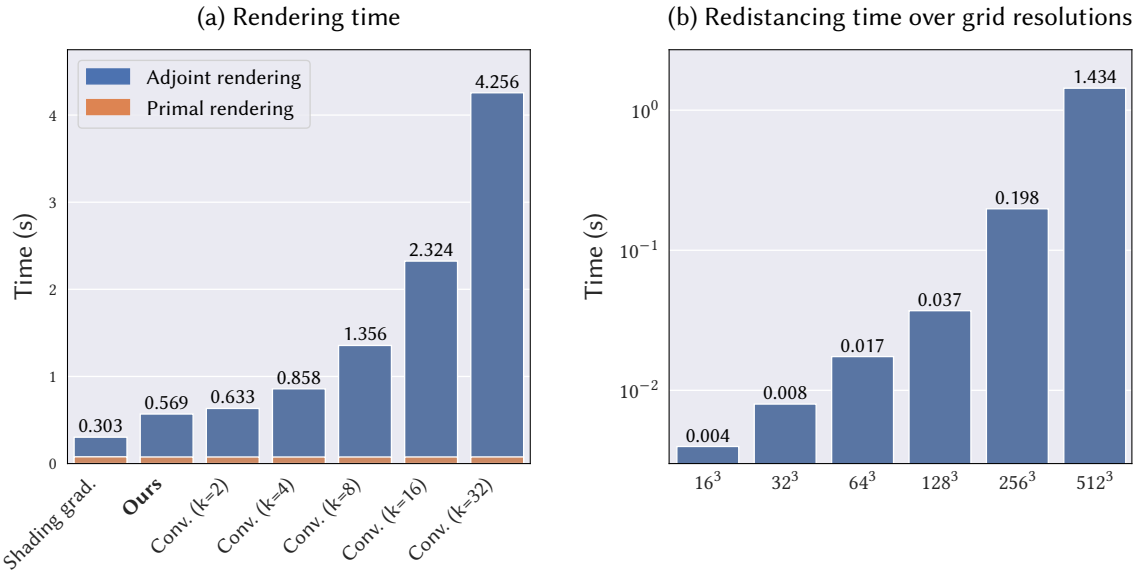


Figure 6.8: We measure the rendering time **(a)** and the SDF redistancing time **(b)**. The rendering time plot reports both the time to render the primal image and the time required to compute SDF gradients in reverse-mode. The first column, *Shading grad.*, only computes shading gradients and ignores discontinuities. For the convolution method, k denotes the number of convolution samples. For all techniques we render 256^2 pixels at 256 primal and 64 adjoint SPP.

6.2 Shape optimization

The reparameterization introduced in the previous section enables computing accurate gradients for renderings of signed distance functions. An important use of such gradients is 3D shape reconstruction given a set of observed views. In this section, we will describe the overall pipeline and settings we use to produce the optimization examples in this chapter. Our goal is to reconstruct a shape by solving

$$\boldsymbol{\pi}^* = \arg\min_{\boldsymbol{\pi}} \sum_{i=1}^{N} \ell\left(I^i(\boldsymbol{\pi}), I^i_{ref}\right), \tag{6.17}$$

where N is the number of views and $\boldsymbol{\pi}$ contains both the SDF parameters, as well as any optimized parameters used in its BSDF. The loss function ℓ measures the differences between images. In the following, we will describe the specific parameters and heuristics that we use to make this optimization practical.

Loss function We use an L_1 loss on linear RGB pixel values for all optimizations. We evaluate it both at the original image resolution, as well as on a 3-level pyramid of downsampled reference and rendered images. This helps to increase the spatial support of the loss function. If we were to only evaluate the L_1 loss at the original resolution, the optimization might more easily get stuck in a local minimum representing a low-quality solution.

Optimizing SDFs During optimization, the differentiable renderer backpropagates gradients to the SDF grid values. Even after a single iteration of gradient descent, the values stored in the grid might not represent a valid SDF anymore [273]. In particular, the SDF will violate the eikonal constraint and in general $\|\partial_{\mathbf{x}}\phi(\mathbf{x}, \boldsymbol{\pi})\| \neq 1$. A common approach to reduce the deviation from a true SDF is adding an eikonal regularization term to the optimization, that penalizes deviations of the gradient norm from 1 [274]. This is particularly useful when the SDF is not stored explicitly, but rather is the output of a neural network. The disadvantage of this regularization approach is that it does not yield an exact SDF and introduces another hyperparameter in the form of a regularization weight. Since we are directly storing the SDF values on a grid, we found it more convenient to explicitly *redistance* the SDF after every iteration of the optimization. The high-level idea is to reconstruct the distance function values by marching outwards from the current zero-level set [275, 276, 277]. In practice, we use a CUDA implementation of the parallel fast sweeping method [278, 279] and redistance the SDF after every iteration

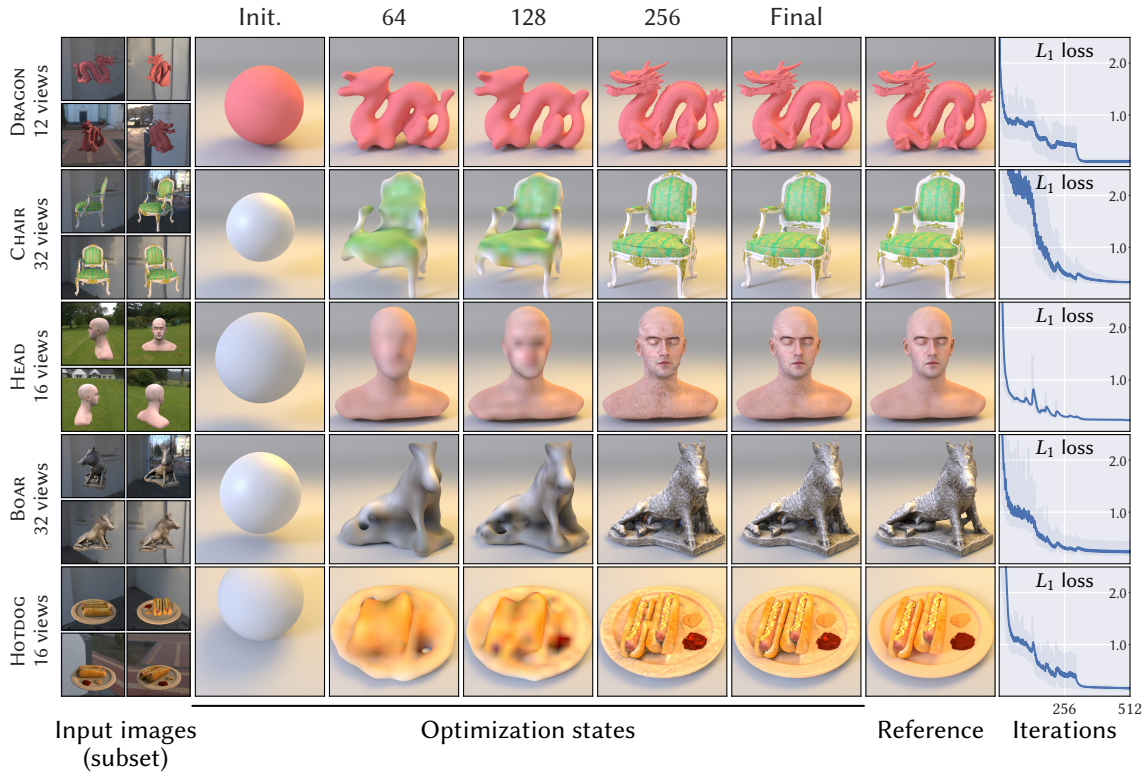


Figure 6.9: Results using our method on a few challenging example objects. All these examples use 512 gradient descent iterations (using batches of 6 views). For the Dragon scene, the BSDF parameters are assumed to be fully known. For the other scenes we optimize albedo textures, and for Chair and Boar additionally surface roughness.

of the optimization. To further improve performance, it could be interesting to use a sparse SDF representation in conjunction with a sparse version of fast sweeping [280]. Another approach would be to use *velocity extension* [281], which infers SDF-compatible grid value updates from the speed of the surface itself. We experimented with simple versions of such an idea, but in the end found the redistancing to work well enough. Alternatively, one could investigate updating the SDF using the adjoint state method, similar to work on art-directable fluid simulations [138] and travel-time tomography [282].

Regularization On top of the image-based loss, we found it beneficial to slightly regularize the SDF using a Laplacian regularization. While the optimization itself is robust, a small amount of regularization can reduce the noise on unobserved regions or due to variance in the gradients and the rendered images. This primarily improves the surface appearance when re-rendering under new illumination conditions. We use a simple discrete Laplacian kernel, which penalizes differences between a voxel and its directly adjacent neighbors. We use a regularization weight of 10^{-5}, which is small enough to

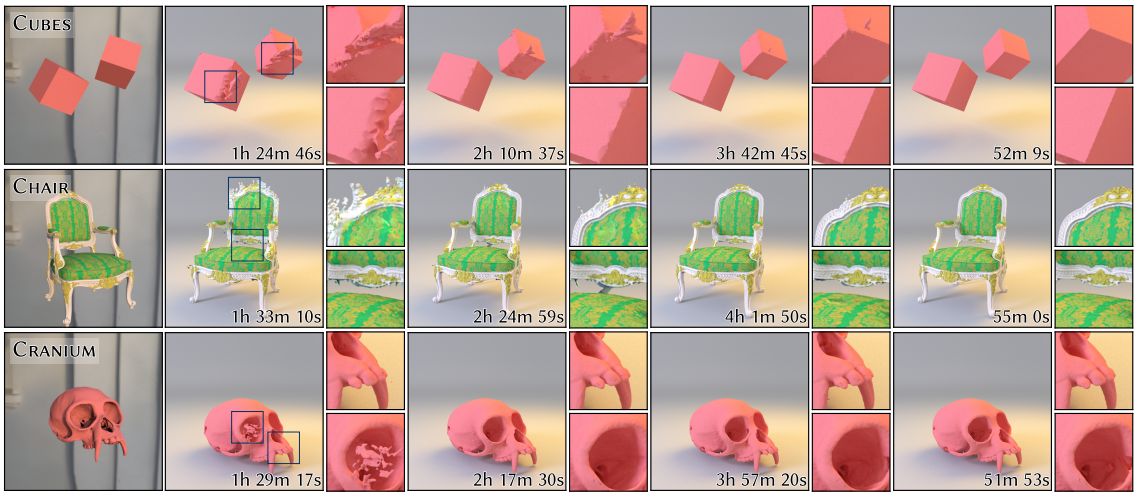


Figure 6.10: We compare the reconstruction results using our reparameterization and the convolution method [69] at equal iteration count. Increasing the number of samples to estimate the convolution integral improves the quality of results, but at the cost of drastically increasing the total time for the optimization. Our method both results in the most accurate reconstruction and the shortest runtime.

not oversmooth the surface.

Multiscale optimization It is further advantageous to initially optimize the SDF at a lower resolution than the target resolution. We start optimizing at a resolution of 16^3 voxels and then double the resolution several times during the optimization until the desired target resolution is reached.

Texture optimization We also optionally optimize albedo and roughness parameters, stored on trilinearly interpolated grids. Optimization of these quantities is straightforward and only requires clamping to valid parameter ranges. When optimizing roughness, we use the Disney BSDF [85], but turn off all lobes except for diffuse and specular lobe.

6.3 Results

Implementation. In the following, we evaluate the correctness, performance and optimization results using our method. Our implementation is based on Mitsuba 3 [50] and benchmarks have been conducted on a NVIDIA TITAN RTX graphics card. We use reverse-mode AD to propagate derivatives to the SDF parameters. We implemented the

majority of our pipeline using Mitsuba 3's Python API. Interpreting our reparameterization as a differentiable ray tracing operation, we absorb its logic into the ray intersection function, which can then be used as follows inside an integrator:

```
si, ray.d, area_element = ray_intersect(ray)
# Evaluate terms in the integrand using the now reparameterized "ray.d"
throughput *= bsdf.eval(si, si.to_local(ray.d))
# ... and multiply by area elementsb
throughput *= area_element
```

The ray intersection routine returns a surface interaction record, the reparameterized ray direction and the area element. This abstraction allows to cleanly implement different integrators leveraging the same reparameterization logic.

We use the same hyperparameters for all our results and did not find our method to be particularly sensitive to the various parameters used to define the weights in Section 6.1. Unless stated otherwise, our optimizations use a direct illumination integrator with emitter sampling. All optimizations use the Adam optimizer [111] with a learning rate that is proportional to the current grid resolution. We optimize SDFs up to a resolution of 256^3 voxels by differentiably rendering 512^2 pixel images. Initial optimization iterations use both a lower resolution SDF and lower rendered image resolutions to improve performance. Similar to previous work, we decorrelate the estimation of the primal image and the gradients [148]. For our optimizations, we use 256 primal and 64 adjoint samples per pixel. These sample counts are chosen conservatively to reduce the impact of Monte Carlo noise on the optimization results. For the highest performance optimization, adaptively sampling both temporal and spatial dimensions could be effective at reducing the overall optimization time.

Gradient validation. We validate the gradients computed using our reparameterization in Figure 6.7. We use forward-mode differentiation to obtain gradients of the pixel values with respect to a translation of the SDF. The reference gradients are obtained using finite differences (with $h = 10^{-3}$). For the scene using indirect illumination, we use a reparameterized version of path replay backpropagation [45], following the work by Zeltner et al. [68]. We also implemented an SDF version of the convolution method by Bangaru et al. [69]. This is the only prior algorithm that can be used to differentiate physically-based renderings of implicit shapes without explicit meshing. We use their convolution kernel and apply it to our 3D vector field defined in Equation 6.4. We set the concentration parameter κ of the spherical von Mises-Fisher distribution to 10^5 for all our experiments. This worked better than the default of 10^4 suggested in the original paper. Higher values (e.g., $\kappa = 10^6$) then again seemed to reduce the quality, in particular

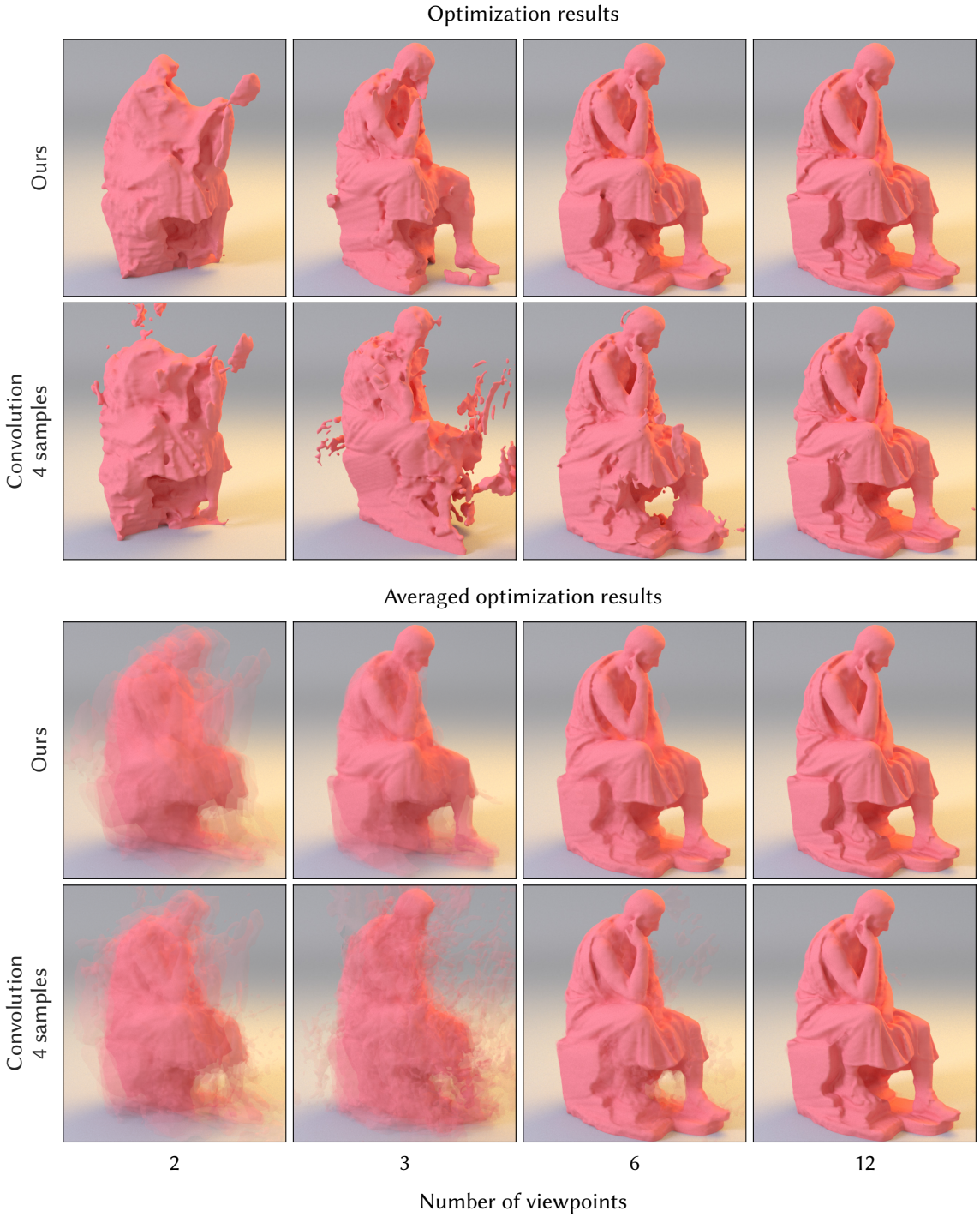


Figure 6.11: The robustness of the shape reconstruction depends on the number of input views. **Top row:** we show an optimization result using a varying number of reference views for both our method and the convolution method. **Bottom row:** we average renderings of optimized shapes for 8 different sets of reference viewpoints. We set up evenly spaced virtual cameras around the object and then rotated them around the vertical axis of the object by varying amounts. The haze around the object in the averaged image is caused by variance in the reconstructed geometry.

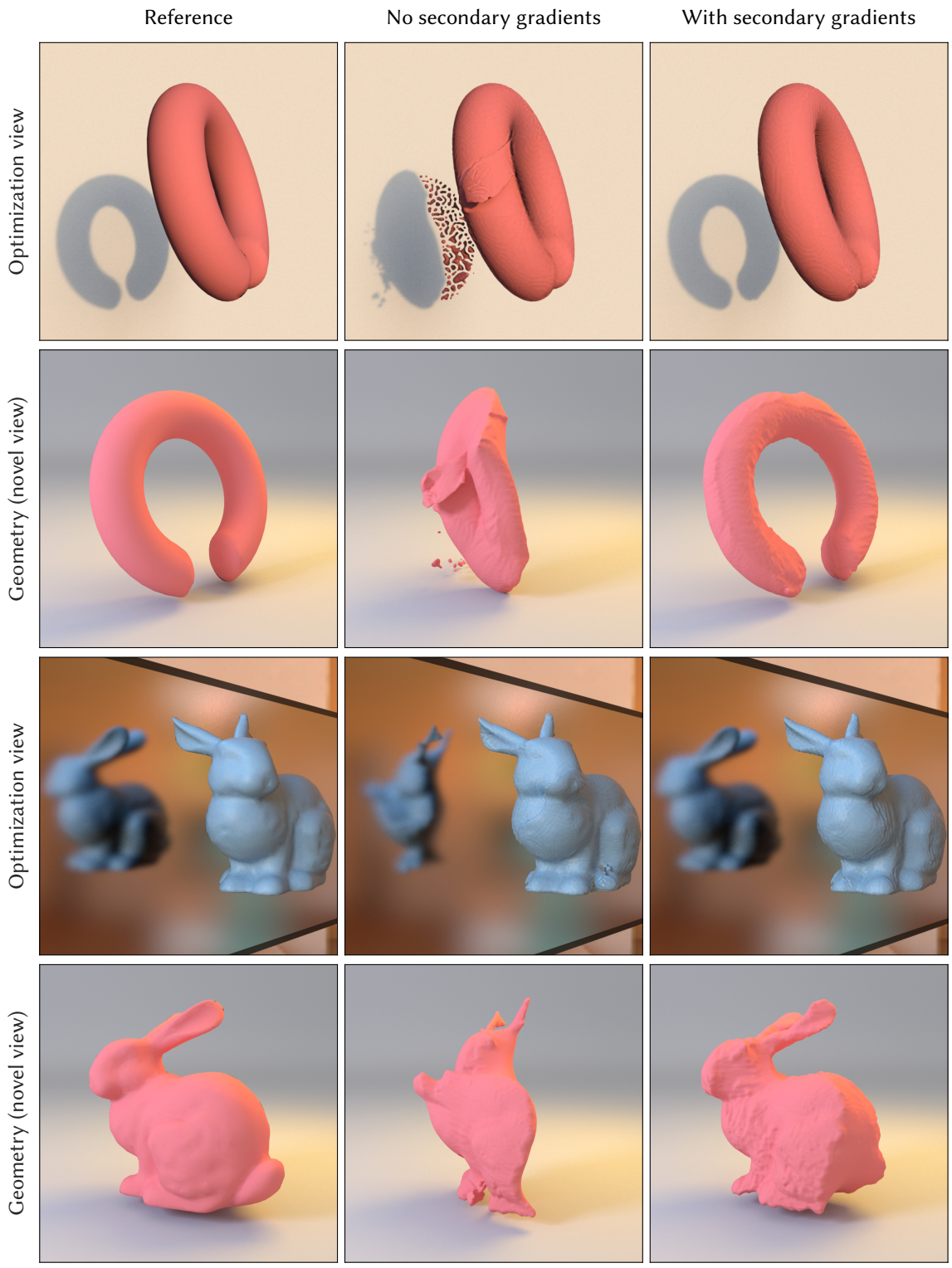


Figure 6.12: We compare single view reconstructions both without using any secondary gradients (middle column) and using secondary gradients (right column). Accounting for indirect effects improves the reconstruction quality when the number of observations is limited.

of shadow gradients. In Figure 6.7, we estimate the convolution integral using varying numbers of auxiliary rays. We can see that using 4 rays already provides some edge gradients, but we oftentimes need up to 16 to get a more accurate estimate with the correct sign. This appears to be consistent with the observations made for triangle meshes in the original paper. Our method on the other hand produces gradients that closely match the finite difference reference.

Benchmarks. We benchmark the different gradient computation algorithms in Figure 6.8. We show both the time required to render the primal image and the time used to estimate gradients. We use our direct illumination integrator for these benchmarks. Compared to only considering shading gradients, our method is around 1.9× slower, since it evaluates additional terms during sphere tracing and also needs to compute the area element. It is faster than the convolution method, in particular as the number of auxiliary rays increases. The primal rendering time remains constant across all methods, since we are careful to only reparameterize when computing gradients. We also benchmark the SDF redistancing implementation to show the potential overhead caused by the redistancing. We only need to redistance once per iteration, but each iteration of gradient descent might require rendering multiple images. As the SDF resolution increases, we also need to increase the resolution of the rendered images to effectively use the additional surface resolution. Overall, we found the overhead due to redistancing negligible compared to the differentiable rendering itself.

Optimization results. In Figure 6.9 we show different reconstructions obtained using our reparameterization and shape optimization scheme. We always initialize our SDF to a sphere of a constant color. For all scenes, we re-render the optimization result from a novel view using a new illumination condition that was not part of the optimization. Our method can reconstruct these various objects without the use of a silhouette constraint. One common issue in shape optimizations are pieces of geometry that are detached from the main shape and are not removed by the optimization. We did not observe such issues with our method. The combination of multiscale optimization and noise in the optimization process even allows to overcome some of the local minima inherent in this setting. For example, this allows to correctly reconstruct the complex topology of the Chair scene. However, a purely surface-based optimization routine can also get stuck in local minima, where the loss will not provide any useful gradients to improve further. In the combination with optimizing albedo textures this can cause some holes of shapes to be erroneously filled in, e.g., between the legs of the Boar statue. This non-convexity

is a known difficulty of the optimization problem and addressing it is beyond the scope of this chapter.

Comparison to the convolution method. While we have already seen that our reparameterization produces more accurate gradients than the SDF version of the method by Bangaru et al. [69], we can also validate that this actually yields better optimization results. In Figure 6.10 we show optimizations both using our and the convolution method using varying numbers of auxiliary rays. Overall, we can see that using a low number of auxiliary rays often results in artifacts in the reconstructed geometry. The inaccurate edge gradients in particular seem to cause problems in scenes with sharp edges, as illustrated in the Cubes scene. As the number of auxiliary rays increases, the convolution method manages to produce better results, at the cost of an increase in overall runtime. Similar issues would likely occur if one were to apply the convolution method to a finely tesselated triangle mesh obtained, e.g., using MeshSDF [213].

Influcence of the number of viewpoints. The quality of the optimization results depends on the number of reference images of the given scene. Studying the behavior of this effect can provide additional insight in the stability of gradient estimation and optimization methods. In Figure 6.11, we compare reconstructions obtained both using our method and the convolution method using 4 auxiliary rays. We show both results as the number of views increases and also show images blending results of 8 separate runs over different configurations to illustrate the stability of the optimization. These results further confirm that our method is more robust at a lower number of reference views than the convolution method. Increasing the number of auxiliary rays would again increase its runtime. At a higher number of viewpoints the optimization is more constrained and the results become more similar.

Benefits of diffentiating secondary effects. One key advantage of our method is that it can differentiate secondary effects such as shadows and indirect illumination. Figure 6.12 showcases two example optimizations where accounting for those gradients improves the reconstructed geometry. Both examples use only a single reference view and in both cases differentiating secondary effects helps to reduce ambiguities by leveraging additional shape cues in the form of shadows and reflections. In the first example, the optimization that ignores secondary gradients converges to an undesirable local minimum, where part of the shadow on the background plane ends up being approximated by many small disconnected components. In the second example, we optimize

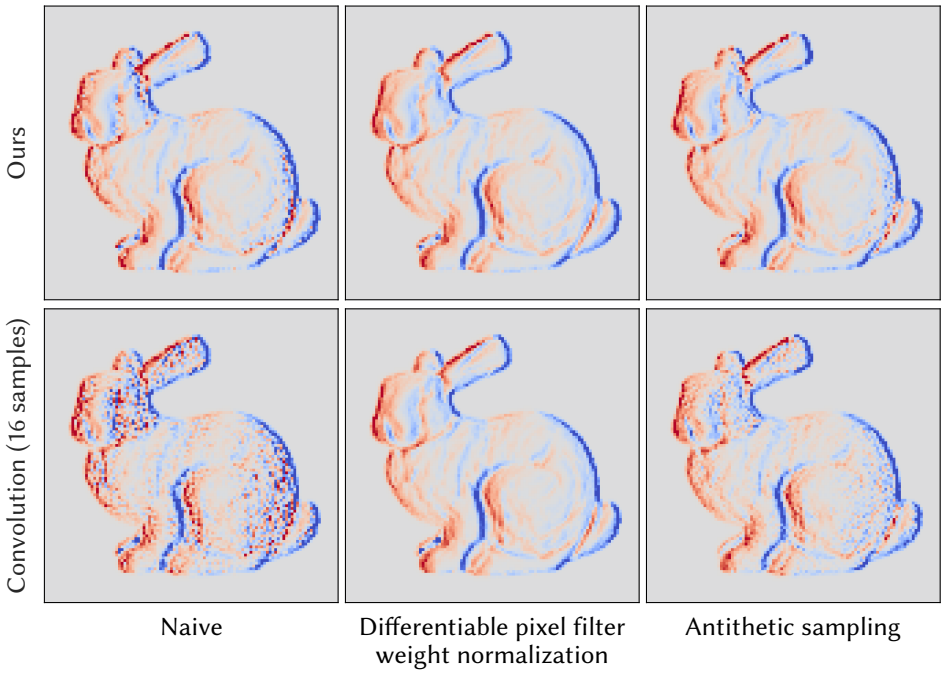


Figure 6.13: This figure shows the image gradients computed using 4 samples per pixels both using our method and the convolution method [69], the latter using 16 samples to estimate the inner convolution integral. We can reduce the noise of the gradients by using a differentiable pixel filter weight normalization or using antithetic sampling.

accounting for indirect illumination, which allows to use the shape's mirror reflection as an additional constraint.

Gradient variance. In Figure 6.13, we visualize forward-mode gradients computed using a low number of samples per pixel. We compare a naïve implementation to one that uses antithetic sampling of the pixel filter, and one that simply keeps the pixel filter normalization weights attached to the differentiation, as discussed in Section 6.1.3. We can see that the differentiable normalization weights reduce the variance both for ours and prior work, and seem to perform slightly better than antithetic sampling. We therefore use the differentiable weight normalization for all implemented methods.

Comparison to using only the shading gradient. While from a theoretical point of view it is clear that we need to account for visibility discontinuities, it is worth validating that not doing so does not work. In Figure 6.14 we run a simple optimization both using our method and an implementation that does not reparameterize discontinuities. While at first glance the gradient images look quite similar, the missing edge gradients make the optimization diverge completely.

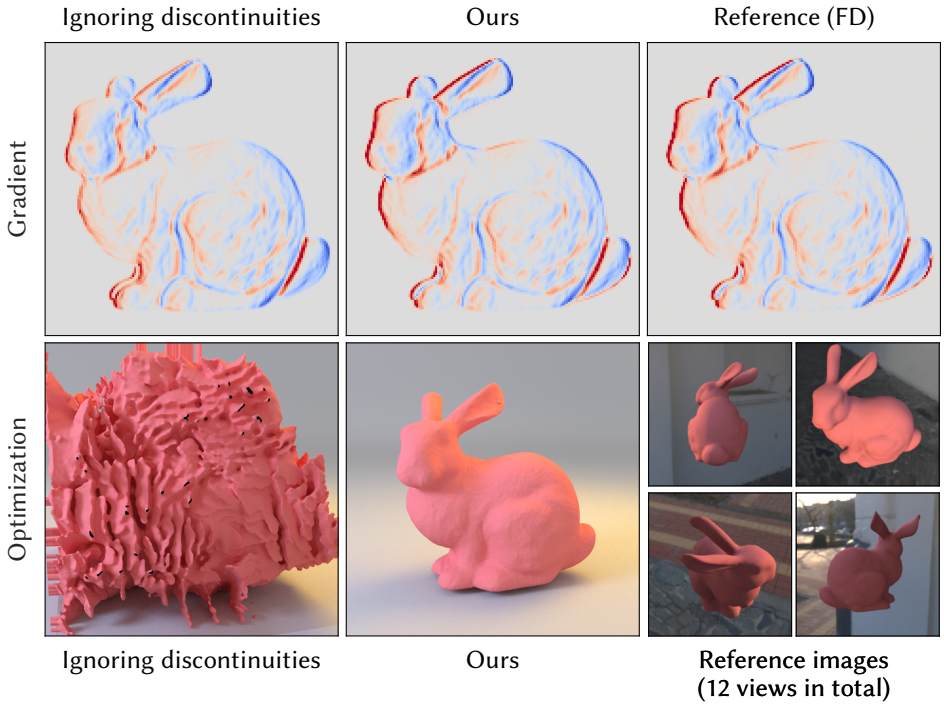


Figure 6.14: **Top row:** Ignoring discontinuities misses important gradient contributions due to occlusion effects. **Bottom row:** Using such strongly biased gradients in an optimization is very likely to completely diverge.

Non-convexity of surface-based reconstruction. Directly optimizing surfaces represented as SDFs can work surprisingly well in many cases. However, the surface reconstruction problem itself can exhibit undesirable local minima, in particular in the presence of complex topology (i.e., objects with holes). This sometimes causes undesirable connections between object parts that should remain disjoint. The gradient information is then not always sufficient to infer that an opening must be created. Figure 6.15 shows such an example, where gradient descent is unable to correctly reconstruct the topology of a complex object.

6.4 Summary and future work

We presented a novel approach to the problem of differentiable rendering of signed distance functions. Our method efficiently computes accurate gradients for image-based optimization of SDFs. We exploit the computational structure of sphere tracing, convenient properties of the SDF representation, and the flexibility admitted by the reparameterization framework.

Our reparameterization handles the discontinuities caused by SDFs, but supporting

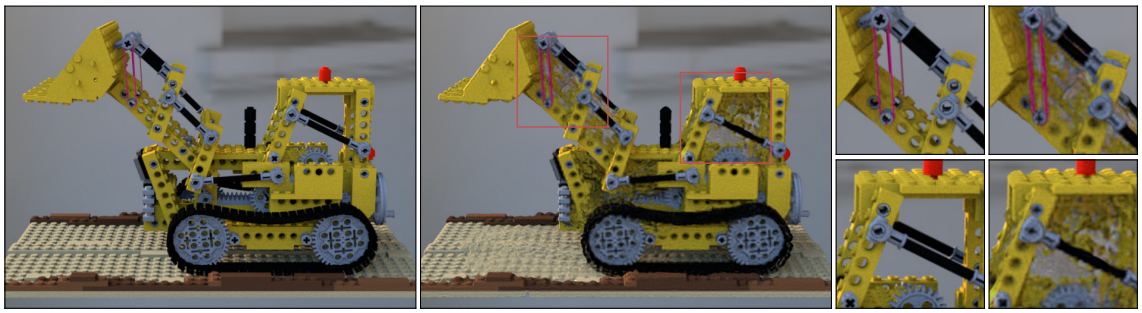


Figure 6.15: The presence of complex topology and spatially varying albedo textures can result in challenging, non-convex optimization problems with many undesirable local minima. In this example, we optimized SDF and albedo texture at a resolution of 256^3 using 40 input images. The result is rendered using the same environment map that is used during optimization.

efficient combination with other shape representations remains future work (e.g., a triangle mesh occluding an SDF). It would be interesting to combine our method with a sparse data structure storing the SDF values to allow scaling the SDF resolution more adaptively.

While our method works well with minimal regularization, investigating more advanced regularization methods could be worthwhile to further improve results. One could for example apply Sobolev preconditioned gradient descent, which has been used successfully for SDF reconstruction [283] and differentiable mesh rendering [284]. Such preconditioning is effective at reducing variance due to sparse gradients, which might remove the need for coarse-to-fine optimization. However, for SDFs the expected overall improvements are smaller than for triangle meshes, which often degenerate when using naïve gradient descent.

For practical shape reconstruction, the main limitation is the inherent non-convexity of direct surface optimization. This could be remedied by either using a more sophisticated method to initialize the SDF, or by using some form of semi-transparency. More concretely, one could try to extend VolSDF [233] with a physically-based illumination model by exploring connections of volume and surface rendering [285]. Related to this, the next chapter investigates how the volumetric transmittance model can be modified to better approximate surfaces.

Lastly, it would be interesting to apply a similar reparameterization approach to triangle meshes. Instead of using a convolution, one could construct a reparameterization during the traversal of the ray acceleration data structure. This could reduce both variance and bias compared to existing gradient estimators.

7 | A Non-Exponential Transmittance Model for Volumetric Scene Representation

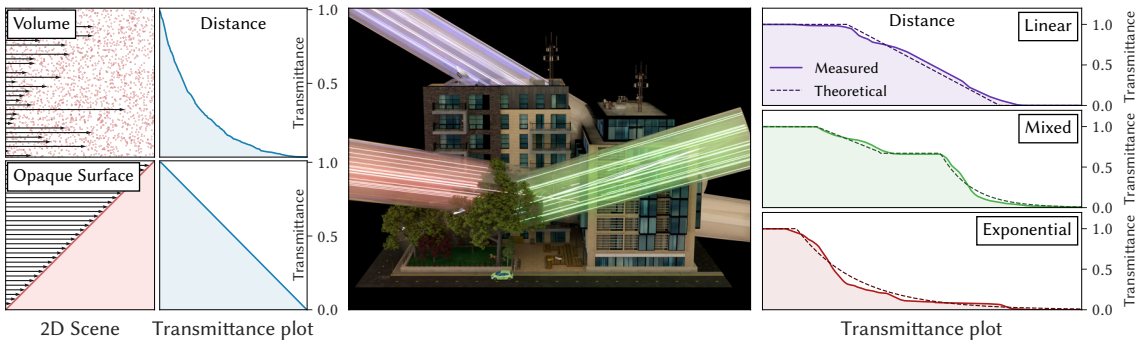


Figure 7.1: **Left:** Consider the fraction of unoccluded rays in two flatland scenarios involving uncorrelated particles (top) and an opaque surface (bottom). The former leads to the standard exponential decay, while the surface case exhibits an unusual linear decay profile. **Right:** We perform a similar experiment in a more complex scene by tracing rays within thick beams (middle) and tracking their free-flight distance. The distributions resulting from the three beams are plotted on the right. Hard surfaces induce linear transmittance (purple), while unstructured geometry like foliage resembles an uncorrelated medium that yields exponential transmittance (red). A beam that first traverses the trees and then a hard surface (green) encounters both linear and exponential transmittance. We also show a parametric fit (dotted plots) using either an exponential or linear model, or piecewise combination of the two in case of the mixed example.

The choice of scene representation is a key design decision for inverse rendering. As shown in the previous chapter, differentiable rendering of implicit surfaces can work well for certain objects, but struggles with fine structures and aggregate geometry (e.g., vegetation). In these cases, optimizing surface geometry becomes highly non-convex. One approach to this problem is to forego surface optimization and optimize a volumetric representation. As we will see in this chapter, this comes with a different set of challenges.

The scene reconstruction problem is closely related to the *level of detail* problem, which was the initial motivation for the work presented in this chapter. Accurate representation of digital scenes requires significant detail in both geometry and textures and can easily exceed the allotted storage or rendering budget. Depending on the chosen view point, the level of detail of the scene may be grossly inappropriate, with an entire

forest or city block visible in a small image region. Level of detail (LoD) techniques are therefore essential [286] and widely used to represent complex photorealistic scenes in production environments [287]. This is especially important for mobile and wearable augmented and virtual reality (AR/VR) devices, where resource limits place stringent requirements on rendering efficiency [288].

Level of detail techniques are oftentimes based on locally simplifying scene geometry. This can for example be done by iteratively collapsing edges [289] until a certain target triangle density is reached. However, such an approach cannot efficiently filter the appearance of fine structures, such as tree leaves, where accurate modeling of partial visibility is required. Ultimately, at farther distances, any geometric structure only achieves partial coverage at the sub-pixel level and can be more efficiently approximated using a volumetric representation [290].

Similarly, volumetric representations are advantageous for image-based scene reconstruction, where they can better handle "fuzzy" or sub-pixel detail than a surface-based representation. Volume parameters can be reconstructed from input photographs using a differentiable renderer and can be represented using a simple uniform grid or a neural network. The volumetric scene representation is smooth and, in contrast to surface rendering, does not require any special treatment of visibility discontinuities.

One challenge is that the theory of radiative transfer is designed to model absorption and scattering of a medium consisting of identically distributed and uncorrelated microscopic particles, but not opaque surfaces. The assumption of uncorrelated particles results in an exponential transmittance function. This model is well suited for unstructured content such as the leaves of a tree, but breaks down for structured and opaque geometry where the assumption of uncorrelated particles is violated. A distinguishing property of surfaces is their ability to be fully *opaque*, i.e., impenetrable to light, which is not possible with classic exponential volumetric light transport theory. When an opaque object is modeled as an exponential volume, a low density leads to a significant portion of light leaking through the surface. Increasing the volume density to compensate leads to bloated silhouettes and an overly opaque appearance of semitransparent parts of the model. Volumetric modeling of scenes containing both opaque and semitransparent elements remains a fundamental challenge in current volumetric modeling approaches.

We build on the observation that approximating the transmittance behavior of an arbitrary scene requires a *non-exponential* transmittance model, as demonstrated in Figure 7.1. We use the recent advances in non-exponential light transport theory [291, 292] to improve the volumetric representation of 3D scenes. Our novel transmittance model

captures the wide spectrum of transport behaviors induced by geometric configurations ranging from completely opaque surfaces to unstructured geometric aggregates. This allows representing the whole scene in a single unified volumetric light transport framework. We demonstrate this efficiency by achieving state-of-the-art results in applications such as appearance prefiltering (Figure 7.2). By using a unified volumetric framework to model the entirety of the scattering in a scene, we can avoid the problem of separating the scene into partitions modeled using volumetric or surface scattering. A purely volumetric representation also simplifies the implementation of both forward and inverse rendering algorithms.

Our non-exponential transmittance formulation can be useful for various tasks that require efficient volumetric representation of an opaque scene. In addition to appearance prefiltering, we also show improvements for scene reconstruction using differentiable rendering. In summary, our core contributions are:

- A unified volumetric representation that handles both opaque surfaces as well as aggregate geometries within a volumetric light transport framework.

- A new practical parametric model for heterogeneous non-exponential transmittance to account for correlations in a continuum from opaque to aggregate geometric configurations.

- A new scene appearance prefiltering method based on our unified volumetric representation and a robust scene-scale parameter optimization routine.

- Efficient image-based volumetric reconstruction of complex scenes using differentiable rendering. We also show some experiments using neural radiance fields (NeRFs) [223].

In the remainder of the chapter, we will first introduce the necessary background on appearance prefiltering and non-exponential media. We then proceed to introduce our new volumetric representation and transmittance model. We then demonstrate and discuss the application of our model to appearance prefiltering and image-based reconstruction.

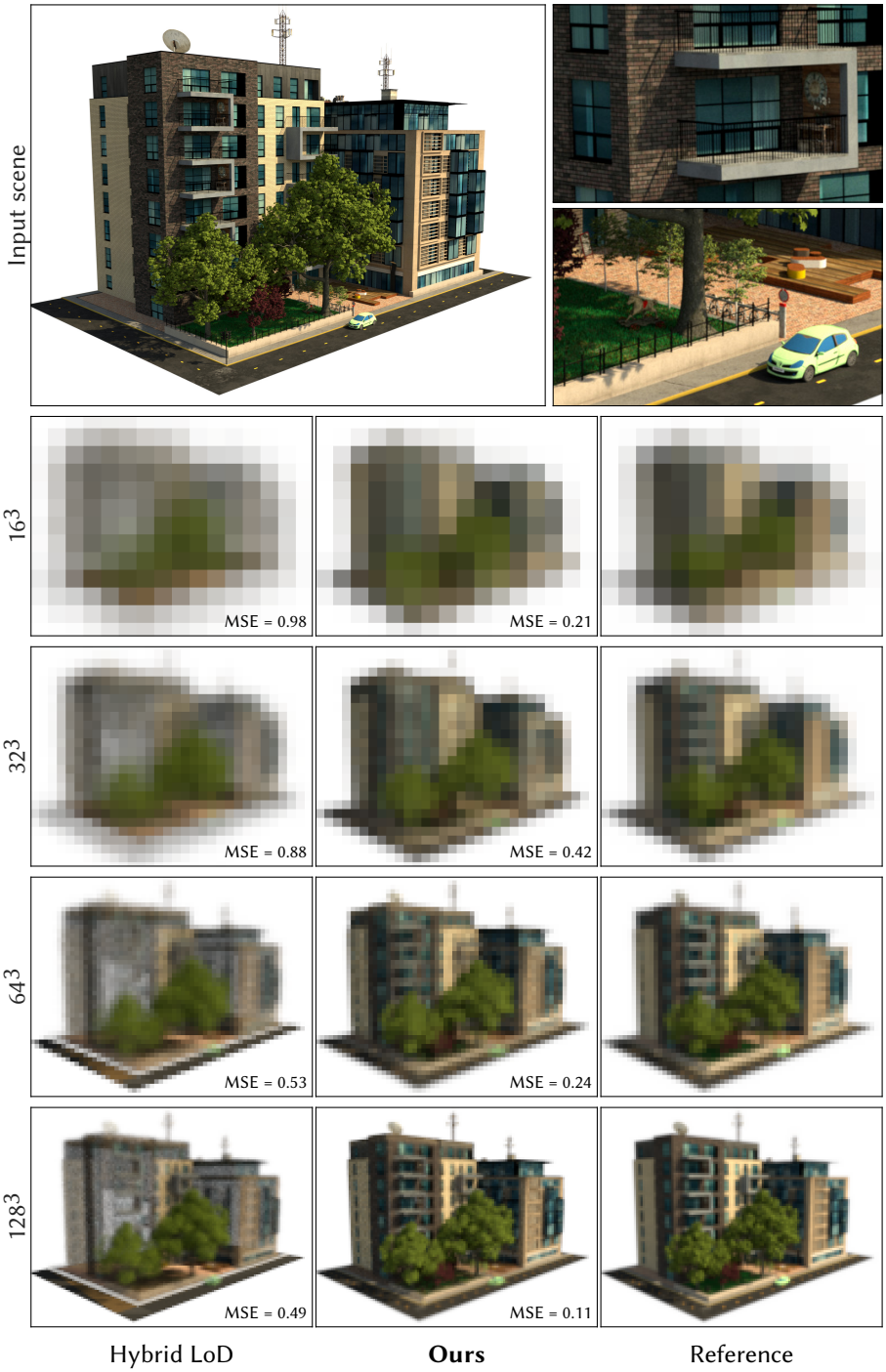


Figure 7.2: Prefiltered rendering of a complex scene (13.8 million triangles) at different resolutions. In each rendering, the original scene is represented as a volume with a voxel grid resolution corresponding to the image resolution. Our method reproduces the appearance of the ground truth reference, while the state-of-the-art *Hybrid LoD* [290] method has difficulties modeling the scene's transmittance function leading to less accurate results. The original scene consists of 750MB of geometry data and 350MB of textures. Our volumetric representation compresses this down to 4.9MB (**220x** compression) for a resolution of 128^3, and even further at lower resolutions. All results are rendered using global illumination.

7.1 Background

7.1.1 Appearance prefiltering

Automatically reducing the complexity of rendered objects or entire scenes is a long-standing problem in rendering. A range of methods have been proposed to handle different aspects of the problem. For example, LEAN mapping [293] prefilters the appearance of normal mapped specular surfaces by converting high-frequency surface normal variation to the roughness of a microfacet BSDF. Specialized methods have also been proposed for displacement maps [294, 295], cloth [216], fur [296], heterogeneous scattering volumes [93] and granular media [297, 298, 299]. These methods either use statistical properties to obtain a closed-form solution for LoD parameters, or run a localized parameter optimization to minimize the difference in appearance between the low and high-resolution versions of an asset. Concurrent work by Hasselgren et al. [300] uses a differentiable rasterizer to optimize the appearance of level of detail meshes.

Reducing the complexity of entire scenes by converting opaque surfaces into a volumetric representation has been a central problem [301]. A common idea is to convert the scene into a hierarchical volumetric representation, e.g., implemented using a sparse octree. In its hierarchical form, this allows to efficiently bound the per-pixel rendering complexity by choosing the prefiltered scale proportionally to the pixel's footprint. The rendering of a volumetric representation can be accelerated by using sparse data structures [302] and empty space skipping [303, 304]. Some of these techniques also find application in volume rendering for scientific visualization [305].

The main challenge with representing surfaces as volumes is in preserving their opaqueness as well as thin structures. This makes volumetric representations suffer either from bloated appearance (density is too high) or light leaking (density is too low), often leading to both at different viewing directions. In order to reduce this bloating and light leaking, Heitz and Neyret [306] used a per-voxel, implicit plane to compute a view-dependent coverage mask.

The state-of-the-art method for general, appearance-preserving LoD is the hybrid approach proposed by Loubet and Neyret [290]. It performs a heterogeneous simplification of a scene by labeling parts of a surface mesh to use either a geometric or a voxelized volumetric representation at every rendering scale. While their method can achieve good results for both opaque geometry and aggregate geometry, this binary classification is an ill-posed problem, as there is a smooth transient phase between surface-like and volumetric appearance. Moreover, their binary classification algorithm is based on a heuristic

considering local mesh topology, and can misclassify complex, unstructured geometry, resulting in opaque surfaces appearing semitransparent, as shown in Figure 7.2.

7.1.2 Non-exponential media

Our work focuses on representing arbitrary scenes in the volumetric rendering framework by introducing a new transmittance model. The transmittance function $\mathrm{T}(\mathbf{x}, \mathbf{y})$ of a participating medium describes the fractional visibility between two points $\mathbf{x}$ and $\mathbf{y}$. In the most general case, the transmittance function maps two 3D points to a scalar. By definition, it is non-increasing, attaining values in the interval $[0, 1]$, and reciprocal, i.e., $\mathrm{T}(\mathbf{x}, \mathbf{y}) = \mathrm{T}(\mathbf{y}, \mathbf{x})$. If the medium is made up of uncorrelated particles, the transmittance follows the Beer-Lambert law and can be expressed as:

$$\mathrm{T}(\mathbf{x}, \mathbf{y}) = \exp\left(-\int_0^{\|\mathbf{x}-\mathbf{y}\|} \sigma_t(\mathbf{x}_t)\,\mathrm{d}t\right), \tag{7.1}$$

where $\mathbf{x}_t$ is the position at a distance t on the segment between $\mathbf{x}$ and $\mathbf{y}$.

On the other hand, in a scene consisting of opaque surfaces, the transmittance is simply the binary visibility between points. Our goal is to model a prefiltered version of this function, where the prefiltering kernel size is determined by the resolution of the voxel grid storing the volume parameters. Conceptually, this prefiltered transmittance function is the result of the convolution of the original function with a kernel K:

$$\mathrm{T}_{\text{prefiltered}}(\mathbf{x}, \mathbf{y}) = \int_{\mathbb{R}^3}\int_{\mathbb{R}^3} \mathrm{K}(\mathbf{s}, \mathbf{t})\mathrm{T}(\mathbf{x}+\mathbf{s}, \mathbf{y}+\mathbf{t})\,\mathrm{d}\mathbf{s}\,\mathrm{d}\mathbf{t}. \tag{7.2}$$

This prefiltered transmittance function does not have to be exponential. When representing a complex 3D scene as a volume, transmittance generally takes one of two main modes, as shown in Figure 7.1: classic exponential mode (e.g., with unstructured geometry, like leaves), as well as linear mode (e.g., when hitting a planar opaque surface). In different parts of the scene, the transmittance may vary continuously between these two extremes.

Non-exponential transmittance functions have recently been introduced to computer graphics [291, 292] to handle more general cases of participating media, such as crystal structures, fabric, or water droplets formed in clouds, where microparticles exhibit some form of correlation. Bitterli et al. [292] proposed a reciprocal path integral formulation for non-exponential media with correlated particles. Concurrent work by Jarabo et al. [291] extends the non-exponential generalized Boltzmann equation (GBE) with support for boundary surfaces to be compatible with rendering algorithms. D'Eon [307]

introduced a weakly reciprocal non-exponential rendering formulation and investigated diffusion approximations [308] and binary mixtures of scatterers [309]. Several previous works [291, 292, 310, 311] have introduced parametric families of non-exponential transmittance functions. These parametric models mostly cover the space of transmittance functions which fall off less quickly than the exponential function (e.g., due to clumping of medium particles). However, our method needs to cover the spectrum between linear and exponential transmittance, which is not supported by these models. Jarabo et al. [291] discuss how particles aligned in certain grid structures give rise to a linear transmittance function and show example renderings using homogeneous media.

Non-exponential volume rendering equation. To render images with volumetric light scattering, the transmittance function is inserted into the volume rendering equation. In the following, we only consider non-emissive volumes. As discussed in Section 3.4, the outgoing radiance is given by the volume rendering equation:

$$L_o(\mathbf{x}, \boldsymbol{\omega}_o) = \int_0^s \sigma_t(\mathbf{x}_t)\mathrm{T}(\mathbf{x}, \mathbf{x}_t)\ \alpha(\mathbf{x}_t) \int_{\mathcal{S}^2} f_p(\mathbf{x}_t, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i) L_i(\mathbf{x}_t, \boldsymbol{\omega}_i)\, \mathrm{d}\boldsymbol{\omega}_i\, \mathrm{d}t + \mathrm{T}(\mathbf{x}, \mathbf{x}_s) L_o(\mathbf{x}_s, \boldsymbol{\omega}_o), \tag{7.3}$$

where $\alpha(\mathbf{x}_t)$ is the medium's albedo. Volumetric path tracing then constructs light paths by alternating between sampling the free-flight distance and the scattered direction. In an exponential medium, the free-flight distance is sampled using the density to $p(t) = \sigma_t(\mathbf{x}_t)\mathrm{T}(\mathbf{x}, \mathbf{x}_t)$.

This equation assumes the transmittance function to be exponential. One key observation made in previous work [291, 292] is that simply replacing T by an arbitrary function is not energy preserving. The solution to that issue lies in realizing that the term $\sigma_t(\mathbf{x}_t)\mathrm{T}(\mathbf{x}, \mathbf{x}_t)$ inside the integral in Equation 7.3 is exactly the physical free-flight distance probability density function of the exponential medium. The transmittance is defined as the probability of a photon traversing a region of space without collision and the corresponding free-flight distance PDF is its negative derivative. That T itself is a factor in that term is simply due to it being an exponential function. Moving to a non-exponential transmittance, this whole term has to be replaced by the associated free-flight distance PDF, as opposed to merely replacing the function T in the integral. Denoting the free-flight distance PDF by $\mathrm{T}_{\mathrm{pdf}}$, we then obtain a generalized volume ren-

dering equation:

$$L_o(\mathbf{x}, \boldsymbol{\omega}_o) = \int_0^s \mathrm{T}_{\mathrm{pdf}}(\mathbf{x}, \mathbf{x}_t)\ \alpha(\mathbf{x}_t) \int_{\mathcal{S}^2} f_p(\mathbf{x}_t, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i) L_i(\mathbf{x}_t, \boldsymbol{\omega}_i)\, \mathrm{d}\boldsymbol{\omega}_i\, \mathrm{d}t + \mathrm{T}(\mathbf{x}, \mathbf{x}_z) L_o(\mathbf{x}_s, \boldsymbol{\omega}_o), \tag{7.4}$$

where $\mathrm{T}_{\mathrm{pdf}}(\mathbf{x}, \mathbf{x}_t) = -\partial_t \mathrm{T}(\mathbf{x}, \mathbf{x}_t)$. This simple formulation of the non-exponential transport has one caveat: If we want the light transport to be reciprocal, the transmittance function and free-flight distance distribution need to be different if the path starts on a surface or medium boundary [292, 307]. Our work is aimed at representing entire scenes as a participating medium. We thus use this simplified model and do not support combining our representation with separate surfaces.

7.2 Heterogeneous non-exponential transmittance

7.2.1 Transmittance model

In this section, we present our novel heterogeneous non-exponential transmittance formulation. We propose an integral formulation, where the transmittance behavior can change from one region of the medium to another, specifically to model the transmittance functions that occur in opaque 3D scenes. In previous work on non-exponential media, the transmittance function is assumed to be parametric on the optical depth and is fixed throughout the medium. The heterogeneity model proposed by Bitterli et al. [292], similar to Camminady et al. [312], expresses the transmittance in a heterogeneous medium as

$$\mathrm{T}(\mathbf{x}, \mathbf{y}) = f(\tau) = f\left(\int_0^{\|\mathbf{x}-\mathbf{y}\|} \sigma_t(t)\, \mathrm{d}t\right), \tag{7.5}$$

where f is the transmittance function, the extinction $\sigma_t(t) \coloneqq \sigma_t(\mathbf{x}_t)$ is evaluated at distance t along the segment between $\mathbf{x}$ and $\mathbf{y}$; and τ is the optical depth.

This formulation has the advantage that it naturally fits into a reciprocal rendering framework and for some special cases even admits unbiased sampling of the free-flight distance. However, as a limitation, it does not support continuously varying the transmittance behavior inside of a medium. This is necessary if we want to be able to capture different transmittance modes present in a single scene. Note that subdividing the scene into homogeneous regions with different non-exponential transmittances would not solve this problem, as this would be restricted to using a voxel grid representation

with nearest-neighbor interpolation. Jarabo et al. [291] discuss handling heterogeneous volumes in a such a way, but also acknowledge its limitations and do not provide a practical algorithm. Our goal is to allow for spatial variation in the transmittance function f.

For scene representation, we need to cover the spectrum between exponential and linear transmittance. While the exponential behavior arises due to photons interacting with uncorrelated medium particles, linear transmittance can be achieved by placing particles in a regular grid or crystal-like structure [291, 313]. Such models could potentially be useful to form a direct connection from the scene's geometry to medium parameters. However, it seems to be difficult to smoothly transition from a linear to an exponential transmittance using such theoretical scatterer distributions. Therefore, we instead build a simple parametric model that can represent the desired appearance space and fit its parameters using gradient descent. Since we need to capture the full spectrum from linear to exponential transmittance, we base our model on a linear combination of the two:

$$f(\tau, \gamma) = \gamma \exp(-\tau) + (1 - \gamma) \max(0, 1 - \tau/2), \tag{7.6}$$

where γ varies between 0 and 1. We will refer to this parameter as *transmittance mode*. The division by two in the linear transmittance ensures that exponential and linear transmittance have the same mean free path.

Given this transmittance function, we need to be able to vary γ spatially. In the following, we present a novel transmittance framework that enables this. The derivation of the framework is independent of the concrete form of f. We construct our model such that it satisfies the following requirements:

1. The formulation needs to be an extension of the traditional exponential transmittance and previous heterogeneous non-exponential transmittance formulations.

2. The transmittance function needs to be non-increasing.

3. The transmittance function should be continuous for continuously varying parameters.

4. Evaluating the transmittance should have similar memory and computational requirements as conventional heterogeneous media.

We do not make any explicit assumptions about the particle correlation in the medium and also do not enforce reciprocity. Our goal is to merely introduce the additional degree

of freedom to spatially vary the transmittance behavior. To achieve this, we write the transmittance as a recursive integral over transmittance function derivatives:

$$\mathrm{T}(\mathbf{x},\mathbf{y}) = 1 + \int_0^{\|\mathbf{x}-\mathbf{y}\|} f_\tau\left(f^{-1}\left(\mathrm{T}(\mathbf{x},\mathbf{x}_t),\gamma(t)\right),\gamma(t)\right)\sigma_t(t)\,\mathrm{d}t \tag{7.7}$$

where f_τ is the partial derivative of f with respect to the optical depth. This formulation satisfies requirements (1) – (4), as we explain in the rest of this and the following section. In particular, to show that requirement (1) holds and to justify our formulation, we first consider a constant transmittance mode, i.e., $f(\tau,\gamma) = f(\tau)$. In that case, we can directly derive our formulation from the previous model (Equation 7.5):

$$\begin{aligned}
\mathrm{T}(\mathbf{x},\mathbf{y}) &= f\left(\int_0^{\|\mathbf{x}-\mathbf{y}\|}\sigma_t(t)\,\mathrm{d}t\right)\\
&= 1 + \int_0^{\|\mathbf{x}-\mathbf{y}\|}\frac{\partial}{\partial t}\left[f\left(\int_0^t \sigma_t(s)\,\mathrm{d}s\right)\right]\mathrm{d}t\\
&= 1 + \int_0^{\|\mathbf{x}-\mathbf{y}\|} f_\tau\left(\int_0^t \sigma_t(s)\,\mathrm{d}s\right)\sigma_t(t)\,\mathrm{d}t\\
&= 1 + \int_0^{\|\mathbf{x}-\mathbf{y}\|} f_\tau\left(f^{-1}\left(\mathrm{T}(\mathbf{x},\mathbf{x}_t)\right)\right)\sigma_t(t)\,\mathrm{d}t.
\end{aligned}$$

The first step uses the fundamental theorem of calculus, the second step applies the chain rule, and in the third step we substitute the optical depth integral using the original transmittance definition, i.e., $\int_0^t \sigma_t(s)\,\mathrm{d}s = f^{-1}(\mathrm{T}(\mathbf{x},\mathbf{x}_t))$. Georgiev et al. [314] used a similar integral formulation to derive new transmittance estimators for exponential light transport. They derived their formulation directly from the RTE, whereas we leveraged the fundamental theorem of calculus.

The requirements (2) and (3) are satisfied by definition and we show how to evaluate the model efficiently in the next section. While satisfying these requirements, our model's simple form has the downside that it does not explicitly track correlation across voxels. Just modifying the transmittance behavior per voxel is still an approximation of the ground truth transmittance function. The only "state" that we carry along a ray is the transmittance up to the current location. As a consequence of this approximation, our model ends up being non-reciprocal. A reciprocal model most likely would need to track some form of correlation between encountered voxels to more faithfully represent the true transmittance function. Combining reciprocity and heterogeneous transmittance behavior remains an important avenue for future work. Since we use our model in conjunction with unidirectional path tracing, we did not find non-reciprocity to be a noticeable issue in any of our experiments. We provide an evaluation of the impact of non-reciprocity when discussing results (Section 7.6).

7.2.2 Evaluation and sampling

To use this model for rendering, we need to evaluate and sample the transmittance function. We use ray marching to estimate transmittance and sample free-flight distances. Ray marching approximates the transmittance integral using a quadrature based on a set of evenly spaced locations along the ray. We found that naïvely approximating the integral from Equation 7.7 using quadrature does not work well. Our model was derived from the original transmittance formulation using the fundamental theorem of calculus. If we want this relation to hold when performing quadrature, we cannot simply evaluate the integrand as is. We need to use a discrete approximation of the derivative terms which are part of the integrand. Otherwise, the error of the quadrature can be almost arbitrarily high.

We, therefore, replace the analytic transmittance derivative by a finite difference approximation, where we use a step size proportional to the medium extinction and the ray marching step size. We provide an expanded explanation and derivation in Appendix D.1. The ray marching algorithm then simplifies to evaluating the following recursive expression:

$$\mathrm{T}(\mathbf{x}, \mathbf{y}) \approx f\left(f^{-1}\left(\mathrm{T}_{N-1}, \gamma(t_i)\right) + \sigma_t(t_i)\Delta_{\text{step}}, \gamma(t_i)\right), \tag{7.8}$$

where N is the number of steps in the ray marching routine, t_i the distance in the current step, Δ_{step} the step size and T_i refers to the transmittance from iteration i in the evaluation (with $\mathrm{T}_0 = 1$). We provide pseudocode for the transmittance evaluation in Listing 7.1. This formulation has a simple intuitive meaning: It can be seen as iteratively decreasing the transmittance according to the local transmittance model and density.

Free-flight distance sampling. We sample the free-flight distance using standard inverse transform sampling. We first draw a uniform random number $U \sim \mathcal{U}(0, 1)$ and then find t such that $\mathrm{T}(\mathbf{x}, \mathbf{x}_t) = U$ by marching along the ray. We perform a bisection search on the last segment to precisely determine the sampled distance.

This works because the transmittance is defined to be one minus the CDF of the free-flight distance distribution. The PDF of the resulting sample is then the negative derivative of the final transmittance value. Since the transmittance is a definite integral evaluated from 0 to t, the derivative of the transmittance with respect to t is simply the integrand itself:

$$\mathrm{T}_{\text{pdf}}(t) = -f_\tau\left(f^{-1}\left(\mathrm{T}(\mathbf{x}, \mathbf{x}_t), \gamma(t)\right), \gamma(t)\right)\sigma_t(t). \tag{7.9}$$

```
def transmittance(σ_t, γ):
    T = 1
    for i in range(N):
        t = i · Δ_step # Evaluate distance along ray
        T = f[f^{-1}(T, γ(t)) + σ_t(t) · Δ_step, γ(t)]
    return T
```

Algorithm 7.1: Pseudocode to evaluate the transmittance using our model.

We assume the extinction and transmittance mode to be monochromatic, which means that this PDF cancels out with the integrand during forward rendering.

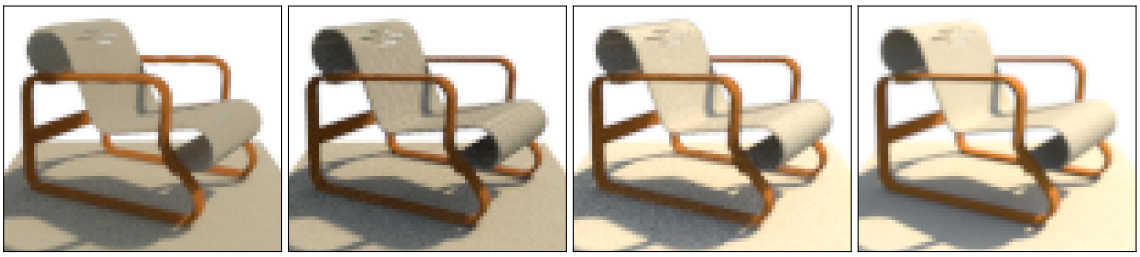
7.3 Volumetric appearance model

In this section, we describe the full set of parameters required by our unified volumetric scene representation. Our representation is parametrized by albedo α, phase function parameters and transmittance parameters. For our main experiments, all parameters are stored on a voxel grid and interpolated trilinearly between voxels.

Transmittance. The transmittance is modeled by our heterogeneous transmittance model introduced in the previous section. We store both extinction σ_t and transmittance mode γ on voxel grids. These parameters are monochromatic, but the method could be extended to handle spectrally varying transmittance parameters.

Phase function. The phase function describes the angular distribution of scattering in a volume and plays a crucial role in representing complex appearance. Since we are interested in representing surfaces, it is preferable to use an anisotropic phase function [90]. We use the SGGX microflake phase function [91], as it is straightforward to use and has robust sampling and evaluation routines. The ellipsoid defining the SGGX' NDF is parameterized by 6 degrees of freedom, which we also store on a grid. Unless specified otherwise, we use an SGGX microflake phase function with diffuse microflakes for all our results.

When using a microflake phase function, the extinction is scaled by the projected area of the microflakes (computed from the NDF). This is useful in our context, as otherwise we do not model any directional dependence of the transmittance. For example, the SGGX microflake distribution can almost perfectly model the discrete NDF of a flat plane. The projected microflake area will then tend to zero as we approach grazing angles, just



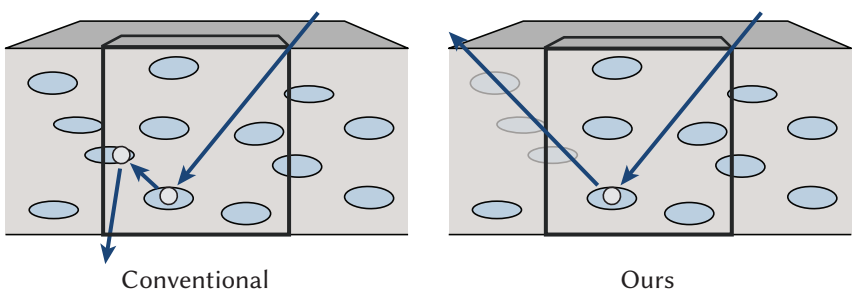
Isotropic SGGX w/o offset Ours Reference

Figure 7.3: This figure illustrates the difference between using an isotropic phase function, a standard SGGX phase function and an SGGX phase function with an offset after scattering (**Ours**). Applying an offset reduces energy loss and allows to more faithfully approximate the reference appearance.

as we would expect for the opacity of plane. For future work, it would be interesting to develop a tighter coupling between microflake theory and non-exponential media. Further, the modeling power of the phase functions is one of the main factors limiting the generality of our methods.

Separation of local and global scattering. As described, the presented model does not in any way restrict multiple scattering inside a voxel. This means that even when using linear transmittance and a sufficiently high extinction, we can still end up with light paths eventually going through an opaque surface due to multiple scattering. This does not only result in light leaking, but also produces significant energy loss on the visible side of a surface.

One possible approach to this problem could be to use one-sided microflakes, which are transparent from the backside [285]. However, generalizing this concept to arbitrary scenes is difficult. We chose a simpler solution: after each medium scatter event, we offset the start of the next ray by the size of a voxel. This is similar to using an epsilon for shadow rays to prevent self-intersections. The following drawing illustrates the issue. Conventional path tracing would simulate multiple scattering inside a voxel and the path could eventually reach the other side. By offsetting the ray origin, this is mostly prevented:



Conventional Ours

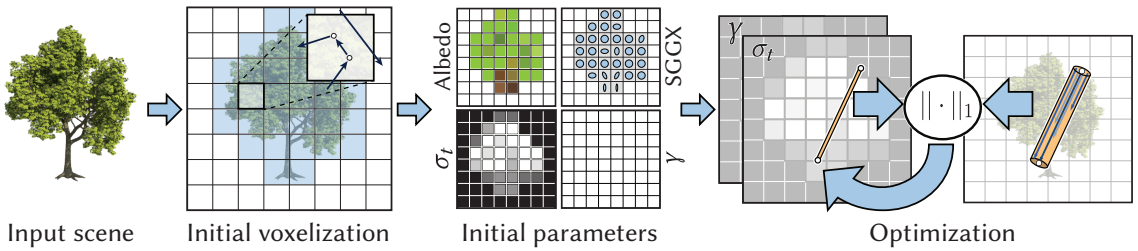


Figure 7.4: Overview of our scene prefiltering pipeline. We first obtain an initial binary voxelization of the scene and then trace rays in each occupied voxel to determine local appearance parameters. We further refine the transmittance model by optimizing the σ_t and γ parameters using gradient descent. During the optimization, we sample random line segments in the scene and compute an L_1 loss between our transmittance model and the reference transmittance (estimated using ray tracing). This whole pipeline is run once for each target resolution.

We show the practical impact of this offset on an example scene in Figure 7.3. Offsetting the ray origin is related to shell tracing [297, 299], where local transport is summarized over a spherical region and the light path proceeds from a location sampled on a sphere around the current location in the medium.

7.4 Application: Appearance prefiltering

A primary application of our model is appearance prefiltering for level of detail, or scene compression. Given a complex scene, we approximate it using a lower resolution volumetric representation. In our volumetric appearance model, we have to determine the parameters that reproduce a certain target appearance. During rendering, we use a voxel grid resolution that is appropriate for the size of the image pixels. Therefore, we build a hierarchy of voxel grids covering different resolutions, doubling the resolution between each scale. We fit the medium parameters separately for each target resolution, as simply downsampling from the finest resolution would not preserve appearance. The fitting time is dominated by the runtime at the highest resolution, so fitting parameters independently at lower resolutions does not significantly impact processing time.

The fitting pipeline consists of multiple stages illustrated in Figure 7.4. First, we compute a binary voxelization of the scene, marking each non-empty voxel for further processing. We use Binvox [315, 316] to determine the set of non-empty voxels. This first step is important for computational efficiency, since the total number of voxels in a grid of width n is n^3, but the number of occupied surface voxels only increases at a rate of $O(n^2)$. Our goal is to create a *surface voxelization*, i.e., we do not fill in solid objects. This ensures that our representation remains sparse, and therefore efficient to render

Resolution	16	32	64	128	Original
Checkerboards	53.7 KB	0.2 MB	0.9 MB	3.3 MB	36 KB
	37.6%	66.1%	83.9%	92.5%	
Fractal	51.7 KB	0.3 MB	1.5 MB	8.0 MB	42 MB
	39.9%	61.1%	73.4%	81.9%	
City building	49.7 KB	0.2 MB	1.1 MB	4.9 MB	1.1 GB
	42.2%	65.9%	80.5%	88.8%	
Trees	35.4 KB	0.2 MB	1.0 MB	5.0 MB	451 MB
	58.8%	73.9%	82.6%	88.7%	
City	32.6 KB	0.1 MB	0.7 MB	3.4 MB	1.5 GB
	62.1%	80.3%	87.9%	92.4%	

Table 7.1: The resulting *file size* and *percentage of empty voxels* of the sparse volumetric representation for several example scenes. The variation in compressed size is due to different levels of sparsity in the original scenes.

and store. We show compression rates for a few example scenes in Table 7.1.

We then trace light paths in each non-empty voxel and estimate the voxel's albedo by averaging their throughput. Therefore, the albedo of a voxel already accounts for multiple scattering within this voxel.

Phase function For the SGGX phase function, we fit parameters using the algorithm provided by Heitz et al. [91]. The ellipsoid defining the SGGX phase function can be expressed by 3 eigenvectors and corresponding projected areas. To fit these parameters, we first obtain a distribution of surface normals inside the voxel by intersecting rays with the geometry. Given a large number of sampled normals, we then compute the covariance matrix of the components of these normals. By performing an eigendecomposition on this covariance matrix we obtain the eigenvectors of the ellipsoid. The projected areas are then computed by projecting the sampled normals onto these three eigenvectors.

7.4.1 Transmittance optimization

There is no closed-form solution for the extinction σ_t and transmittance mode γ. Even for an exponential medium, the modulation of the extinction by the view-dependent microflake area makes it so that the extinction parameters can only be determined by iterative optimization. Previous work [290] fits the extinction value locally by performing gradient descent. This can be done by tracing a number of rays for each voxel and then

optimizing the extinction value to reproduce the observed directionally varying opacity behavior.

At first, this seems like the right way to solve this problem: to determine the per-voxel extinction, we should just consider what happens inside the region of the scene represented by that single voxel. However, there are two main issues with this approach. First, it completely ignores any correlation effects across voxels. The second issue is less obvious but equally important: Due to the nature of the volumetric approximation, it is typically impossible to perfectly fit the reference scene. By fitting parameters independently per voxel, we effectively prioritize per voxel error over error at larger scales or even image-space error. Empirically we found this to lead to significantly less accurate results than optimizing across multiple voxels. Both the exponential and our model benefit from optimizing transmittance over several voxels, as shown in Figure 7.5

To optimize our transmittance parameters, we first initialize the extinction coefficient of each non-empty voxel by tracing N rays through its region in the reference scene. We then compute

$$\sigma_t^{\text{init}} = -\frac{1}{s} \log \left(\frac{1}{N} \sum_{i=1}^{N} V_i \right), \tag{7.10}$$

where s is the side length of a single voxel and V_i the visibility of sample i (i.e., 1 if the ray passed through the voxel and 0 otherwise). The transmittance mode γ is initialized to 1, which corresponds to exponential transmittance. We then optimize the parameters by minimizing the following objective function:

$$\ell(\sigma_t, \gamma) = \int_{\mathcal{V}} \int_{\mathcal{V}} |\mathrm{T}(\mathbf{x}, \mathbf{y}) - \mathrm{T}^{\text{ref}}(\mathbf{x}, \mathbf{y})| \; p(\mathbf{x}, \mathbf{y}) \, \mathrm{d}\mathbf{x} \, \mathrm{d}\mathbf{y}, \tag{7.11}$$

where $\mathbf{x}$ and $\mathbf{y}$ are 3D points following a distribution $p(\mathbf{x}, \mathbf{y})$. In practice, we sample $\mathbf{x}$ and $\mathbf{y}$ by generating randomly oriented segments passing through occupied voxels. Practically, we found that segments of 20 voxels in length are sufficient for capturing even sophisticated correlations along grazing angles. The reference transmittance $\mathrm{T}^{\text{ref}}(\mathbf{x}, \mathbf{y})$ is computed by tracing rays between the two points and evaluating the binary visibility function. Since our volumetric approximation is bandlimited due to the voxel grid resolution, we also filter the reference transmittance with a small Gaussian kernel of a standard deviation of $s/6$, where s is the voxel size in world units. This means, that instead of just tracing rays between $\mathbf{x}$ and $\mathbf{y}$, we randomly offset the ray origin in a plane perpendicular to the vector between the two points (as illustrated in Figure 7.4). This aids optimization, as it reduces variance in the evaluation of the reference transmittance.

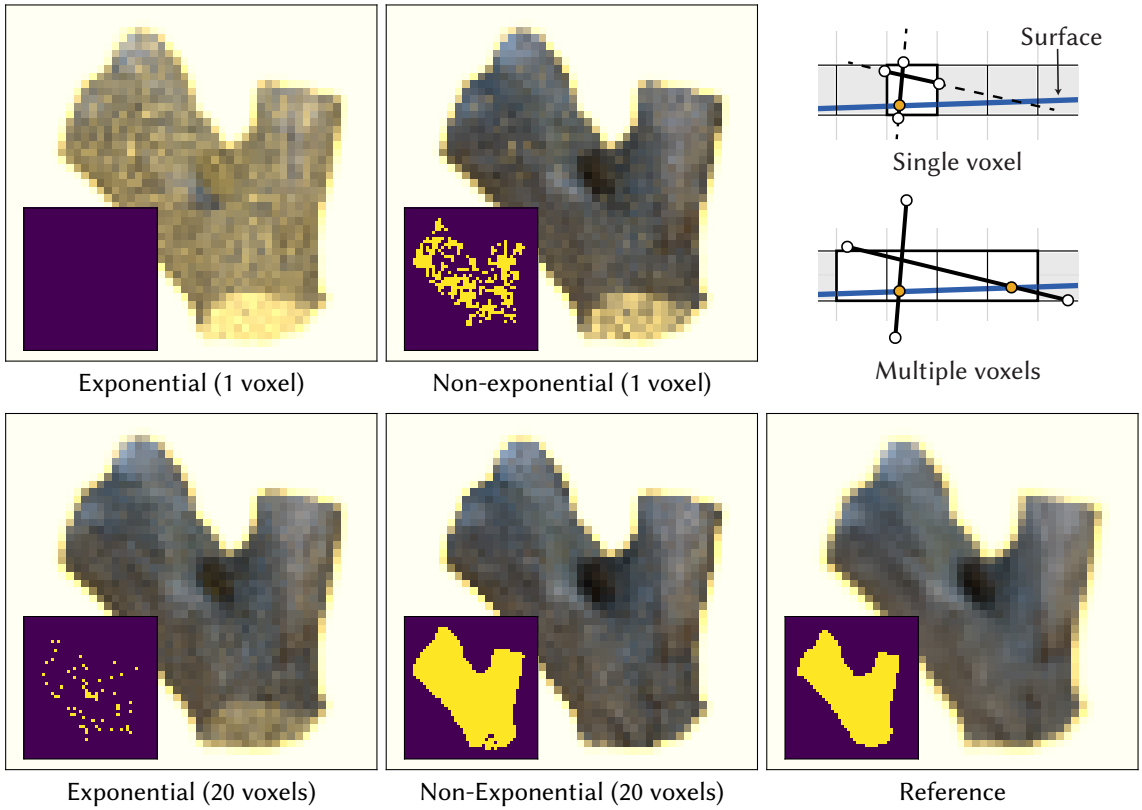


Figure 7.5: Fitting over several voxels can drastically reduce error. In the top row, we fit medium parameters to represent the transmittance behavior of a single voxel. In the bottom row, the extinction is fit over beams of the length of 20 voxels. In these examples, the composited background has a constant RGB color of (4, 3, 0.9), which helps to highlight leakage. In the insets, we highlight pixels with more than 99.99% opacity. The exponential representation does not even come close to being fully opaque. The illustration in the top right shows how fitting over multiple voxels better represents the absorption behavior for rays at grazing angles.

We optimize this loss over batches of 16000 voxels simultaneously using the Adam optimizer [111] and a learning rate of 0.5. We run this optimization for 512 epochs over all occupied voxels.

7.4.2 Transmittance gradient

To optimize our transmittance model, we need to compute the gradient of the objective function with respect to the parameters. Simply using automatic differentiation to optimize transmittance parameters results in high memory consumption, as the state of each loop iteration has to be stored. This was also observed by previous work on optimizing transmittance [250]. We leverage the invertibility of individual loop iterations in our ray marching routine to compute gradients without allocating a large amount of temporary storage (as described in Section 4.2.3). We only store the current wavefront of rays and the volume parameters. We first evaluate the transmittance using ray marching and then compute the objective function and its gradient, which is then propagated to the parameters in a reversed ray marching loop. In each iteration of the reverse loop, we compute the previous loop state using the inverse transmittance function. We validated our analytic gradients against finite differences and automatic differentiation. The pseudocode for our reverse transmittance evaluation is given in Listing 7.2. The adjoint function takes the loss gradient δ_T and the result of the forward pass, T, as inputs. Additionally, we pass the transmittance parameters and corresponding gradient variables, in which the gradients are accumulated.

```
def transmittance_adjoint(δ_T, T, σ_t, γ, δ_σt, δ_γ):
    for i in reversed(range(N)):
        t = i · Δ_step
        σ̂_t = σ_t(t) # Get current medium parameters
        γ̂ = γ(t)
        T = f[f^-1(T, g) − σ̂_t · Δ_step, γ̂] # Inverse loop iteration
        τ = f^-1(T, γ̂) + σ̂_t · Δ_step # Optical depth
        δ_σt += δ_T · Δ_step · f_τ(τ, γ̂) # Accumulate σ_t gradient
        δ_γ += δ_T · [f_τ(τ, γ̂) · f_γ^-1(T, γ̂) + f_γ(τ, γ̂)] # Accumulate γ gradient
        δ_T *= f_τ(τ, γ̂) · f_T^-1(T, γ̂) # Update Jacobian
```

Algorithm 7.2: Pseudocode for the memoryless adjoint transmittance evaluation. Subscripts to functions denote partial derivatives, e.g., $f_\tau = {}^{\partial f}/_{\partial \tau}$.

7.5 Application: Image-based reconstruction

Our non-exponential transmittance model can also be applied to the problem of image-based volumetric scene reconstruction. Our volumetric model is always continuous and therefore, unlike surface-based representation, does not require any edge sampling or reparameterization.

Given a set of reference images, we can optimize extinction values, transmittance mode, phase function parameters and albedo on a voxel grid to match the reference appearance. In our experiments, we use voxel grid resolutions of up to 256^3. For all parameters, we start with a uniform grid as our initial guess. We optimize parameters using differentiable rendering and up to 4 bounces of indirect illumination. A higher number of indirect bounces could be used, but this increases computation time and did not significantly change our results. We use a coarse-to-fine optimization routine to improve the convexity of the problem. We start at a volume and image resolution of 4^3 and 4^2, respectively, and then increase both resolutions after a number of iterations up to reaching the final volume resolution. The low starting resolution not only makes the problem more convex but also makes the initial optimization iterations cheaper to render.

We perform the optimization under a uniform illumination and then re-render the result using a novel illumination condition. The illumination condition we use for the optimization is advantageous as no hard shadows are cast into the scene. We leave the issue of optimizing under more difficult illumination conditions to future work. Our optimization uses a silhouette loss to further reduce ambiguities between fore- and background. We found this to be useful to achieve fully opaque results, even when using our non-exponential model.

7.6 Results

We implemented our method on top of Mitsuba 2 [48] and Enoki [145]. We use an NVIDIA RTX TITAN GPU and run large parts of the pipeline using CUDA and OptiX [147]. For the image-based reconstruction, we implemented a differentiable volume renderer using CUDA and analyic derivatives, building on our path replay backpropagation algorithm. To differentiate transmittance terms, we use the gradient computation from Listing 7.2. This project predates the Dr.Jit compiler [49], but could now easily be implemented using that instead of handwritten CUDA ceode.

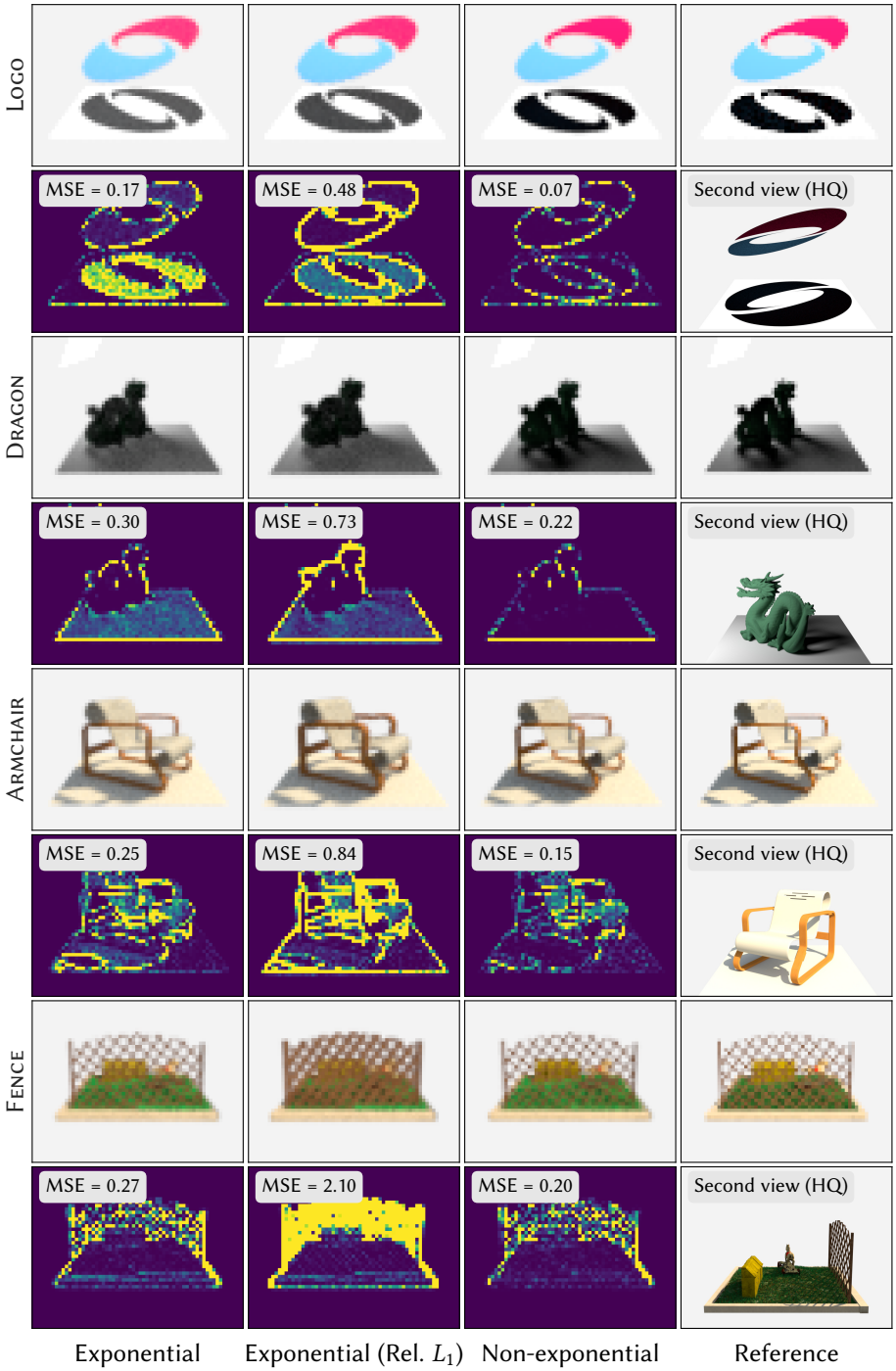


Figure 7.6: These results show the differences between using conventional exponential and our novel non-exponential transmittance. We also compare to the result of optimizing the exponential model using a relative loss, which favors higher extinction values. Regardless of the loss function, the exponential model often leads to light leaking, resulting in overly bright shadows. Using a relative loss, we get bloated appearance on semitransparent structures, such in the Fence scene.

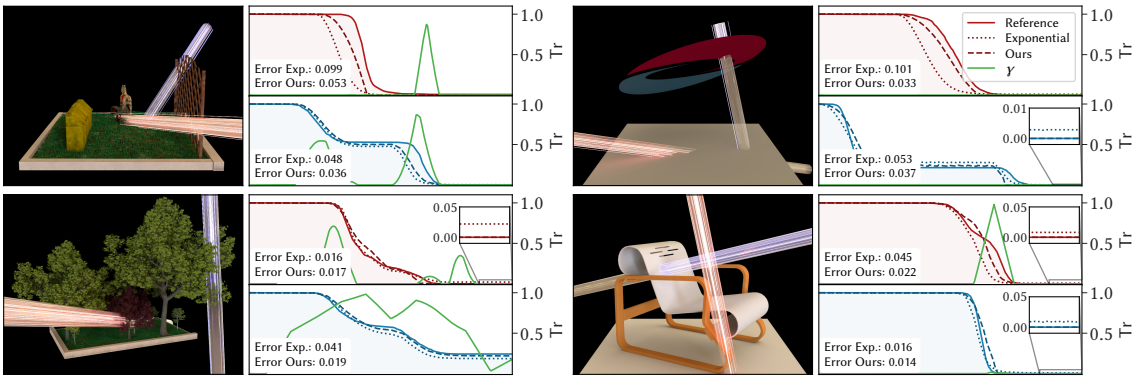


Figure 7.7: We plot the reference simulated transmittance (solid line) against both our non-exponential and exponential transmittance fits. In all these results we use a voxel grid of 16^3 resolution and compare to the ground-truth transmittance computed using ray tracing in a beam of the same footprint. The exponential transmittance often decreases too rapidly for surfaces, which leads to bloating. However, in other scenarios, despite this rapid decrease in transmittance, the exponential model leads to significant leaking, as highlighted in the inset plots, due to its inability to easily reach zero. The non-exponential model fits the transmittance curves more faithfully and prevents leakage.

For our appearance prefiltering results, we implemented a dynamic mip mapping scheme based on ray differentials [257]. Using the ray differentials, we estimate the size of a pixel in world space as we intersect the volume bounding box. We then probabilistically choose one of the two volume resolutions which most closely match the scale of the pixels. For simplicity, we use the same volume resolution for the whole light path. Please see the supplemental video of the paper for animated zoom-ins using this method[1]. The following static images all have been computed at a fixed scale to facilitate comparisons to previous work.

Performance. We evaluate the performance of our methods using both exponential and non-exponential transmittance models. When prefiltering the CITY BUILDING scene (see Figure 7.2) at a resolution of 128^3 voxels, it takes around 1 minute to compute albedo, SGGX parameters and initial extinction values. Optimizing the transmittance takes 4 minutes using an exponential model and around 4.5 minutes using our non-exponential transmittance, which amounts to around 12.5% overhead. Rendering the optimized volume on the CPU is around 35% slower when using our non-exponential model than when using the exponential model. For reference, path tracing the same scene using Embree [146] is around 5× faster than rendering the volumetric representation. This is not surprising, as our volume rendering implementation was only lightly optimized.

[1]The video is available on `http://rgl.epfl.ch/publications/Vicini2021NonExponential`

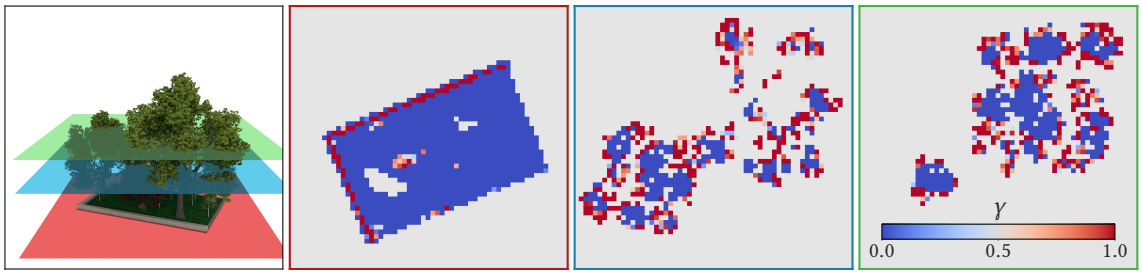


Figure 7.8: We visualize the fitted values for γ over several horizontal slices in the same scene. The voxel grid used here has a resolution of 64^3. These slices show that the optimizer indeed automatically classifies voxels containing opaque geometry to use linear transmittance (blue) and voxels containing more unstructured geometry (e.g. leaves) toward using exponential transmittance (red).

Image-based reconstruction of a volumetric scene representation is significantly more challenging than appearance prefiltering and takes around 1 hour for the results shown in Figure 7.14 (256^3 voxels). The non-exponential model takes around 10% more time to optimize. However, the CUDA implementation could be simplified to support purely exponential media more efficiently.

Exponential vs. non-exponential transmittance. For appearance prefiltering, we show the practical benefits from switching to a non-exponential transmittance in Figure 7.6. We ran our transmittance optimization both for the exponential and our general non-exponential model. For both sets of results, we separate local and global scattering by offsetting the starting location of rays. This is a form of non-exponentiality, but a standard exponential medium would result in energy loss, as shown in Figure 7.3.

We found that the exponential model suffers from significant leaking artifacts. One possible approach to reduce leaking is to more strongly penalize this type of error. Therefore, we experimented with using a relative loss, where the L_1 transmittance loss is divided by the ground truth transmittance (clamped to a small epsilon to prevent division by zero). This increases the loss value if the reference transmittance is zero or close to zero. We found that this helped to reduce leaking, but at the same time drastically increased bloating, especially for semitransparent regions such as the fence in Figure 7.6. This indicates that the exponential model cannot easily be fixed by just changing the optimization routine. This is a trade-off between opaqueness and amount of bloating inherent to the exponential transmittance model. Our more general non-exponential model can find a better compromise between these conflicting goals. We found it to mostly eliminate leaking as well as reducing bloating compared to the exponential model.

In Figure 7.7 we plot the transmittance obtained after optimizing medium param-

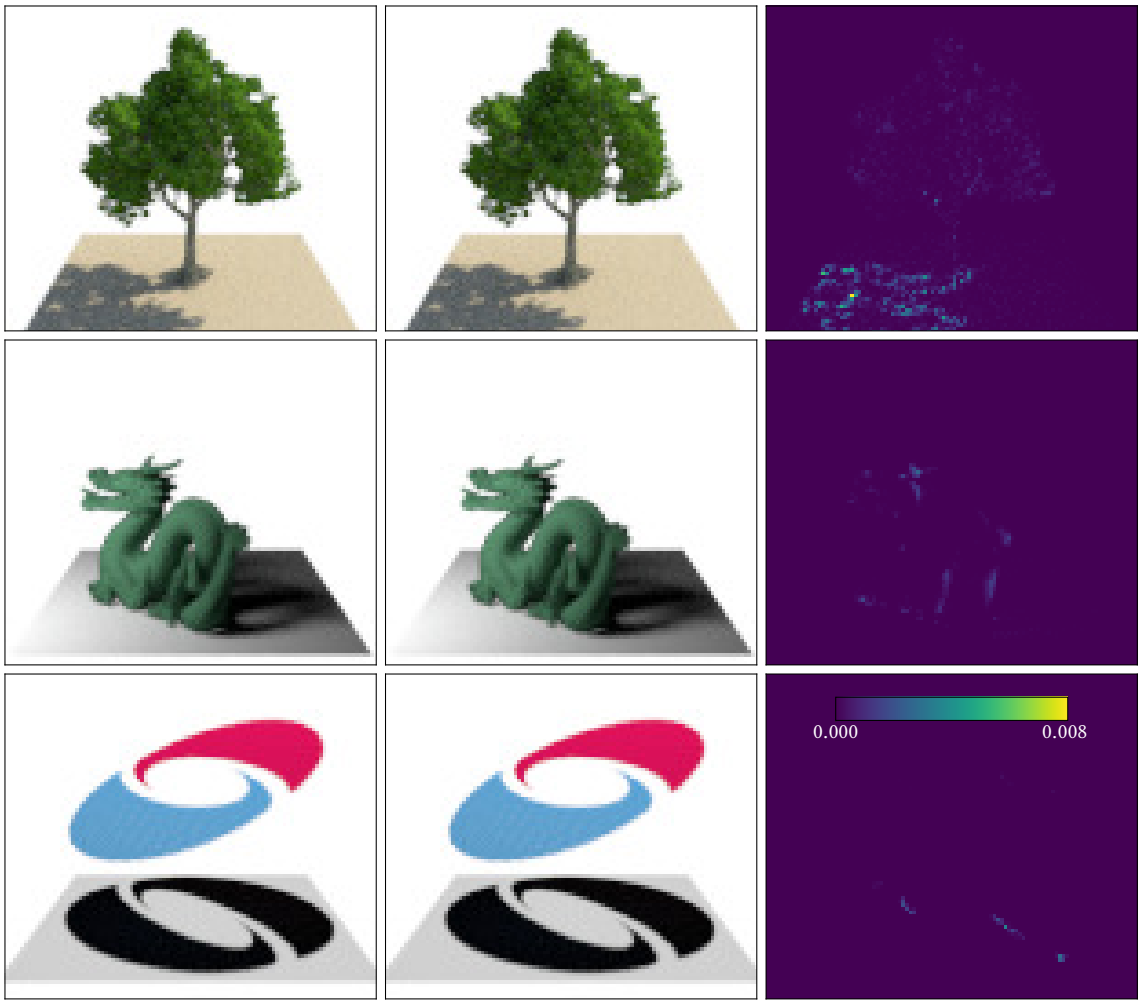
eters, and the corresponding values of γ. These plots demonstrate that the optimizer is able to automatically detect regions with different modes (opaque vs. aggregate uncorrelated) and pick the proper transmittance mode for it by selecting the γ parameter value. It further showcases the issues related to bloating and leaking found in the exponential model. We can also see that both the non-exponential and exponential model are approximations and do not give a perfect match to the reference transmittance.

Figure 7.8 shows slices of the optimized voxel grid of γ values. The optimizer prefers using linear transmittance for solid objects, while it resorts to the classic exponential transmittance for aggregate or unstructured detail, such as leaves toward the outside of the trees.

Reciprocity. Our transmittance model is not reciprocal and the transmittance $T(\mathbf{x}, \mathbf{y})$ might not match $T(\mathbf{y}, \mathbf{x})$. We evaluate the practical impact of this limitation by switching the evaluation direction of the transmittance when tracing shadow rays. We render several volumetric scene representations both using the unmodified path tracer and the implementation with the reversed shadow rays. The results of this experiment are shown in Figure 7.9. The differences caused by non-reciprocal behavior are almost imperceptible. This indicates that the non-reciprocity is not a limiting factor in practice. We provide some additional evaluation in Figure 7.10.

Specular surfaces The problem of finding the right transmittance model is mostly orthogonal to the problem of defining the phase function. We therefore use simple diffuse BSDFs in most of our scenes. However, our prefiltering method can also be applied to scenes with specular surfaces, as we show in Figure 7.11. Our volumetric model reproduces the overall appearance of the golden bunny, but leads to a slight loss in sharpness of the reflection.

Comparison to prior work We compare our prefiltering method to the state-of-the-art Hybrid LoD method [290]. Figure 7.12 shows the comparison on several example scenes. We used the original implementation provided by the authors, which we extended to be able to load Mitsuba 2 scenes. We run both methods at resolutions 16^2 and 64^2 and compare to the ground truth of rendering the reference scene at the same resolution. The Hybrid LoD method produces good results on volume-friendly scenes (i.e., aggregate details), e.g., the Trees scene. However, it often misclassifies complex and mixed types of geometry as a volume (e.g., for the City building and City scenes), which leads to a semitransparent appearance that is far from the ground truth. The



Original | Reversed shadow rays | L_1 error

Figure 7.9: We evaluate the practical impact of our model's non-reciprocity by rendering our volume as usual, but evaluating shadow rays in the reverse direction (second column). The difference to the original rendering using the unmodified path tracer is almost imperceptible. All error maps are scaled consistently and normalized to make the small differences visible.

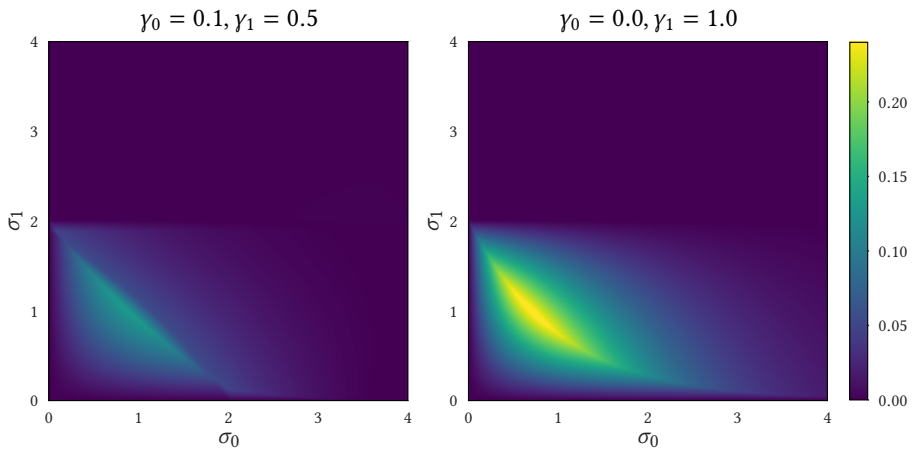


Figure 7.10: We plot the absolute difference between evaluating the transmittance across two voxels in either direction. The error depends on the medium parameters σ_t and γ. We show two plots over the extinction values of the two voxels, each for different settings of transmittance modes. The difference is maximized if the two adjacent voxels have completely opposite transmittance modes ($\gamma_0 = 0$ and $\gamma_1 = 1$), and gets smaller as the difference in transmittance mode decreases. The error goes to zero if both voxels have the same transmittance mode, or both are either fully transparent or fully opaque.

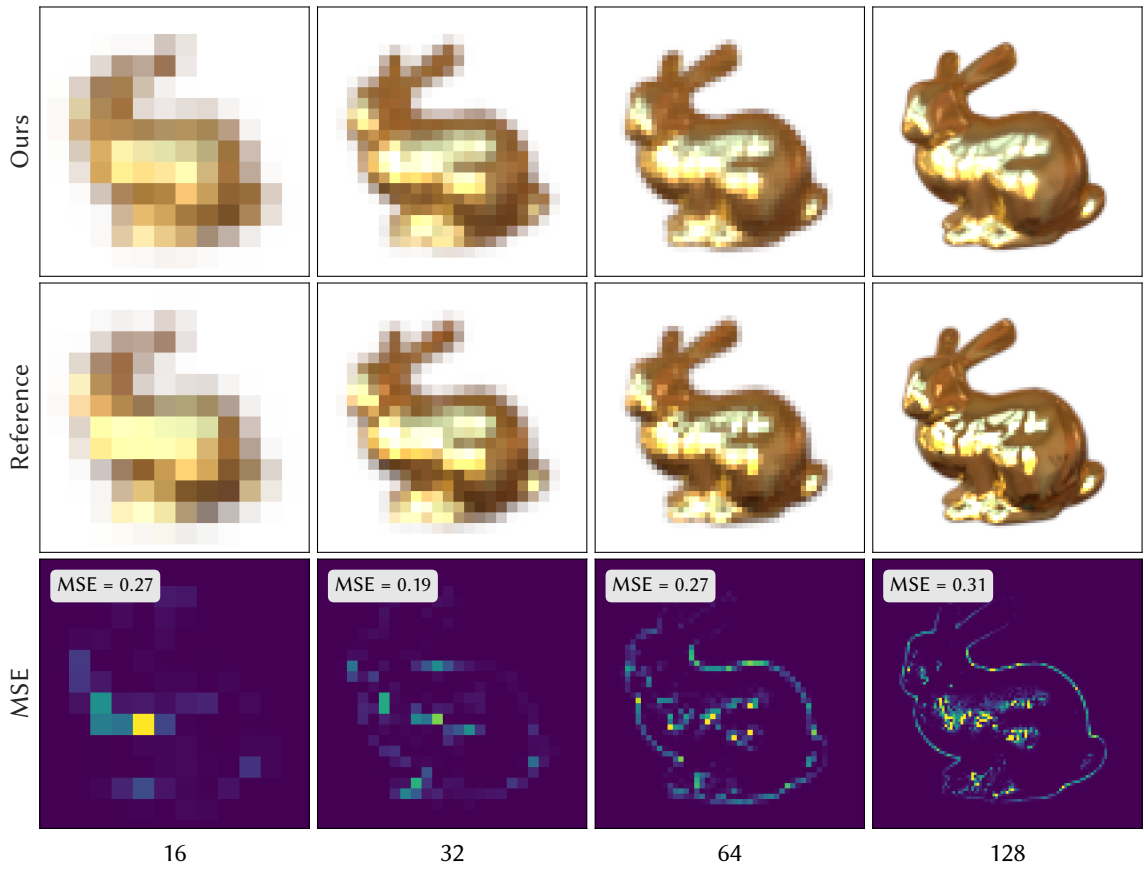


Figure 7.11: By switching the SGGX phase function to use specular microflakes, we can also compute level of detail for metallic objects.

Checkerboards scene also shows how this behavior is scale-dependent. At the higher resolution, it correctly classifies all surfaces as surfaces. However, as the resolution is decreased it switches to a volumetric representation, which changes the appearance of the object drastically. Our method does not perform a binary classification and therefore can maintain a consistent approximation quality across scales. The optimizer automatically lands at the best transmittance mode, also including the spectrum between linear and exponential transmittance modes. We show a higher resolution rendering of an LoD volume computed using our method compared to Hybrid LoD to show the issues of this binary classification in Figure 7.13. The binary classification results in drastic changes of appearance across a single LoD.

Application to image-based reconstruction As described in Section 7.5, we can also apply our non-exponential transmittance model to improve image-based reconstruction using differentiable path tracing. In Figure 7.14, we show reconstructions of several example scenes using an exponential and our non-exponential representation. All results use a voxel grid resolution of 256^3, which means that we optimize ca. 184 million parameters. However, since a lot of the voxels are empty, the optimization problem remains tractable. We optimize for 64 views of the synthetic scene at once.

Both the exponential and our non-exponential model converge to a meaningful volumetric approximation of the reference scene. However, the exponential model again suffers from leaking, despite the explicit silhouette loss. This is primarily visible in thin structures. At the same time, this exponential model also suffers from more bloating than our non-exponential model, as shown well in the details of the Lego scene. The exponential model fills in parts of the beams where holes should be present, while leading to leaking in other parts. This shows that using an exponential transmittance model is not sufficient when trying to relight a volumetric scene reconstruction, which is consistent with our observations made for the level of detail use case. Overall, we observe that the improvement on image-based reconstruction is smaller than for appearance prefiltering. Image-based optimization using global illumination is a harder problem than fitting transmittance parameters against a known 3D scene. Additionally, the image-based optimization does not enforce *sparsity* of the volume density field. The lack of sparsity constraints makes the results less sensitive to the transmittance model than for appearance prefiltering, where we explicitly enforce sparsity.

Application to neural radiance fields We also ran some experiments where we modified the NeRF code base [223] to use a non-exponential transmittance. The re-

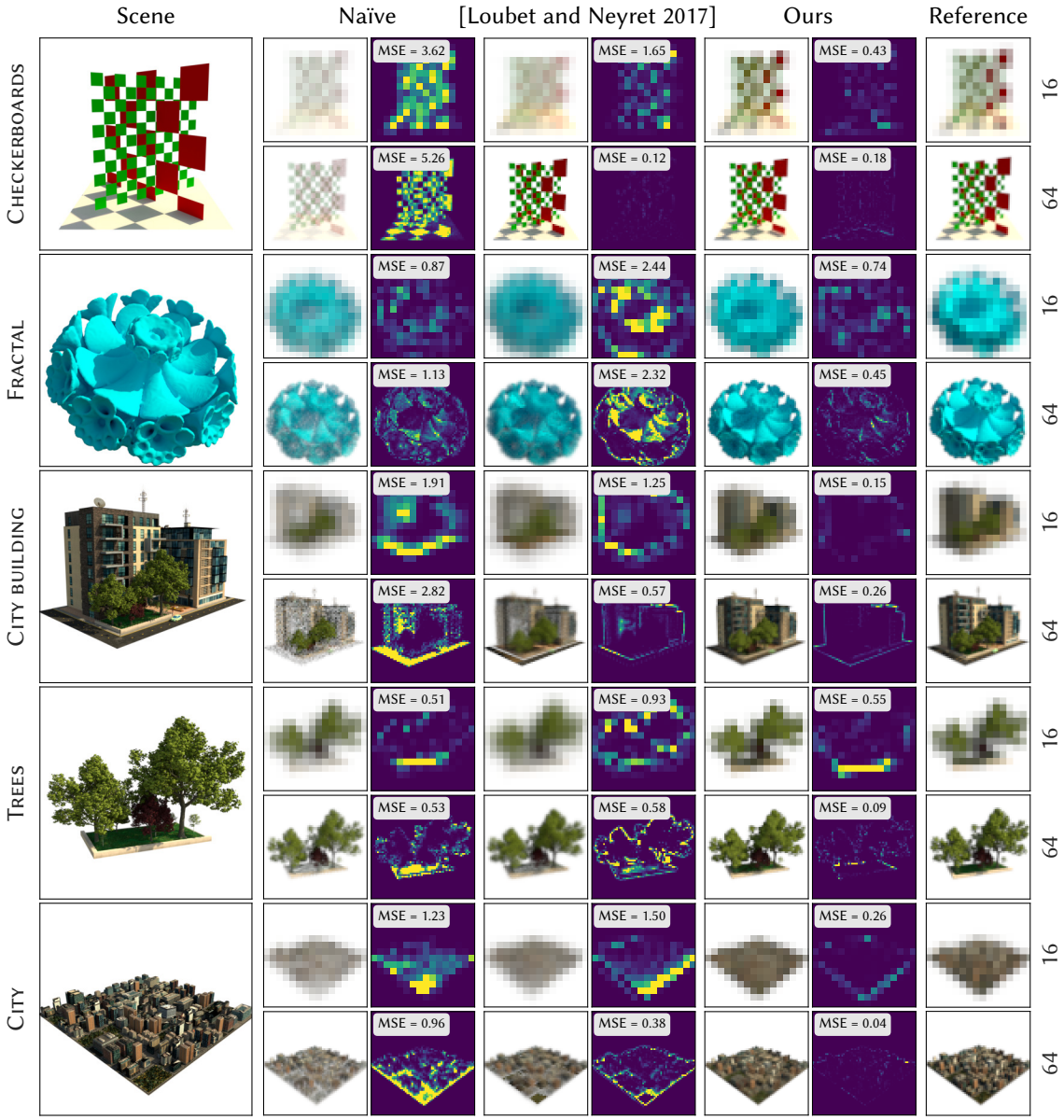
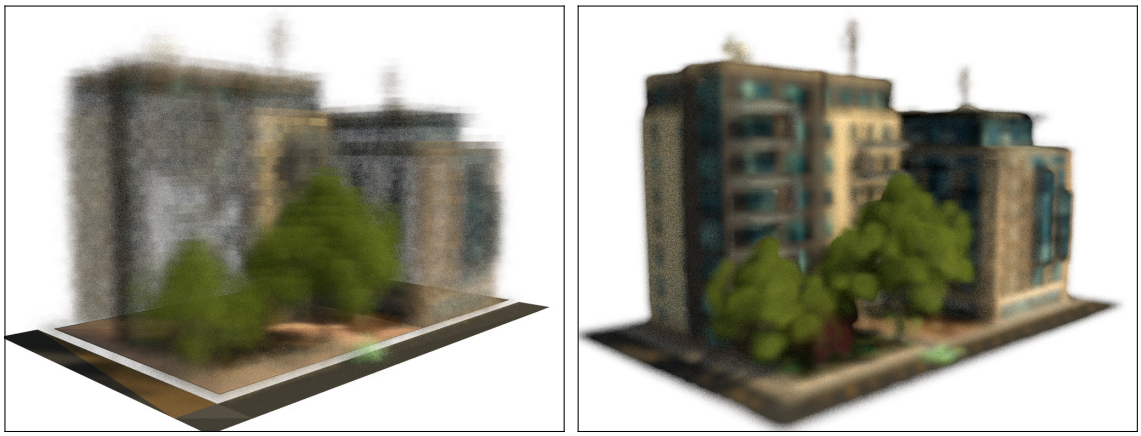


Figure 7.12: Prefiltering results on a variety of scenes with different complexity. *Reference* is the path traced ground truth geometry. The reference is rendered using a Gaussian pixel filter to bandlimit the signal. *Naïve* is a naïve volumetric approximation of the scene. This is the starting point for our optimization. All LoD methods use a voxel resolution which matches the resolution of the rendered images. *Loubet and Neyret 2017* is the previous state of the art in automatic volumetric level of detail and *Ours* is our non-exponential model. All results are rendered using 1024 samples per pixel and we visualize mean squared errors (MSE) compared to the reference image. All MSE values have been multiplied by a factor of 100 for readability.



Hybrid LoD Ours

Figure 7.13: If we render low-resolution LoD volumes at a higher pixel resolution, we can see that the Hybrid LoD method by Loubet and Neyret [290] struggles to correctly separate surfaces and volumes. Our method results in a smoother model, as it does not apply a binary classification between surface-like and volume-like scattering.

sults of this are shown in Figure 7.15. Using a non-exponential transmittance, we can achieve some improvements in the sharpness of the generated images. On the Red Plane scene, the unmodified NeRF model struggles to represent the vertical plane. Our non-exponential model manages to produce more accurate results. However, since the NeRF model does not admit relighting, the exponential transmittance does not suffer from the severe light leaking issues present in the other applications. The fixed lighting and directionally varying emission component seem to make NeRF less sensitive to the accuracy of the transmittance model. The combination of non-exponential transmittance with neural networks to solve real computer vision problems remains an interesting future direction.

7.7 Summary and future work

In this chapter, we introduced a new volumetric transmittance model, which represents both surface-like and volumetric appearance in a unified framework. Our model does not require explicit binary classification and we show that its parameters can be optimized efficiently using gradient-based optimization. We further show improvements to image-based reconstruction using differentiable rendering. Our model can easily be integrated into existing systems and is simple to implement.

One of the main practical limitations is the representation power of the phase function. The SGGX phase function [91] can represent diffuse and metallic surfaces, but

Figure 7.14: Results of reconstructing scenes using an image-based optimization. We compare our non-exponential transmittance model with the exponential model and the ground truth reference. All volumes use a resolution of 256^3 and we use the same resolution to render the resulting images.

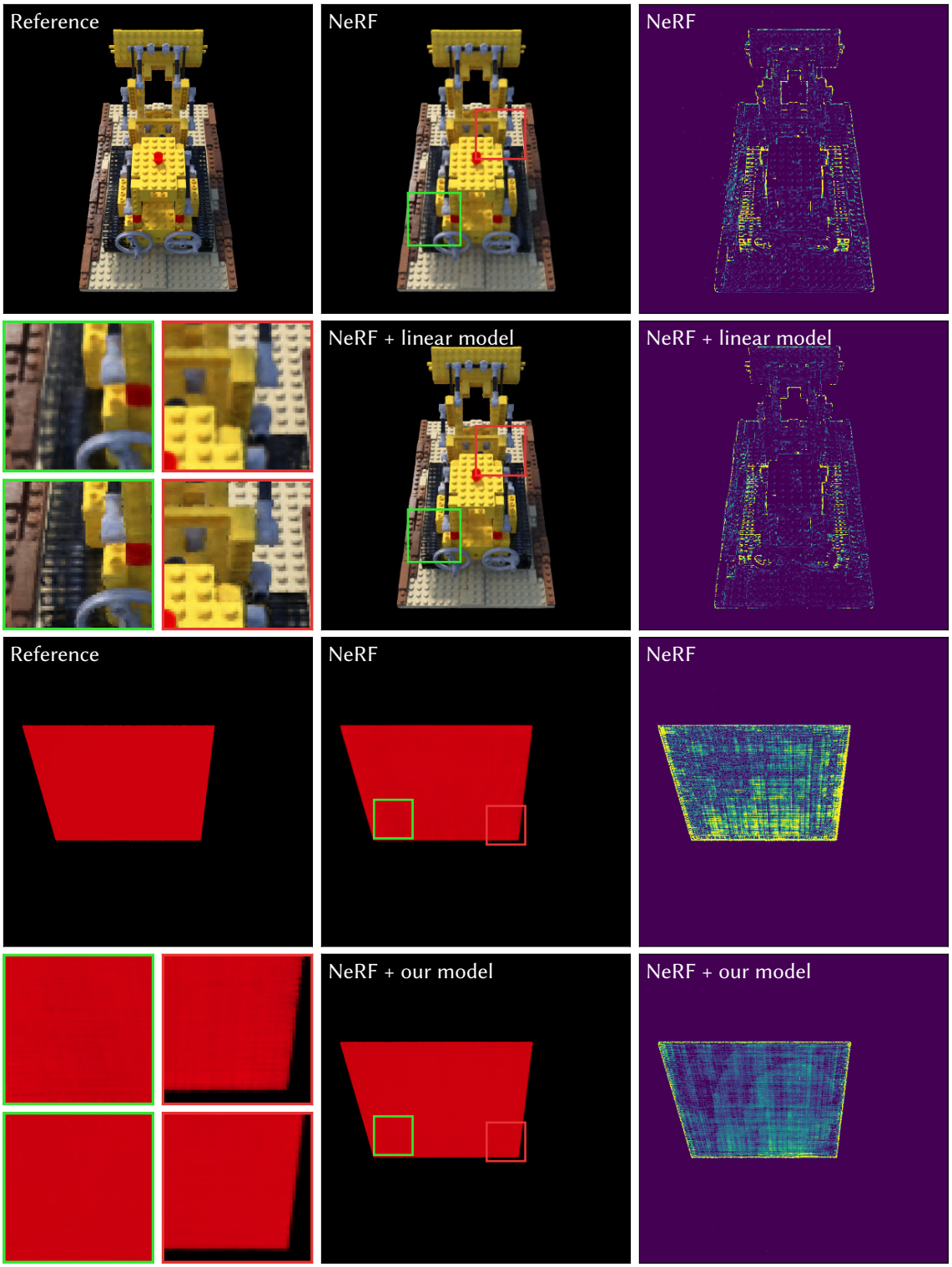


Figure 7.15: Neural volumetric representation of the Lego scene (top) using NeRF [223] with the original exponential transmittance model and with a purely linear model. We also applied exponential NeRF and a version using our transmittance model to the Red plane scene (bottom).

more work is required to define more general phase functions to handle plastic-like or dielectric materials. The phase function further serves an important role in preventing leaking due to multiple scattering. With the SGGX phase functions, more sampled directions will point into the surface as the roughness of the normal distribution is increased. This can result in leaking of multiple scattering on the back of a surface and energy loss on the front side. Dupuy et al. [285] discussed the idea of approximating surface appearance using one-sided microflakes, but it is an open question if that concept could be used for scene representation.

While we found our transmittance formulation to work well for the shown applications, it is not constrained to be reciprocal. Developing a reciprocal formulation would make the model applicable in the context of bidirectional rendering algorithms. It would also be interesting to investigate if the transmittance could be sampled and evaluated in an unbiased way, instead of using biased ray marching. For evaluation, recent work on de-biasing Monte Carlo estimators has demonstrated unbiased evaluation for certain non-exponential transmittance models [53].

Overall, we hope that this work enables future research to go beyond standard exponential transmittance models. We believe that this is a necessary step toward further unification of surface and volume rendering. With the gain in popularity of differentiable rendering, unifying these two worlds can benefit practical scene reconstruction approaches.

8 | Conclusion

We conclude this thesis by summarizing our main contributions and providing an outlook on directions for future work.

8.1 Contributions

Path replay backpropagation. In Chapter 5, we presented a novel differentiation method for path tracing and similar algorithms. Instead of storing a large computation graph, we exploit the local invertibility of the Jacobian of each iteration in the main path tracing loop, which enables computing unbiased gradients in linear time and at constant memory use. This is a crucial building block for the differentiation of complex light transport. No prior method was able to scale to the full complexity posed by rendering algorithms such as delta tracking. Furthermore, we demonstrated the generality of our approach by proposing a version for perfectly specular surfaces. While developed for differentiable rendering, our approach can also be used for other applications such as inverse problems specified using partial differential equations [44].

Differentiable SDF rendering. We proposed the first general differentiable SDF rendering method in Chapter 6. Unlike many prior works, it does not assume knowledge about object silhouettes or masks. By constructing a reparameterization during sphere tracing, we can compute accurate gradients of both directly visible edges and indirect effects (e.g., shadows). Since we carefully design our reparameterization to not build an AD graph over sphere tracing iterations, this algorithm again does not require storing a large computation graph. We show that the gradients estimated using our reparameterization enable image-based shape reconstruction. Our method is both faster and more accurate than the previous state-of-the-art reparameterization approach.

Non-exponential scene representation. In the last project (Chapter 7), we demonstrate that we can more accurately represent general scenes as participating media by switching to a non-exponential transmittance model. This benefits both prefiltering for level of detail and image-based reconstruction. This project shows that these problems are closely related, and can be investigated jointly. Similarly to image-based reconstruction, the prefiltering problem can be addressed using gradient-based optimization techniques. Converting an existing scene to a volumetric representation is an easier problem

than the full image-based reconstruction. It can be useful to illustrate the potential improvements due to changes in the scene representation. The non-exponential model that we propose successfully eliminates most light leaking problems faced by the conventional, exponential model.

Overall, our methods expand the scope and applicability of differentiable rendering. We can differentiate important transport phenomena efficiently and accurately, which will hopefully enable future applications.

8.2 Open problems and future work

Recently, a lot of progress has been made on differentiable rendering methods (see also Chapter 4). However, there are still many open problems and potential future directions. In this section, we outline some of the main open problems.

Scene and object reconstruction. A key application of differentiable rendering is scene reconstruction. While many methods have been proposed, we are still unable to fully recover a relightable representation of an arbitrary scene. Refractive objects for example still break most state-of-the-art algorithms. Ideally, one would develop a unified, physically-based representation that can be optimized while considering global illumination. While volumetric representations are promising, it is currently unclear how to best extend them to handle the large variety of real-world materials (e.g., layered BSDFs). A potential avenue would be to use a participating medium with a learned neural phase function. However, it seems difficult to accurately model high-frequency aspects of light transport (e.g, perfectly specular BSDFs) using such a representation. The problem is also inherently riddled with ambiguities. For example, a flat mirror on an interior wall could also be an opening to a symmetrical room.

To ensure some level of convexity, the optimization probably needs to progress in a multi-stage fashion, where each stage considers additional light transport properties. Neural radiance fields [223] impressively demonstrate how optimizing a purely emissive representation is comparably easy. In hindsight, this is not surprising: traditional photogrammetry algorithms can be understood as computing an emissive representation, as they simply project captured images onto reconstructed geometry. They do not consider shadows, interreflections and BSDF models.

At a high level, it is also unclear what quality of representation we can expect to recover from using only a single illumination condition, as done when casually capturing

multiple photos of a real scene. The robust reconstruction of a general, physically-based representation likely requires at least some form of active illumination (e.g., a camera flash). Including multiple illumination conditions into a differentiable rendering optimization is straightforward, but this has not yet been done at scene scale. Ambiguities could also be addressed by constructing suitable priors, e.g., by training a neural representation on a large data set of scenes. The main challenge in such approaches is the generalization to unknown objects or scenes.

Efficiency. The methods in this thesis are all designed to be computationally efficient. Most of the featured experiments ran in a few minutes or at most a few hours. However, further efficiency gains are likely possible. Our optimizations all build on a basic unidirectional path tracer, without any advanced variance reduction techniques. Many efficient forward rendering algorithms could potentially be adapted to the differentiable rendering context. With forward path tracing of certain scenes already running at interactive frame rates, differentiable rendering could possibly be optimized to run similarly fast.

In particular, methods that re-use information across multiple renderings could be useful. We could for example use a path guiding data structure that is refined across iterations. Other options are temporal denoising algorithms, adaptive sampling or control variates. Any technique reducing the variance of the rendering itself could help reduce the number of samples needed during an optimization. Gradient estimators often have a higher variance than their primal versions, and developing targeted variance reduction schemes for differentiable Monte Carlo estimators is still largely an open problem.

System. The Mitsuba 3 and Dr.Jit systems appear to be in a good local minimum for the development of differentiable rendering algorithms. The combination of uninterrupted tracing, megakernel compilation and fine-grained control over AD has proven useful. Long term, the system could benefit from aggressively caching JIT tracing. Currently, the performance overhead of tracing long Python functions can be significant. Eliminating this could reduce the need to partially implement methods in C++ simply to reduce tracing costs. It could also enable more sophisticated optimization passes, which are too slow to be run on every rendering. The system could also be extended to support higher-order derivatives. These would for example be useful to automatically compute reparameterization Jacobians (i.e., positional derivatives), which are then further differentiated with respect to the scene parameters.

Applications to other fields. As mentioned in the introduction, differentiable Monte Carlo estimators could benefit a range of practical problems outside of computer graphics. It would be interesting to import some of the tools developed in differentiable rendering into other fields. One example are PDEs that admit a solution using Monte Carlo methods, but we expect these methods to also benefit important problems in computational design, (medical) imaging and atmospheric sciences. More research is required to fully understand the potential applications as well as application-specific priors and regularizations that are needed to tackle ill-posed inverse problems. We are hopeful that differentiable Monte Carlo methods will become increasingly useful for a broad range of problems.

A | Equivalence of area element and divergence derivatives

In the following, we prove that we can either evaluate the area element (using the cross product) or the divergence of the mapping computed in ambient space. Under differentiation with respect to the scene parameter π, these are equivalent. Since we reparameterize the manifold of the unit sphere, we found this is to be not immediately obvious. We would like to show:

$$\partial_\pi \left\| \mathrm{D}\mathcal{T}_{\boldsymbol{\omega},\pi}(\mathbf{s}) \times \mathrm{D}\mathcal{T}_{\boldsymbol{\omega},\pi}(\mathbf{t}) \right\| = \partial_\pi \mathrm{div}\, \mathcal{T}(\boldsymbol{\omega}, \pi). \tag{A.1}$$

The differential $\mathrm{D}\mathcal{T}_{\boldsymbol{\omega},\pi}$ is computed in ambient space and in the following we write it as a parameter dependent Jacobian matrix $\mathbf{J}(\pi)$. We then prove the original statement by simplifying the derivative of the area element:

$$\begin{aligned} \partial_\pi \left\| \mathbf{J}(\pi)\mathbf{s} \times \mathbf{J}(\pi)\mathbf{t} \right\| &= \boldsymbol{\omega} \cdot \left(\partial_\pi \mathbf{J}(\pi)\mathbf{s} \times \mathbf{J}(\pi)\mathbf{t} + \mathbf{J}(\pi)\mathbf{s} \times \partial_\pi \mathbf{J}(\pi)\mathbf{t} \right) \\ &= \boldsymbol{\omega} \cdot \left(\partial_\pi \mathbf{J}(\pi)\mathbf{s} \times \mathbf{t} + \mathbf{s} \times \partial_\pi \mathbf{J}(\pi)\mathbf{t} \right), \end{aligned}$$

where we differentiated the vector norm and moved the derivative operator inside the cross product and "$\cdot$" denotes the dot product. We also used that $\mathbf{J}(\pi)$ maps $\mathbf{s}$ and $\mathbf{t}$ onto themselves and therefore $\mathbf{J}(\pi)\mathbf{s} \times \mathbf{J}(\pi)\mathbf{t} = \boldsymbol{\omega}$. We simplify further by using the fact that the triple product $\mathbf{a} \cdot (\mathbf{b} \times \mathbf{c})$ is invariant under circular shifts of its arguments:

$$\begin{aligned} \boldsymbol{\omega} \cdot \left(\partial_\pi \mathbf{J}(\pi)\mathbf{s} \times \mathbf{t} + \mathbf{s} \times \partial_\pi \mathbf{J}(\pi)\mathbf{t} \right) &= \partial_\pi \mathbf{J}(\pi)\mathbf{s} \cdot \mathbf{t} \times \boldsymbol{\omega} + \partial_\pi \mathbf{J}(\pi)\mathbf{t} \cdot \boldsymbol{\omega} \times \mathbf{s} \\ &= \partial_\pi \mathbf{J}(\pi)\mathbf{s} \cdot \mathbf{s} + \partial_\pi \mathbf{J}(\pi)\mathbf{t} \cdot \mathbf{t}, \end{aligned}$$

where we additionally used that $\mathbf{t} \times \boldsymbol{\omega} = \mathbf{s}$ and $\boldsymbol{\omega} \times \mathbf{s} = \mathbf{t}$. We then add the zero-valued term $\partial_\pi \mathbf{J}(\pi)\boldsymbol{\omega} \cdot \boldsymbol{\omega}$ to the expression above and use that the sum of inner products of basis vectors with $\partial_\pi \mathbf{J}(\pi)$ is exactly its trace:

$$\begin{aligned} &= \partial_\pi \mathbf{J}(\pi)\mathbf{s} \cdot \mathbf{s} + \partial_\pi \mathbf{J}(\pi)\mathbf{t} \cdot \mathbf{t} + \partial_\pi \mathbf{J}(\pi)\boldsymbol{\omega} \cdot \boldsymbol{\omega} \\ &= \mathrm{T}(\partial_\pi \mathbf{J}(\pi)) = \partial_\pi \mathrm{T}(\mathbf{J}(\pi)) = \partial_\pi \mathrm{div}\, \mathcal{T}(\boldsymbol{\omega}, \pi). \end{aligned}$$

With this, Equation A.1 has been proven. This makes the equivalence of reparameterization and divergence formulation explicit and shows that in practice either formulation yields the same result. Both formulations correctly account for the geometry of the unit sphere.

To complete the proof, we still need to show the following:

$$\partial_\pi \mathbf{J}(\pi)\boldsymbol{\omega} \cdot \boldsymbol{\omega} = 0. \tag{A.2}$$

We do so assuming that we can write $\mathcal{T}(\boldsymbol{\omega}, \pi) = \bar{\mathcal{T}}(\boldsymbol{\omega}, \pi)/\left\|\bar{\mathcal{T}}(\boldsymbol{\omega}, \pi)\right\|$, where $\bar{\mathcal{T}}(\boldsymbol{\omega}, \pi)$ might not have unit norm, but preserves the direction, i.e. $\bar{\mathcal{T}}(\boldsymbol{\omega}, \pi) = \lambda\boldsymbol{\omega}$. Our reparameterization keeps the primal value of $\boldsymbol{\omega}$ fixed, but there can still be an arbitrarily complex differentiable relation to the parameter π. We then explicitly compute the parameter derivative of the Jacobian matrix $\mathbf{J}(\pi) = \partial_{\boldsymbol{\omega}}\mathcal{T}$:

$$\begin{aligned}
\partial_\pi \mathbf{J}(\pi) &= \partial_\pi \left[\partial_{\boldsymbol{\omega}} \left(\frac{\bar{\mathcal{T}}(\boldsymbol{\omega}, \pi)}{\left\|\bar{\mathcal{T}}(\boldsymbol{\omega}, \pi)\right\|} \right) \right] = \partial_\pi \left[\left(\frac{1}{\left\|\bar{\mathcal{T}}\right\|}\mathbb{I} - \frac{1}{\left\|\bar{\mathcal{T}}\right\|^3}\bar{\mathcal{T}}\bar{\mathcal{T}}^T \right) \partial_{\boldsymbol{\omega}}\bar{\mathcal{T}} \right] \\
&= \underbrace{\left(\frac{1}{\left\|\bar{\mathcal{T}}\right\|}\mathbb{I} - \frac{1}{\left\|\bar{\mathcal{T}}\right\|^3}\bar{\mathcal{T}}\bar{\mathcal{T}}^T \right) \partial_\pi \partial_{\boldsymbol{\omega}}\bar{\mathcal{T}}}_{(1)} + \underbrace{\partial_\pi \left[\frac{1}{\left\|\bar{\mathcal{T}}\right\|}\mathbb{I} - \frac{1}{\left\|\bar{\mathcal{T}}\right\|^3}\bar{\mathcal{T}}\bar{\mathcal{T}}^T \right] \partial_{\boldsymbol{\omega}}\bar{\mathcal{T}}^T}_{(2)},
\end{aligned}$$

where $\mathbb{I}$ is the 3 by 3 identity matrix and $\bar{\mathcal{T}}\bar{\mathcal{T}}^T$ is an outer product. We omit the arguments of $\bar{\mathcal{T}}(\boldsymbol{\omega}, \pi)$ for brevity. In order for Equation A.2 to hold, we need to show that multiplying $\boldsymbol{\omega}$ with this matrix is a projection into tangent space of the unit sphere at $\boldsymbol{\omega}$. For the term (1) this follows immediately, since the terms inside the parentheses are such a projection (recall that $\bar{\mathcal{T}}$ is simply a scaled $\boldsymbol{\omega}$). For (2) this is a bit more laborious to show, but can be done by computing the derivative of the terms inside the brackets:

$$\begin{aligned}
\partial_\pi \left[\frac{1}{\left\|\bar{\mathcal{T}}\right\|}\mathbb{I} - \frac{1}{\left\|\bar{\mathcal{T}}\right\|^3}\bar{\mathcal{T}}\bar{\mathcal{T}}^T \right] = -\frac{1}{\left\|\bar{\mathcal{T}}\right\|^3} \Bigg(& \underbrace{\bar{\mathcal{T}}^T \partial_\pi \bar{\mathcal{T}} \left(\mathbb{I} - \frac{1}{\left\|\bar{\mathcal{T}}\right\|^2}\bar{\mathcal{T}}\bar{\mathcal{T}}^T \right)}_{(1)} \\
& + \underbrace{\partial_\pi \bar{\mathcal{T}}\bar{\mathcal{T}}^T - \frac{\bar{\mathcal{T}}^T \partial_\pi \bar{\mathcal{T}}}{\left\|\bar{\mathcal{T}}\right\|^2}\bar{\mathcal{T}}\bar{\mathcal{T}}^T}_{(2)} + \underbrace{\bar{\mathcal{T}}\partial_\pi \bar{\mathcal{T}}^T - \frac{\bar{\mathcal{T}}^T \partial_\pi \bar{\mathcal{T}}}{\left\|\bar{\mathcal{T}}\right\|^2}\bar{\mathcal{T}}\bar{\mathcal{T}}^T}_{(3)} \Bigg)
\end{aligned}$$

Here, we can again see that (1) is a projection (scalar factors do not matter), and (2) and (3) require a few additional steps. We can simplify (2) to again see that it indeed projects onto tangent space:

$$\begin{aligned}
\partial_\pi \bar{\mathcal{T}}\bar{\mathcal{T}}^T - \frac{\bar{\mathcal{T}}^T \partial_\pi \bar{\mathcal{T}}}{\left\|\bar{\mathcal{T}}\right\|^2}\bar{\mathcal{T}}\bar{\mathcal{T}}^T &= \left(\partial_\pi \bar{\mathcal{T}} - \frac{\bar{\mathcal{T}}^T \partial_\pi \bar{\mathcal{T}}}{\left\|\bar{\mathcal{T}}\right\|^2}\bar{\mathcal{T}} \right) \bar{\mathcal{T}}^T \\
&= \left(\mathbb{I} - \frac{1}{\left\|\bar{\mathcal{T}}\right\|^2}\bar{\mathcal{T}}\bar{\mathcal{T}}^T \right) \partial_\pi \bar{\mathcal{T}}\bar{\mathcal{T}}^T.
\end{aligned}$$

The second equality used that $\bar{\mathcal{T}}^T \partial_\pi \bar{\mathcal{T}}$ is a scalar to commute it with $\bar{\mathcal{T}}$. We skip the derivation for the term (3), as it is analogous. This shows that Equation A.2 indeed holds.

B | Probabilistic regularization of path replay backpropagation

The adjoint phase of our attached PRB estimator requires an unbiased estimate of the directional derivative of the incident radiance at every path vertex. Regularization is necessary to avoid situations in which the associated computation becomes ill-defined. We now justify that this regularization of the Jacobian and the associated inversion of noisy matrices do not introduce bias.

During the adjoint phase, we must undo the effects of the path prefix to obtain the directional derivative at the *current* path vertex. This is in part done by subtracting matrices, which is unproblematic. However, the second part is more involved: we must multiply the directional radiance derivative by the inverse matrix $\mathbf{J}_{\text{ray}}^{-1}$. The matrix $\mathbf{J}_{\text{ray}} = \mathbf{A}_1 \cdot \mathbf{A}_2 \cdots \mathbf{A}_k$ is the product of all the local ray derivatives $\mathbf{A}_i$ encountered in the path prefix. The challenge is now that some of these matrices may not be invertible (e.g., when the associated vertex has a diffuse BRDF). The subsequent derivations will prove two claims:

1. Adding noise to factors of the Jacobian product does not affect its expected value.
2. This noise can be added in a way so that even an estimator based on matrix inverses remains unbiased.

Regarding claim (1), we must show that

$$\mathbb{E}_{\mathbf{J}_{\text{ray}}}[=]\,\mathbb{E}\left[\mathbf{A}_1 \cdot \mathbf{A}_2 \cdots \mathbf{A}_k\right] = \mathbb{E}\left[(\mathbf{A}_1 + \varepsilon_1\mathbf{I}) \cdot (\mathbf{A}_2 + \varepsilon_2\mathbf{I}) \cdots (\mathbf{A}_k + \varepsilon_k\mathbf{I})\right] \tag{B.1}$$

where $\mathbf{A}_1, \ldots, \mathbf{A}_k$ are (fixed) matrices, and $\varepsilon_1, \ldots, \varepsilon_k \in \mathbb{R}$ are i.i.d. random variables with an expected value of 0. The equality follows directly from the linearity of the expected value and the statistical independence of the ε_k. Without loss of generality, we can consider the case of $k = 2$:

$$\mathbb{E}\left[(\mathbf{A}_1 + \varepsilon_1\mathbf{I})(\mathbf{A}_2 + \varepsilon_2\mathbf{I})\right] = \mathbb{E}\left[\mathbf{A}_1\mathbf{A}_2\right] + \underbrace{\mathbb{E}\left[\varepsilon_1\mathbf{I}\mathbf{A}_2\right]}_{=0} + \underbrace{\mathbb{E}\left[\mathbf{A}_1\varepsilon_2\mathbf{I}\right]}_{=0} + \underbrace{\mathbb{E}\left[\varepsilon_1\varepsilon_2\mathbf{I}\right]}_{=0} = \mathbb{E}\left[\mathbf{A}_1\mathbf{A}_2\right] \quad \square \tag{B.2}$$

The first equality uses the linearity of the expected value. We then see that all terms except for the first one become zero due since $\mathbb{E}_{\varepsilon_k}[=]\,0$.

Now for claim 2, we need to show that we can invert the prefix of that product without introducing bias. This is possible by using the same noise both during the forward accumulation and the backward pass. We can then invert the *same* matrices as during the forward pass. Under the assumption that the noisy matrices are then indeed invertible, the computation of the relevant Jacobian at bounce i can be written as:

$$\begin{aligned}
&\mathbb{E}\Big[\underbrace{[(\mathbf{A}_1+\varepsilon_1\mathbf{I})\cdots(\mathbf{A}_i+\varepsilon_i\mathbf{I})]^{-1}}_{\text{Path prefix}}(\mathbf{A}_1+\varepsilon_1\mathbf{I})\cdots(\mathbf{A}_i+\varepsilon_i\mathbf{I})\cdot(\mathbf{A}_{i+1}+\varepsilon_{i+1}\mathbf{I})\cdots(\mathbf{A}_k+\varepsilon_k\mathbf{I})\Big] \\
&= \mathbb{E}\Big[\cancel{[(\mathbf{A}_1+\varepsilon_1\mathbf{I})\cdots(\mathbf{A}_i+\varepsilon_i\mathbf{I})]^{-1}}\,\cancel{(\mathbf{A}_1+\varepsilon_1\mathbf{I})\cdots(\mathbf{A}_i+\varepsilon_i\mathbf{I})}\cdot(\mathbf{A}_{i+1}+\varepsilon_{i+1}\mathbf{I})\cdots(\mathbf{A}_k+\varepsilon_k\mathbf{I})\Big] \\
&= \mathbb{E}\left[(\mathbf{A}_{i+1}+\varepsilon_{i+1}\mathbf{I})\cdots(\mathbf{A}_k+\varepsilon_k\mathbf{I})\right] \qquad \square
\end{aligned}$$

We see that we do *not* take the expected value of a matrix inverse, but the accumulated gradient sample values only ever include regular matrix products and additive noise. Therefore, the regularization in itself does not introduce bias. However, our system of adding uniform noise to every matrix is simplistic, and better strategies may be needed to guarantee good numerical conditioning.

C | Derivations for differentiable SDF rendering

C.1 Ray intersection gradient

When computing a ray intersection with an implicit surface, we solve a root-finding problem over the intersection distance t. An important quantity of interest is the parameter gradient $\partial_\pi t$ of the intersection distance. It turns out that we can readily compute this quantity using the inverse function theorem [207, 208].

We define $f(t, \pi)$ to be the value of the implicit function at a distance t along a fixed ray: $f(t, \pi) = \phi(\mathbf{x}_t, \pi)$ and $f^{-1}(s, \pi)$ its inverse with respect to the first argument. The intersection distance t satisfies $f(t, \pi) = 0$ and can therefore be written as $t = f^{-1}(0, \pi)$. We can then expand using the chain rule:

$$\begin{aligned} 0 = \frac{d}{d\pi} t &= \frac{d}{d\pi}\left[f^{-1}(f(t,\pi),\pi)\right] \\ &= \frac{\partial}{\partial \pi} f^{-1}(f(t,\pi_0),\pi) + \frac{\partial}{\partial \pi} f(t,\pi) \frac{\partial}{\partial s} f^{-1}(f(t,\pi_0),\pi_0). \end{aligned} \tag{C.1}$$

We first use the standard rule for derivatives of inverse functions to see that:

$$\frac{\partial}{\partial s} f^{-1}(f(t,\pi_0),\pi_0) = \frac{1}{\frac{\partial}{\partial t} f(t,\pi_0)}. \tag{C.2}$$

We therefore get:

$$\frac{\partial}{\partial \pi} f^{-1}(f(t,\pi_0),\pi) = -\frac{\frac{\partial}{\partial \pi} f(t,\pi)}{\frac{\partial}{\partial t} f(t,\pi_0)}. \tag{C.3}$$

The denominator is simply equal to $\partial_t f(t, \pi_0) = \langle \phi_{\mathbf{x}}, \mathbf{d} \rangle$, where $\mathbf{d}$ is the ray direction and $\phi_{\mathbf{x}}$ the spatial derivative of the implicit function. So finally, the derivative of the intersection distance with respect to the parameter π becomes:

$$\frac{\partial}{\partial \pi} t(\pi) = -\frac{\frac{\partial}{\partial \pi} \phi(\mathbf{x}_t, \pi_0)}{\langle \phi_{\mathbf{x}}(\mathbf{x}_t, \pi_0), \mathbf{d} \rangle}. \tag{C.4}$$

This derivation works for any sufficiently smooth implicit surface, and in particular also holds for signed distance functions.

C.2 Parameter derivative of the reparameterization $\mathcal{T}$

The parameter derivative of the reparameterization $\mathcal{T}$ should follow the motion of the visible surface boundary on the unit sphere. We validate this by explicitly computing this derivative. In practice, the following computation will be carried out by automatic differentiation. Taking the derivative $\partial_\pi \mathcal{T}$ of our reparameterization we obtain:

$$\begin{aligned}
\partial_\pi \mathcal{T}(\boldsymbol{\omega}, \pi) &= \partial_\pi \left[\frac{\bar{\mathcal{T}}(\boldsymbol{\omega}, \pi)}{\left\| \bar{\mathcal{T}}(\boldsymbol{\omega}, \pi) \right\|} \right] \\
&= \left(\frac{1}{\left\| \bar{\mathcal{T}}(\boldsymbol{\omega}, \pi_0) \right\|} \mathbb{I} - \frac{1}{\left\| \bar{\mathcal{T}}(\boldsymbol{\omega}, \pi_0) \right\|^3} \bar{\mathcal{T}}(\boldsymbol{\omega}, \pi_0) \cdot \bar{\mathcal{T}}(\boldsymbol{\omega}, \pi_0)^T \right) \partial_\pi \bar{\mathcal{T}}(\boldsymbol{\omega}, \pi) \\
&= \left(\frac{1}{t} \mathbb{I} - \frac{1}{t^3} \bar{\mathcal{T}}(\boldsymbol{\omega}, \pi_0) \cdot \bar{\mathcal{T}}(\boldsymbol{\omega}, \pi_0)^T \right) \partial_\pi \bar{\mathcal{T}}(\boldsymbol{\omega}, \pi) \\
&= \left(\frac{1}{t} \mathbb{I} - \frac{1}{t} \boldsymbol{\omega} \cdot \boldsymbol{\omega}^T \right) \partial_\pi \bar{\mathcal{T}}(\boldsymbol{\omega}, \pi) \\
&= \frac{1}{t} \left(\mathbb{I} - \boldsymbol{\omega} \cdot \boldsymbol{\omega}^T \right) \partial_\pi \bar{\mathcal{T}}(\boldsymbol{\omega}, \pi) \\
&= \frac{1}{t} \left(\mathbb{I} - \boldsymbol{\omega} \cdot \boldsymbol{\omega}^T \right) \partial_\pi \mathcal{V}(\mathbf{x}_t, \pi).
\end{aligned} \tag{C.5}$$

The first equality used the definition of $\mathcal{T}$. In the second equality we evaluate the derivative of the division by the norm and use the chain rule. Here, $\mathbb{I}$ is the 3 by 3 identity matrix and $\bar{\mathcal{T}}(\boldsymbol{\omega}, \pi_0) \cdot \bar{\mathcal{T}}(\boldsymbol{\omega}, \pi_0)^T$ is the outer product. For the terms inside the parentheses, we do not need to track parameter derivatives. In other words, we have $\pi = \pi_0$ and the term $V(\mathbf{x} + t\boldsymbol{\omega}, \pi) - V(\mathbf{x} + t\boldsymbol{\omega}, \pi_0)$ in the definition of $\bar{\mathcal{T}}$ is then equal to 0. This allows simplifying further to arrive at the final expression, where we use the shorthand notation $\mathbf{x}_t \coloneqq \mathbf{x} + t\boldsymbol{\omega}$. This shows that the normalization projects the derivative vector $\partial_\pi \mathcal{V}(\mathbf{x}_t, \pi)$ to tangent space and divides by the distance, producing a vector correctly matching the motion of the discontinuity.

C.3 Jacobian of the reparameterization $\mathcal{T}$

Accounting for the distortion of the integration domain requires computing the Jacobian $\partial_{\boldsymbol{\omega}} \mathcal{T}(\boldsymbol{\omega}, \pi) \in \mathbb{R}^{3\times 3}$. To do so, we first compute the positional Jacobian $\partial_{\mathbf{x}} \mathcal{V}$. The vector field $\mathcal{V}$ is used to define the reparameterization $\mathcal{T}$ and was defined as:

$$\mathcal{V}(\mathbf{x}, \pi) = -\frac{\partial_{\mathbf{x}} \phi(\mathbf{x}, \pi_0)}{\left\| \partial_{\mathbf{x}} \phi(\mathbf{x}, \pi_0) \right\|^2} \phi(\mathbf{x}, \pi) \tag{C.6}$$

Note that only $\phi(\mathbf{x}, \pi)$ is differentiable with respect to the scene parameter π. Taking the derivative with respect to the position $\mathbf{x}$ we get:

$$\partial_{\mathbf{x}} \mathcal{V}(\mathbf{x}, \pi) = -\mathbf{A} \cdot \partial_{\mathbf{x}}^2 \phi(\mathbf{x}, \pi_0) \phi(\mathbf{x}, \pi) - \frac{\partial_{\mathbf{x}} \phi(\mathbf{x}, \pi_0)}{\|\partial_{\mathbf{x}} \phi(\mathbf{x}, \pi_0)\|^2} \cdot \partial_{\mathbf{x}} \phi(\mathbf{x}, \pi)^T \tag{C.7}$$

with $\mathbf{A}$ being the Jacobian of the division by the squared norm:

$$\mathbf{A} = \frac{1}{\|\partial_{\mathbf{x}} \phi(\mathbf{x}, \pi_0)\|^2} \left(\mathbb{I} - \frac{2}{\|\partial_{\mathbf{x}} \phi(\mathbf{x}, \pi_0)\|^2} \partial_{\mathbf{x}} \phi(\mathbf{x}, \pi_0) \cdot \partial_{\mathbf{x}} \phi(\mathbf{x}, \pi_0)^T \right)$$

Accounting for the additional weighting factor $w_{\mathcal{V}}$, we compute the weighted vector field's Jacobian using the product rule:

$$\begin{aligned} \partial_{\mathbf{x}} \bar{\mathcal{V}}(\mathbf{x}, \pi) = \partial_{\boldsymbol{\omega}} \left[w_{\mathcal{V}}(\mathbf{x}) \mathcal{V}(\mathbf{x}, \pi) \right] &= \mathcal{V}(\mathbf{x}, \pi) \cdot \partial_{\boldsymbol{\omega}} w_{\mathcal{V}}(\mathbf{x})^T \\ &+ w_{\mathcal{V}}(\mathbf{x}) \partial_{\boldsymbol{\omega}} \mathcal{V}(\mathbf{x}, \pi) \end{aligned} \tag{C.8}$$

So far, this Jacobian assumes we vary the 3D position $\mathbf{x}$. We need to convert it to a Jacobian with respect to the ray direction $\boldsymbol{\omega}$:

$$\partial_{\boldsymbol{\omega}} \bar{\mathcal{V}}(\mathbf{x}_t, \pi) = \partial_{\mathbf{x}} \bar{\mathcal{V}}(\mathbf{x}, \pi) \partial_{\boldsymbol{\omega}} \mathbf{x}_t, \tag{C.9}$$

where $\partial_{\boldsymbol{\omega}} \mathbf{x}_t = \partial_{\boldsymbol{\omega}} \left[\mathbf{x}_o + t\boldsymbol{\omega} \right] = \mathbb{I} \cdot t + \boldsymbol{\omega} \cdot \partial_{\boldsymbol{\omega}} t^T$. And finally, accounting for the conversion to a reparameterization on the unit sphere:

$$\partial_{\boldsymbol{\omega}} \mathcal{T}(\boldsymbol{\omega}, \pi) = \mathbf{B} \cdot \left(\partial_{\boldsymbol{\omega}} \mathbf{x}_t + \partial_{\boldsymbol{\omega}} \bar{\mathcal{V}}(\mathbf{x}_t, \pi) - \partial_{\boldsymbol{\omega}} \bar{\mathcal{V}}(\mathbf{x}_t, \pi_0) \right), \tag{C.10}$$

where $\mathbf{B}$ is the normalization Jacobian which we used before:

$$\mathbf{B} = \frac{1}{\left\| \bar{\mathcal{T}}(\boldsymbol{\omega}, \pi) \right\|} \mathbb{I} - \frac{1}{\left\| \bar{\mathcal{T}}(\boldsymbol{\omega}, \pi) \right\|^3} \bar{\mathcal{T}}(\boldsymbol{\omega}, \pi) \cdot \bar{\mathcal{T}}(\boldsymbol{\omega}, \pi)^T.$$

Finally, we then simply evaluate $\mathrm{T}\left(\partial_{\boldsymbol{\omega}} \mathcal{T}(\boldsymbol{\omega}, \pi)\right)$. We can further reduce the number of 3×3 matrix products by analyzing this expression more closely. First of all, we know that $\left\| \bar{\mathcal{T}}(\boldsymbol{\omega}, \pi) \right\| = t$ and does not depend on π in a differentiable way (t is computed using π_0). We will also drop any term that is additive and does not depend on π directly, as its derivative will be zero. In the end, we only need the derivative of the trace of the Jacobian:

$$\begin{aligned} \partial_{\pi} \mathrm{T}\left(\partial_{\boldsymbol{\omega}} \mathcal{T}(\boldsymbol{\omega}, \pi)\right) &= \partial_{\pi} \mathrm{T}\left(\mathbf{B} \cdot \left(\partial_{\boldsymbol{\omega}} \mathbf{x}_t + \partial_{\boldsymbol{\omega}} \bar{\mathcal{V}}(\mathbf{x}_t, \pi) - \partial_{\boldsymbol{\omega}} \bar{\mathcal{V}}(\mathbf{x}_t, \pi_0) \right) \right) \\ &= \partial_{\pi} \mathrm{T}\left(\mathbf{B} \cdot \partial_{\boldsymbol{\omega}} \mathbf{x}_t \right) + \partial_{\pi} \mathrm{T}\left(\mathbf{B} \cdot \partial_{\boldsymbol{\omega}} \bar{\mathcal{V}}(\mathbf{x}_t, \pi) \right) \end{aligned}$$

We can now show that $\partial_\pi \mathrm{T}\left(\mathbf{B} \cdot \partial_{\boldsymbol{\omega}} \mathbf{x}_t\right) = 0$:

$$\begin{aligned}\partial_\pi \mathrm{T}\left(\mathbf{B} \cdot \partial_{\boldsymbol{\omega}} \mathbf{x}_t\right) &= \partial_\pi \mathrm{T}\left(\mathbf{B} \cdot \left[\mathbb{I} \cdot t + \boldsymbol{\omega} \cdot \partial_{\boldsymbol{\omega}} t^T\right]\right) \\ &= t \cdot \partial_\pi \mathrm{T}\left(\mathbf{B}\right) + \partial_\pi \mathrm{T}\left(\mathbf{B} \cdot \boldsymbol{\omega} \cdot \partial_{\boldsymbol{\omega}} t^T\right)\end{aligned}$$

Since $\mathbf{B}$ is simply the projection onto tangent space, we know that $\mathrm{T}(\mathbf{B}) = \mathrm{rank}(\mathbf{B}) = 2$ and therefore its derivative is zero. Additionally, $\mathbf{B} \cdot \boldsymbol{\omega} \cdot \partial_{\boldsymbol{\omega}} t^T$ is zero, since $\mathbf{B}$ removes any component in the direction of $\boldsymbol{\omega}$. Hence, $\partial_\pi \mathrm{T}\left(\mathbf{B} \cdot \partial_{\boldsymbol{\omega}} \mathbf{x}_t\right) = 0$ and the derivative simplifies to:

$$\partial_\pi \mathrm{T}\left(\partial_{\boldsymbol{\omega}} \mathcal{T}(\boldsymbol{\omega}, \pi)\right) = \partial_\pi \mathrm{T}\left(\mathbf{B} \cdot \partial_{\boldsymbol{\omega}} \bar{\mathcal{V}}(\mathbf{x}_t, \pi)\right)$$

C.4 Nesting reparameterizations

When reparameterizing a rendering algorithm, we need to correctly nest reparameterizations of the solid angle domain. For example, when rendering an image with direct illumination, we solve the following nested integration for each pixel:

$$I(\pi) = \int_{\mathcal{S}^2} f_0(\boldsymbol{\omega}_0, \pi) \int_{\mathcal{S}^2} f_1(\boldsymbol{\omega}_0, \boldsymbol{\omega}_1, \mathbf{x}_1(\boldsymbol{\omega}_0, \pi), \pi)\, \mathrm{d}\boldsymbol{\omega}_1\, \mathrm{d}\boldsymbol{\omega}_0, \tag{C.11}$$

where f_0 and f_1 contain all relevant importance, visibility, BSDF and incident radiance terms. We use this simplified notation to reduce notational clutter. We explicitly write the dependency of f_1 on the ray intersection $\mathbf{x}_1(\boldsymbol{\omega}_0, \pi)$ of the camera ray in direction $\boldsymbol{\omega}_0$. The inner integration is over reflected directions $\boldsymbol{\omega}_1$.

To differentiate this nested integral, we need to introduce two reparameterizations. The first one reparameterizes the primary ray and the second one the secondary ray (or put more simply, the shadow ray). First, applying a reparameterization $\mathcal{T}_0$ on the primary ray:

$$\begin{aligned}I(\pi) = &\int_{\mathcal{S}^2} f_0(\mathcal{T}_0(\boldsymbol{\omega}_0, \pi), \pi)\, |\mathcal{T}_0| \\ &\int_{\mathcal{S}^2} f_1(\mathcal{T}_0(\boldsymbol{\omega}_0, \pi), \boldsymbol{\omega}_1, \mathbf{x}_1(\mathcal{T}_0(\boldsymbol{\omega}_0, \pi), \pi), \pi)\, \mathrm{d}\boldsymbol{\omega}_1\, \mathrm{d}\boldsymbol{\omega}_0,\end{aligned} \tag{C.12}$$

where in a slight abuse of notation $|\mathcal{T}_0|$ denotes the area element. Assuming a static sensor, we drop the dependency of $\mathcal{T}_0$ on $\mathbf{x}_0$ for clarity. We further reparameterize the

second ray to obtain:

$$I(\pi) = \int_{S^2} f_0(\mathcal{T}_0(\boldsymbol{\omega}_0, \pi), \pi)\, |\mathcal{T}_0| \int_{S^2} f_1\Big(\mathcal{T}_0(\boldsymbol{\omega}_0, \pi), \mathcal{T}_1(\boldsymbol{\omega}_1, \mathbf{x}_1(\mathcal{T}_0(\boldsymbol{\omega}_0, \pi), \pi), \pi), \mathbf{x}_1(\mathcal{T}_0(\boldsymbol{\omega}_0, \pi), \pi), \pi\Big)\, |\mathcal{T}_1|\, \mathrm{d}\boldsymbol{\omega}_1\, \mathrm{d}\boldsymbol{\omega}_0. \tag{C.13}$$

The second reparameterization can now potentially depend on the first reparameterization, and hence π, through the position $\mathbf{x}_1$. On top of that, $\mathbf{x}_1$ might also directly depend on the parameters, e.g., if π controls the distance of the object from the sensor, $\mathbf{x}_1$ will move away or closer to the sensor as π changes.

The remaining question is whether our reparameterization $\mathcal{T}_1$ still evaluates to the correct motion over the unit sphere when differentiated. This can be validated by plugging the parameter-dependent position into the definition of our reparameterization. Equation 6.6 now needs to evaluate $\mathcal{V}$ using a parameter-dependent position $\mathbf{x}$:

$$\partial_\pi \mathcal{V}(\mathbf{x}_t(\pi), \pi) = \partial_\pi \mathcal{V}(\mathbf{x}(\pi) + t\boldsymbol{\omega}, \pi). \tag{C.14}$$

We previously assumed the ray origin to be fixed and did not consider the parameter dependence of $\mathbf{x}_t$. Here we simplified the notation by omitting the explicit dependency of $\mathbf{x}$ on $\mathcal{T}_0$.

We can now use the following fact: when solving the inner integral and integrating over secondary rays, $\mathbf{x}(\pi)$ is fixed. That means that the dependency of $\mathbf{x}$ on π can be re-interpreted as simply another way in which the SDF values depend on π. When evaluating $\mathcal{V}$ we therefore need to evaluate

$$\mathcal{V}(\mathbf{x}_t(\pi), \pi) = -\frac{\partial_{\mathbf{x}} \phi(\mathbf{x}_t(\pi_0), \pi_0)}{\|\partial_{\mathbf{x}} \phi(\mathbf{x}_t(\pi_0), \pi_0)\|^2} \phi(\mathbf{x}_t(\pi), \pi). \tag{C.15}$$

It is important to detach the evaluation of the positional gradient from the parameter dependence introduced through $\mathbf{x}$. With this, we moved the parameter dependency of $\mathbf{x}$ into the vector field evaluation, which then in turn guarantees the correct (relative) motion of the surface, as our formulation of $\mathcal{V}$ does not assume any specific relation of π and ϕ in order to produce a valid reparameterization. For future work, it could be interesting to generalize these derivations to path space, similar to the work by Zhang et al. [160].

D | Non-exponential transmittance

D.1 Ray marching algorithm

We use ray marching to estimate the transmittance by our model. Our transmittance model was defined to be:

$$\mathrm{T}(\mathbf{x}, \mathbf{y}) = 1 + \int_0^{\|\mathbf{x}-\mathbf{y}\|} \frac{\partial f}{\partial \tau} \left(f^{-1} \left(\mathrm{T}(\mathbf{x}, \mathbf{x}_t), \gamma(t) \right), \gamma(t) \right) \sigma_t(t) \, \mathrm{d}t \tag{D.1}$$

To evaluate the transmittance, we need to approximate the value of this integral. We can do that by applying a Riemann summation:

$$\mathrm{T}(\mathbf{x}, \mathbf{y}) \approx 1 + \sum_{i=1}^{N} \frac{\partial f}{\partial \tau} \left(f^{-1} \left(\mathrm{T}_{i-1}, \gamma(t_i) \right), \gamma(t_i) \right) \sigma_t(t_i) \Delta_{\text{step}} \tag{D.2}$$

where for convenience we defined the transmittance after $i-1$ steps in the summation as T_{i-1} (with $\mathrm{T}_0 = 1$). However, when trying to use this formulation, one will encounter an issue. Our transmittance model was derived using the fundamental theorem of calculus, but the fundamental theorem of calculus does not hold under a basic Riemann summation. This can be fixed by also using a discrete version of the derivative term in the integrand. We use a finite difference approximation with the step size of $\sigma_t \Delta_{\text{step}}$:

$$\frac{\partial f}{\partial \tau}(\tau, \gamma) \approx \frac{1}{\sigma_t \Delta_{\text{step}}} \left[f(\tau + \sigma_t \Delta_{\text{step}}, \gamma) - f(\tau, \gamma) \right] \tag{D.3}$$

Using the extinction as the finite difference offset is convenient, as the division by it will then cancel out with the $\sigma_t \Delta_{\text{step}}$ term in the original Riemann sum:

$$\begin{aligned} \mathrm{T}(\mathbf{x}, \mathbf{y}) &\approx 1 + \sum_{i=1}^{N} \frac{1}{\sigma_t(t_i) \Delta_{\text{step}}} \Big[f \left(f^{-1}(\mathrm{T}_{i-1}, \gamma(t_i)) + \sigma_t(t_i) \Delta_{\text{step}}, \gamma(t_i), \gamma(t_i) \right) \\ &\qquad - f \left(f^{-1}(\mathrm{T}_{i-1}, \gamma(t_i)), \gamma(t_i) \right) \Big] \sigma_t(t_i) \Delta_{\text{step}} \\ &= 1 + \sum_{i=1}^{N} f \left(f^{-1}(\mathrm{T}_{i-1}, \gamma(t_i)) + \sigma_t(t_i) \Delta_{\text{step}}, \gamma(t_i) \right) - \mathrm{T}_{i-1} \end{aligned} \tag{D.4}$$

In the last step, we actually arrive at a telescoping sum. This means that we can also write the ray marching algorithm as a simple recursive function:

$$\mathrm{T}(\mathbf{x}, \mathbf{y}) \approx f \left(f^{-1} \left(\mathrm{T}_{N-1}, \gamma(t_i) \right) + \sigma_t(t_i) \Delta_{\text{step}}, \gamma(t_i) \right) \tag{D.5}$$

Each iteration in the ray marching algorithm can simply just update the current value of T based on its previous value and the current values for σ_t and γ.

To evaluate our model, we need to invert the transmittance function $f(\tau, \gamma)$ with respect to the first parameter. The simple form of f allows expressing the inverse explicitly using the Lambert W function:

$$f^{-1}(y, \gamma) = \begin{cases} 2 - 2y & \gamma = 0 \\ -\log(y/\gamma) & \gamma \exp(-2) > y \\ 2\frac{y+\gamma-1}{\gamma-1} + W\left[-2\gamma \exp\left(-2\frac{y+\gamma-1}{\gamma-1}\right)\right] & \gamma \exp(-2) \leq y \end{cases}. \tag{D.6}$$

D.2 Volume access statistics

In the following, we provide additional statistics on the number of non-empty voxels accessed at render time. This shows that a sparse voxelization of the scene scales favorably as the resolution increases.

Resolution	16	32	64	128
City building	153.1	204.9	231.1	257.2
City	97.7	102.6	128.3	161.0
Trees	122.9	177.9	242.5	283.9
Checkerboards	230.2	291.9	288.8	310.4
Fractal	269.1	347.8	474.4	532.7
Plane	126.5	125.9	152.1	150.1

Table D.1: The amount of non-empty voxel data accessed per sample for different scenes and resolutions. The reported numbers are the average number of bytes accessed while rendering different views of the scene. The data is measured *per sample*, using multiple scattering and next event estimation. These measurements show that the growth in data access size is sublinear as the volume resolution increases.

References

[1] Hideki Tamura, Hiroshi Higashi, and Shigeki Nakauchi. "Dynamic Visual Cues for Differentiating Mirror and Glass". In: *Scientific Reports* 8.8403 (May 2018). DOI: 10.1038/s41598-018-26720-x.

[2] Alan L. Gilchrist. "The Perception of Surface Blacks and Whites". In: *Scientific American* 240.3 (Mar. 1979). DOI: 10.1038/scientificamerican0379-112.

[3] James T. Kajiya. "The Rendering Equation". In: *Computer Graphics (Proc. SIGGRAPH)*. 1986. DOI: 10.1145/15886.15902.

[4] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation (3rd ed.)* 3rd. Morgan Kaufmann Publishers Inc., Oct. 2016, p. 1266.

[5] Alexander Keller, Luca Fascione, Marcos Fajardo, Iliyan Georgiev, Per Christensen, Johannes Hanika, Christian Eisenacher, and Greg Nichols. "The Path Tracing Revolution in the Movie Industry". In: *ACM SIGGRAPH Courses*. 2015. DOI: 10.1145/2776880.2792699.

[6] Petrik Clarberg, Simon Kallweit, Craig Kolb, Pawel Kozlowski, Yong He, Lifan Wu, Edward Liu, Benedikt Bitterli, and Matt Pharr. *Real-Time Path Tracing and Beyond*. HPG 2022 Keynote. July 2022. URL: https://www.youtube.com/watch?t=2507&v=au4cPLuEpNM.

[7] Roshdi Rashed. "A Pioneer in Anaclastics: Ibn Sahl on Burning Mirrors and Lenses". In: *Isis* 81.3 (1990). DOI: 10.4324/9781315248011-14.

[8] Nicholas Metropolis and Stanislaw Ulam. "The Monte Carlo Method". In: *Journal of the American Statistical Association* 44.247 (1949). DOI: 10.1080/01621459.1949.10483310.

[9] Mark Pauly, Thomas Kollig, and Alexander Keller. "Metropolis Light Transport for Participating Media". In: *Proceedings of the Eurographics Workshop on Rendering Techniques*. 2000. DOI: 10.1007/978-3-7091-6303-0_2.

[10] Jan Novák, Andrew Selle, and Wojciech Jarosz. "Residual Ratio Tracking for Estimating Attenuation in Participating Media". In: *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 33.6 (Nov. 2014). DOI: 10.1145/2661229.2661292.

[11] Henrik Wann Jensen, Stephen R. Marschner, Marc Levoy, and Pat Hanrahan. "A Practical Model for Subsurface Light Transport". In: *SIGGRAPH Comput. Graph.* 2001. DOI: 10.1145/383259.383319.

[12] Craig Donner and Henrik Wann Jensen. "Light Diffusion in Multi-layered Translucent Materials". In: *ACM Trans. Graph. (Proc. SIGGRAPH).* 2005. DOI: 10.1145/1073204.1073308.

[13] Delio Vicini, Vladlen Koltun, and Wenzel Jakob. "A Learned Shape-Adaptive Subsurface Scattering Model". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 38.4 (July 2019). DOI: 10.1145/3306346.3322974.

[14] Stephen R. Marschner, Henrik Wann Jensen, Mike Cammarano, Steve Worley, and Pat Hanrahan. "Light Scattering from Human Hair Fibers". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 22.3 (July 2003). DOI: 10.1145/882262.882345.

[15] Ling-Qi Yan, Miloš Hašan, Wenzel Jakob, Jason Lawrence, Steve Marschner, and Ravi Ramamoorthi. "Rendering Glints on High-Resolution Normal-Mapped Specular Surfaces". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 33.4 (July 2014). DOI: 10.1145/2601097.2601155.

[16] Wenzel Jakob, Miloš Hašan, Ling-Qi Yan, Ravi Ramamoorthi, and Steve Marschner. "Discrete Stochastic Microfacet Models". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 33.4 (July 2014). DOI: 10.1145/2601097.2601186.

[17] Wenzel Jakob, Eugene d'Eon, Otto Jakob, and Steve Marschner. "A Comprehensive Framework for Rendering Layered Materials". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 33.4 (July 2014). DOI: 10.1145/2601097.2601139.

[18] Ioannis Gkioulekas, Shuang Zhao, Kavita Bala, Todd Zickler, and Anat Levin. "Inverse Volume Rendering with Material Dictionaries". In: *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 32.6 (Nov. 2013). DOI: 10.1145/2508363.2508377.

[19] Romain Prévost, Emily Whiting, Sylvain Lefebvre, and Olga Sorkine-Hornung. "Make It Stand: Balancing Shapes for 3D Fabrication". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 32.4 (2013). DOI: 10.1145/2461912.2461957.

[20] Moritz Bächer, Emily Whiting, Bernd Bickel, and Olga Sorkine-Hornung. "Spin-It: Optimizing Moment of Inertia for Spinnable Objects". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 33.4 (July 2014). DOI: 10.1145/2601097.2601157.

[21] Yue Dong, Jiaping Wang, Fabio Pellacini, Xin Tong, and Baining Guo. "Fabricating spatially-varying subsurface scattering". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 29.4 (July 2010). DOI: 10.1145/1778765.1778799.

[22] Marios Papas, Christian Regg, Wojciech Jarosz, Bernd Bickel, Philip Jackson, Wojciech Matusik, Steve Marschner, and Markus Gross. "Fabricating Translucent Materials using Continuous Pigment Mixtures". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 32.4 (July 2013). DOI: 10.1145/2461912.2461974.

[23] Qilin Sun, Congli Wang, Fu Qiang, Dun Xiong, and Heidrich Wolfgang. "End-to-End Complex Lens Design with Differentiable Ray Tracing". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 40.4 (2021).

[24] Tao Ren, Michael F. Modest, and Somesh Roy. "Monte Carlo Simulation for Radiative Transfer in a High-Pressure Industrial Gas Turbine Combustion Chamber". In: *Journal of Engineering for Gas Turbines and Power* 140.5 (Dec. 2017). DOI: 10.1115/1.4038153.

[25] Maxime Roger, Mouna El-Hafi, Richard A Fournier, Stéphane Blanco, Amaury Guilhem de Lataillade, Vincent Eymet, and Patrice Perez. "Applications of sensitivity estimations by Monte-Carlo methods". In: *International symposium on radiative transfer*. June 2004. DOI: 10.1615/ICHMT.2004.RAD-4.50.

[26] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. "Equation of State Calculations by Fast Computing Machines". In: *The Journal of Chemical Physics* 21.6 (1953). DOI: 10.1063/1.1699114.

[27] G. E. Albert. *A general theory of stochastic estimates of the Neumann series of certain Fredholm integral equations and related series*. Tech. rep. Aug. 1953. DOI: 10.2172/4427633.

[28] Guennady A. Mikhailov. "Monte-Carlo calculation of derivatives of functionals from the solution of the transfer equation according to the parameters of the system". In: *USSR Computational Mathematics and Mathematical Physics* 7.4 (1967). DOI: 10.1016/0041-5553(67)90162-0.

[29] M.Z Brainina, V.L Generozov, V.G Kuznetsov, and V.A Sakovich. "Evaluation of dose derivatives by the Monte Carlo method for optimizing protective screen shape and composition". In: *USSR Computational Mathematics and Mathematical Physics* 7.4 (1967). DOI: 10.1016/0041-5553(67)90170-X.

[30] L.L. Sidorenko and A.I. Khisamutdinov. “Evaluation by Monte Carlo methods of the derivatives of linear functionals of the flow with respect to the parameters of surfaces”. In: *USSR Computational Mathematics and Mathematical Physics* 21.3 (1981). DOI: 10.1016/0041-5553(81)90087-2.

[31] Guennady A. Mikhailov. “On the calculation of nuclear reactor disturbances by the Monte Carlo method”. In: *USSR Computational Mathematics and Mathematical Physics* 6.2 (1966). DOI: 10.1016/0041-5553(66)90076-0.

[32] Iván Lux and Lázló Koblinger. *Monte Carlo Particle Transport Methods: Neutron and Photon Calculations*. Boston: CRC Press, 1990. DOI: 10.1201/9781351074834.

[33] F. W. Paul Götz, A. R. Meetham, and Gordon Miller Bourne Dobson. “The vertical distribution of ozone in the atmosphere”. In: *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character* 145.855 (1934). DOI: 10.1038/132281a0.

[34] Clive D. Rogers. *Inverse methods for atmospheric sounding : theory and practice*. Series on atmospheric, oceanic and planetary physics. World Scientific Publishing, 2000. DOI: 10.1142/3171.

[35] Tamar Loeub, Aviad Levis, Vadim Holodovsky, and Yoav Y. Schechner. “Monotonicity Prior for Cloud Tomography”. In: *European Conference on Computer Vision (ECCV)*. 2020. DOI: 10.1007/978-3-030-58523-5_17.

[36] Viktor S. Antyufeev. *Monte Carlo Method for Solving Inverse Problems of Radiation Transfer*. Inverse and ill-posed problems. VSP, 2000. DOI: 10.1515/9783110920307.

[37] Habib Zaidi and Marie-Louise Montandon. “Scatter Compensation Techniques in PET”. In: *PET Clinics* 2.2 (Feb. 2008). DOI: 10.1016/j.cpet.2007.10.003.

[38] Lorenz Kuger and Gaël Rigaud. “On multiple scattering in Compton scattering tomography and its impact on fan-beam CT”. In: *Inverse Problems and Imaging* 16.5 (2022). DOI: 10.3934/ipi.2022029.

[39] Mervin E. Muller. “Some Continuous Monte Carlo Methods for the Dirichlet Problem”. In: *The Annals of Mathematical Statistics* 27.3 (1956). DOI: 10.1214/aoms/1177728169.

[40] Boris S. Elepov and Guennady A. Mikhailov. “Solution of the dirichlet problem for the equation $\Delta u - cu = -q$ by a model of “walks on spheres””. In: *USSR Computational Mathematics and Mathematical Physics* 9.3 (1969). DOI: 10.1016/0041-5553(69)90070-6.

[41] John M DeLaurentis and Louis A Romero. “A Monte Carlo method for Poisson’s equation”. In: *Journal of Computational Physics* 90.1 (1990). DOI: 10.1016/0021-9991(90)90199-B.

[42] Rohan Sawhney and Keenan Crane. “Monte Carlo geometry processing: A grid-free approach to PDE-based methods on volumetric domains”. In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 39.4 (2020). DOI: 10.1145/3386569.3392374.

[43] Rohan Sawhney, Dario Seyb, Wojciech Jarosz, and Keenan Crane. “Grid-free Monte Carlo for PDEs with spatially varying coefficients”. In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 41.4 (2022). DOI: 10.1145/3528223.3530134.

[44] Ekrem Fatih Yilmazer, Delio Vicini, and Wenzel Jakob. *Solving Inverse PDE Problems using Grid-Free Monte Carlo Estimators*. 2022. arXiv: 2208.02114.

[45] Delio Vicini, Sébastien Speierer, and Wenzel Jakob. “Path Replay Backpropagation: Differentiating Light Paths using Constant Memory and Linear Time”. In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 40.4 (Aug. 2021). DOI: 10.1145/3450626.3459804.

[46] Delio Vicini, Sébastien Speierer, and Wenzel Jakob. “Differentiable Signed Distance Function Rendering”. In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 41.4 (July 2022). DOI: 10.1145/3528223.3530139.

[47] Delio Vicini, Wenzel Jakob, and Anton Kaplanyan. “A Non-Exponential Transmittance Model for Volumetric Scene Representations”. In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 40.4 (Aug. 2021). DOI: 10.1145/3450626.3459815.

[48] Merlin Nimier-David, Delio Vicini, Tizian Zeltner, and Wenzel Jakob. “Mitsuba 2: A Retargetable Forward and Inverse Renderer”. In: *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 38.6 (Nov. 2019). DOI: 10.1145/3355089.3356498.

[49] Wenzel Jakob, Sébastien, Nicolas Roussel, and Delio Vicini. “Dr.Jit: A Just-In-Time Compiler for Differentiable Rendering”. In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 41.4 (July 2022). DOI: 10.1145/3528223.3530099.

[50] Wenzel Jakob, Sébastien Speierer, Nicolas Roussel, Merlin Nimier-David, Delio Vicini, Tizian Zeltner, Baptiste Nicolet, Miguel Crespo, Vincent Leroy, and Ziyi Zhang. *Mitsuba 3 renderer*. Version 3.0.1. https://mitsuba-renderer.org. 2022.

[51] Eric Veach. "Robust Monte Carlo Methods for Light Transport Simulation". PhD thesis. Stanford, CA: Stanford University, Dec. 1997.

[52] Matthias Zwicker, Wojciech Jarosz, Jaakko Lehtinen, Bochang Moon, Ravi Ramamoorthi, Fabrice Rousselle, Pradeep Sen, Cyril Soler, and Sung-Eui Yoon. "Recent Advances in Adaptive Sampling and Reconstruction for Monte Carlo Rendering". In: *Computer Graphics Forum (Proc. Eurographics - State of the Art Reports)* 34.2 (May 2015). DOI: 10.1111/cgf.12592.

[53] Zackary Misso, Benedikt Bitterli, Iliyan Georgiev, and Wojciech Jarosz. "Unbiased and consistent rendering using biased estimators". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 41.4 (July 2022). DOI: 10.1145/3528223.3530160.

[54] Eric P. Lafortune and Yves D. Willems. "Bi-directional path tracing". In: *Proc. International Conference on Computational Graphics and Visualization Techniques (Compugraphics '93)*. Dec. 1993.

[55] Eric Veach and Leonidas Guibas. "Bidirectional estimators for light transport". In: *Photorealistic Rendering Techniques*. Springer, 1995, pp. 145–167. DOI: 10.1007/978-3-642-87825-1_11.

[56] Eric P. Lafortune and Yves D. Willems. "A 5D tree to reduce the variance of Monte Carlo ray tracing". In: *Rendering Techniques (Proc. EG Workshop on Rendering)*. 1995. DOI: 10.1007/978-3-7091-9430-0_2.

[57] Jiří Vorba, Ondřej Karlík, Martin Šik, Tobias Ritschel, and Jaroslav Křivánek. "Online Learning of Parametric Mixture Models for Light Transport Simulation". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 33.4 (Aug. 2014). DOI: 10.1145/2601097.2601203.

[58] Thomas Müller, Markus Gross, and Jan Novák. "Practical Path Guiding for Efficient Light-Transport Simulation". In: *Computer Graphics Forum (Proc. EGSR)* 36.4 (June 2017). DOI: 10.1111/cgf.13227.

[59] Eric Heitz. "Sampling the GGX Distribution of Visible Normals". In: *Journal of Computer Graphics Techniques (JCGT)* 7.4 (Nov. 2018).

[60] Eric Veach and Leonidas J. Guibas. “Optimally Combining Sampling Techniques for Monte Carlo Rendering”. In: *SIGGRAPH Comput. Graph.* 1995. DOI: 10.1145/218380.218498.

[61] Kondapaneni Ivo, Petr Vévoda, Pascal Grittmann, Tomáš Skřivan, Philipp Slusallek, and Jaroslav Křivánek. “Optimal Multiple Importance Sampling”. In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 38.4 (July 2019). DOI: 10.1145/3306346.3323009.

[62] Fabrice Rousselle, Wojciech Jarosz, and Jan Novák. “Image-space Control Variates for Rendering”. In: *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 35.6 (Dec. 2016). DOI: 10.1145/2980179.2982443.

[63] Miguel Crespo, Adrian Jarabo, and Adolfo Muñoz. “Primary-Space Adaptive Control Variates using Piecewise-Polynomial Approximations”. In: *ACM Trans. Graph.* 40.3 (July 2021). DOI: 10.1145/3450627.

[64] Thomas Müller, Fabrice Rousselle, Alexander Keller, and Jan Novák. “Neural Control Variates”. In: *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 39.6 (Nov. 2020). DOI: 10.1145/3414685.3417804.

[65] Jaakko Lehtinen, Tero Karras, Samuli Laine, Miika Aittala, Frédo Durand, and Timo Aila. “Gradient-Domain Metropolis Light Transport”. In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 32.4 (2013). DOI: 10.1145/2461912.2461943.

[66] Markus Kettunen, Marco Manzi, Miika Aittala, Jaakko Lehtinen, Frédo Durand, and Matthias Zwicker. “Gradient-Domain Path Tracing”. In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 34.4 (2015). DOI: 10.1145/2766997.

[67] Cheng Zhang, Zhao Dong, Michael Doggett, and Shuang Zhao. “Antithetic Sampling for Monte Carlo Differentiable Rendering”. In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 40.4 (Aug. 2021). DOI: 10.1145/3450626.3459783.

[68] Tizian Zeltner, Sébastien Speierer, Iliyan Georgiev, and Wenzel Jakob. “Monte Carlo Estimators for Differential Light Transport”. In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 40.4 (Aug. 2021). DOI: 10.1145/3450626.3459807.

[69] Sai Bangaru, Tzu-Mao Li, and Frédo Durand. “Unbiased Warped-Area Sampling for Differentiable Rendering”. In: *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 39.6 (2020). DOI: 10.1145/3414685.3417833.

[70] Melissa E. O'Neill. *PCG: A Family of Simple Fast Space-Efficient Statistically Good Algorithms for Random Number Generation.* Tech. rep. HMC-CS-2014-0905. Harvey Mudd College, Sept. 2014. URL: https://www.cs.hmc.edu/tr/hmc-cs-2014-0905.pdf (Accessed on 30/08/2022).

[71] Luc Devroye. *Non-Uniform Random Variate Generation.* New York: Springer-Verlag, 1986. DOI: 10.1007/978-1-4613-8643-8.

[72] Peter A. Morris, Reuben S. Aspden, Jessica E. C. Bell, Robert W. Boyd, and Miles J. Padgett. "Imaging with a small number of photons". In: *Nature Communications* 6.1 (Jan. 2015). DOI: 10.1038/ncomms6913.

[73] Peter Seitz and Albert J.P. Theuwissen. *Single-Photon Imaging*. Springer Series in Optical Sciences. Springer, 2011. DOI: 10.1007/978-3-642-18443-7.

[74] Craig Kolb, Don Mitchell, and Pat Hanrahan. "A Realistic Camera Model for Computer Graphics". In: *SIGGRAPH Comput. Graph.* 1995. DOI: 10.1145/218380.218463.

[75] Jonathan Dupuy and Wenzel Jakob. "An Adaptive Parameterization for Efficient Material Acquisition and Rendering". In: *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 37.6 (Nov. 2018). DOI: 10.1145/3272127.3275059.

[76] Petr Beckmann and Andre Spizzichino. *The Scattering of Electromagnetic Waves from Rough Surfaces*. Pergamon, 1963.

[77] Kenneth E. Torrance and Ephraim M. Sparrow. "Theory for Off-Specular Reflection From Roughened Surfaces". In: *Journal of the Optical Society of America* 57.9 (Sept. 1967). DOI: 10.1364/JOSA.57.001105.

[78] T. S. Trowbridge and K. P. Reitz. "Average irregularity representation of a rough surface for ray reflection". In: *Journal of the Optical Society of America* 65.5 (May 1975). DOI: 10.1364/JOSA.65.000531.

[79] Robert L. Cook and Kenneth E. Torrance. "A Reflectance Model for Computer Graphics". In: *ACM Trans. Graph.* 1.1 (Jan. 1982). DOI: 10.1145/965161.806819.

[80] Bruce Walter, Stephen R. Marschner, Hongsong Li, and Kenneth E. Torrance. "Microfacet Models for Refraction through Rough Surfaces". In: *Proceedings of the 18th Eurographics Conference on Rendering Techniques*. EGSR'07. 2007.

[81] Eric Heitz, Johannes Hanika, Eugene d'Eon, and Carsten Dachsbacher. "Multiple-Scattering Microfacet BSDFs with the Smith Model". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 35.4 (July 2016). DOI: 10.1145/2897824.2925943.

[82] Yu Guo, Miloš Hašan, and Shuang Zhao. "Position-Free Monte Carlo Simulation for Arbitrary Layered BSDFs". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 37.6 (Dec. 2018). DOI: 10.1145/3272127.3275053.

[83] Andrea Weidlich and Alexander Wilkie. "Arbitrarily Layered Micro-Facet Surfaces". In: *Proceedings of the 5th International Conference on Computer Graphics and Interactive Techniques in Australia and Southeast Asia*. GRAPHITE '07. 2007. DOI: 10.1145/1321261.1321292.

[84] Tizian Zeltner and Wenzel Jakob. "The Layer Laboratory: A Calculus for Additive and Subtractive Composition of Anisotropic Surface Reflectance". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 37.4 (July 2018). DOI: 10.1145/3197517.3201321.

[85] Brent Burley. "Physically-based shading at Disney". In: *SIGGRAPH Course Notes. Practical physically-based shading in film and game production.* (2012).

[86] Christian Lessig, Eugene Fiume, and Mathieu Desbrun. *On the Mathematical Formulation of Radiance*. 2012. arXiv: 1205.4447.

[87] Eric Veach and Leonidas J. Guibas. "Metropolis Light Transport". In: *SIGGRAPH Comput. Graph.* 1997. DOI: 10.1145/258734.258775.

[88] Subrahmanyan Chandrasekhar. *Radiative transfer*. New York: Dover publications, 1960.

[89] Louis G. Henyey and Jesse L. Greenstein. "Diffuse radiation in the galaxy". In: *Astrophysical Journal* 93 (1941). DOI: 10.1086/144246.

[90] Wenzel Jakob, Jonathan T. Moon, Adam Arbree, Kavita Bala, and Steve Marschner. "A Radiative Transfer Framework for Rendering Materials with Anisotropic Structure". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 29.10 (July 2010). DOI: 10.1145/1778765.1778790.

[91] Eric Heitz, Jonathan Dupuy, Cyril Crassin, and Carsten Dachsbacher. "The SGGX Microflake Distribution". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 34.4 (July 2015). DOI: 10.1145/2766988.

[92] Shuang Zhao, Wenzel Jakob, Steve Marschner, and Kavita Bala. "Structure-aware synthesis for predictive woven fabric appearance". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 31.4 (2012). DOI: 10.1145/2185520.2185571.

[93] Guillaume Loubet and Fabrice Neyret. "A new microflake model with microscopic self-shadowing for accurate volume downsampling". In: *Computer Graphics Forum* 37.2 (2018). DOI: 10.1111/cgf.13346.

[94] Eric Heitz and Eugene d'Eon. "Importance Sampling Microfacet-Based BSDFs using the Distribution of Visible Normals". In: *Computer Graphics Forum (Proc. EGSR)* 33.4 (2014). DOI: 10.1111/cgf.12417.

[95] Bailey Miller, Iliyan Georgiev, and Wojciech Jarosz. "A Null-Scattering Path Integral Formulation of Light Transport". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 38.4 (July 2019). DOI: 10.1145/3306346.3323025.

[96] E. R. Woodcock, T. Murphy, P. J. Hemmings, and T. C. Longworth. "Techniques used in the GEM code for Monte Carlo neutronics calculations in reactors and other systems of complex geometry". In: *Applications of Computing Methods to Reactor Problems* (1965).

[97] Mathieu Galtier, Stéphane Blanco, Cyril Caliot, Christophe Coustet, Jérémi Dauchet, Mouna El Hafi, Vincent Eymet, Richard Fournier, Jacques Gautrais, Anaïs Khuong, et al. "Integral formulation of null-collision Monte Carlo algorithms". In: *Journal of Quantitative Spectroscopy and Radiative Transfer* 125 (2013). DOI: 10.1016/j.jqsrt.2013.04.001.

[98] Peter Kutz, Ralf Habel, Yining Karl Li, and Jan Novák. "Spectral and Decomposition Tracking for Rendering Heterogeneous Volumes". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 36.4 (2017). DOI: 10.1145/3072959.3073665.

[99] Wenzel Jakob and Steve Marschner. "Manifold Exploration: A Markov Chain Monte Carlo Technique for Rendering Scenes with Difficult Specular Transport". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 31.4 (July 2012). DOI: 10.1145/2185520.2185554.

[100] Henrik Wann Jensen. "Importance Driven Path Tracing using the Photon Map". In: *Rendering Techniques (Proc. EG Workshop on Rendering)*. 1995. DOI: 10.1007/978-3-7091-9430-0_31.

[101] Thomas Müller, Brian McWilliams, Fabrice Rousselle, Markus Gross, and Jan Novák. "Neural Importance Sampling". In: *ACM Trans. Graph.* 38.5 (Oct. 2019). DOI: 10.1145/3341156.

[102] Alexander Wilkie, Sehera Nawaz, Marc Droske, Andrea Weidlich, and Johannes Hanika. "Hero Wavelength Spectral Sampling". In: *Computer Graphics Forum (Proc. EGSR)* 33.4 (July 2014). DOI: 10.1111/cgf.12419.

[103] Heang K. Tuy and Lee Tan Tuy. "Direct 2-D display of 3-D objects". In: *IEEE Computer Graphics and Applications* 4.10 (1984). DOI: 10.1109/MCG.1984.6429333.

[104] K. Perlin and E. M. Hoffert. "Hypertexture". In: *SIGGRAPH Comput. Graph.* 23.3 (July 1989). DOI: 10.1145/74334.74359.

[105] Markus Kettunen, Eugene d'Eon, Jacopo Pantaleoni, and Jan Novák. "An Unbiased Ray-Marching Transmittance Estimator". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 40.4 (July 2021). DOI: 10.1145/3450626.3459937.

[106] Matthias Raab, Daniel Seibert, and Alexander Keller. "Unbiased Global Illumination with Participating Media". In: *Monte Carlo and Quasi-Monte Carlo Methods 2006*. 2008. DOI: 10.1007/978-3-540-74496-2_35.

[107] Jan Novák, Iliyan Georgiev, Johannes Hanika, Jaroslav Křivánek, and Wojciech Jarosz. "Monte Carlo Methods for Physically Based Volume Rendering". In: *ACM SIGGRAPH Courses*. 2018. DOI: 10.1145/3214834.3214880.

[108] Augustin-Louis Cauchy. "Méthode générale pour la résolution des systèmes d'équations simultanées". In: *Comptes rendus de l'Académie des Sciences, Paris* 25 (1847).

[109] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *J. Mach. Learn. Res.* 12 (July 2011).

[110] Matthew D. Zeiler. *ADADELTA: An Adaptive Learning Rate Method*. 2012. arXiv: 1212.5701.

[111] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *Proc. International Conference on Learning Representations (ICLR), Conference Track*. 2015.

[112] Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. "Differentiable Monte Carlo Ray Tracing through Edge Sampling". In: *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 37.6 (2018). DOI: 10.1145/3272127.3275109.

[113] James C. Spall. "A Stochastic Approximation Technique for Generating Maximum Likelihood Parameter Estimates". In: *1987 American Control Conference*. 1987.

[114] John F Nolan. "Analytical differentiation on a digital computer". PhD thesis. Massachusetts Institute of Technology, 1953.

[115] Robert Edwin Wengert. "A simple automatic derivative evaluation program". In: *Communications of the ACM* 7.8 (1964). DOI: 10.1145/355586.364791.

[116] Seppo Linnainmaa. "Taylor expansion of the accumulated rounding error". In: *BIT Numerical Mathematics* 16.2 (1976). DOI: 10.1007/BF01931367.

[117] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning Representations by Back-propagating Errors". In: *Nature* 323.6088 (1986). DOI: 10.1038/323533a0.

[118] Andreas Griewank and Andrea Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. Vol. 105. SIAM, 2008.

[119] Yu M Volin and GM Ostrovskii. "Automatic computation of derivatives with the use of the multilevel differentiating technique—1. algorithmic basis". In: *Computers & mathematics with applications* 11.11 (1985).

[120] Andreas Griewank and Andrea Walther. "Algorithm 799: revolve: an implementation of checkpointing for the reverse or adjoint mode of computational differentiation". In: *ACM Transactions on Mathematical Software (TOMS)* 26.1 (2000). DOI: 10.1145/347837.347846.

[121] Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Jia Yangqing, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. URL: https://www.tensorflow.org/ (Accessed on 30/08/2022).

[122] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2019, pp. 8024–8035.

[123] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. *cuDNN: Efficient Primitives for Deep Learning*. 2014. arXiv: 1410.0759.

[124] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. *JAX: composable transformations of Python + NumPy programs*. 2018. URL: http://github.com/google/jax (Accessed on 30/08/2022).

[125] Google. *XLA: Optimizing Compiler for Machine Learning*. 2017. URL: https://www.tensorflow.org/xla (Accessed on 30/08/2022).

[126] Laurent Hascoet and Valérie Pascual. "The Tapenade automatic differentiation tool: Principles, model, and specification". In: *ACM Transactions on Mathematical Software (TOMS)* 39.3 (2013). DOI: 10.1145/2450153.2450158.

[127] Michael Innes. *Don't Unroll Adjoint: Differentiating SSA-Form Programs*. 2019. arXiv: 1810.07951 [cs.PL].

[128] William S. Moses, Valentin Churavy, Ludger Paehler, Jan Hückelheim, Sri Hari Krishna Narayanan, Michel Schanen, and Johannes Doerfert. "Reverse-Mode Automatic Differentiation and Optimization of GPU Kernels via Enzyme". In: *Proc. International Conference for High Performance Computing, Networking, Storage and Analysis*. 2021. DOI: 10.1145/3458817.3476165.

[129] Chris Lattner and Vikram Adve. "LLVM: A Compilation Framework for Lifelong Program Analysis and Transformation". In: *Proc. International Symposium on Code Generation and Optimization*. Mar. 2004. DOI: 10.1109/CGO.2004.1281665.

[130] Barak A. Pearlmutter and Jeffrey Mark Siskind. "Reverse-Mode AD in a Functional Framework: Lambda the Ultimate Backpropagator". In: *ACM Trans. Program. Lang. Syst.* 30.2 (Mar. 2008). DOI: 10.1145/1330017.1330018.

[131] Cheng Zhang and Edward Liu. *TensorRay: Path-Space Differentiable Renderer on the GPU*. 2022. URL: https://github.com/TensorRay/TensorRay (Accessed on 30/08/2022).

[132] Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. "DiffTaichi: Differentiable Programming for Physical Simulation". In: *Proc. International Conference on Learning Representations (ICLR), Conference Track* (2020).

[133] Yuanming Hu, Tzu-Mao Li, Luke Anderson, Jonathan Ragan-Kelley, and Frédo Durand. "Taichi: a language for high-performance computation on spatially sparse data structures". In: *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 38.6 (2019). DOI: 10.1145/3355089.3356506.

[134] Jonathan Ragan-Kelley, Andrew Adams, Sylvain Paris, Marc Levoy, Saman Amarasinghe, and Frédo Durand. "Decoupling Algorithms from Schedules for Easy Optimization of Image Processing Pipelines". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 31.4 (July 2012). DOI: 10.1145/2185520.2185528.

[135] Tzu-Mao Li, Michaël Gharbi, Andrew Adams, Frédo Durand, and Jonathan Ragan-Kelley. "Differentiable programming for image processing and deep learning in Halide". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 37.4 (2018). DOI: 10.1145/3197517.3201383.

[136] Tzu-Mao Li, Michal Lukáč, Gharbi Michaël, and Jonathan Ragan-Kelley. "Differentiable Vector Graphics Rasterization for Editing and Learning". In: *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 39.6 (2020). DOI: 10.1145/3414685.3417871.

[137] Lev Semenovich Pontryagin. *Mathematical theory of optimal processes*. CRC Press, 1962. DOI: 10.1201/9780203749319.

[138] Antoine McNamara, Adrien Treuille, Zoran Popović, and Jos Stam. "Fluid control using the adjoint method". In: *ACM Trans. Graph. (Proc. SIGGRAPH)*. Vol. 23. 3. ACM. 2004. DOI: 10.1145/1015706.1015744.

[139] Shayan Hoshyari, Hongyi Xu, Espen Knoop, Stelian Coros, and Moritz Bächer. "Vibration-minimizing motion retargeting for robotic characters". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 38.4 (2019). DOI: 10.1145/3306346.3323034.

[140] Aidan N. Gomez, Mengye Ren, Raquel Urtasun, and Roger B. Grosse. "The Reversible Residual Network: Backpropagation without Storing Activations". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017.

[141] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. "Neural ordinary differential equations". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2018.

[142] Laurent Dinh, David Krueger, and Yoshua Bengio. "NICE: Non-linear Independent Components Estimation". In: *Proc. International Conference on Learning Representations (ICLR), Workshop Track*. 2015.

[143] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. "Density estimation using Real NVP". In: *Proc. International Conference on Learning Representations (ICLR), Conference Track*. 2017. arXiv: 1605.08803.

[144] Sebastian Weiss and Rüdiger Westermann. "Differentiable Direct Volume Rendering". In: *IEEE Transactions on Visualization and Computer Graphics (Proc. VIS)*. Vol. 28. 1. 2022. DOI: 10.1109/TVCG.2021.3114769.

[145] Wenzel Jakob. *Enoki: structured vectorization and differentiation on modern processor architectures*. 2019. URL: https://github.com/mitsuba-renderer/enoki (Accessed on 30/08/2022).

[146] Ingo Wald, Sven Woop, Carsten Benthin, Gregory S Johnson, and Manfred Ernst. "Embree: a kernel framework for efficient CPU ray tracing". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 33.4 (2014). DOI: 10.1145/2601097.2601199.

[147] Steven G. Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, and Martin Stich. "OptiX: A General Purpose Ray Tracing Engine". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 29.4 (July 2010). DOI: 10.1145/1778765.1778803.

[148] Ioannis Gkioulekas, Anat Levin, and Todd Zickler. "An evaluation of computational imaging techniques for heterogeneous inverse scattering". In: *European Conference on Computer Vision (ECCV)*. 2016. DOI: 10.1007/978-3-319-46487-9_42.

[149] Dejan Azinović, Tzu-Mao Li, Anton Kaplanyan, and Matthias Nießner. "Inverse Path Tracing for Joint Material and Lighting Estimation". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019. DOI: 10.1109/CVPR.2019.00255.

[150] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. "Perceptual Losses for Real-Time Style Transfer and Super-Resolution". In: *European Conference on Computer Vision (ECCV)*. 2016. DOI: 10.1007/978-3-319-46475-6_43.

[151] Pramook Khungurn, Daniel Schroeder, Shuang Zhao, Kavita Bala, and Steve Marschner. "Matching Real Fabrics with Micro-Appearance Models". In: *ACM Trans. Graph.* 35.1 (Dec. 2015). DOI: 10.1145/2818648.

[152] Cheng Zhang, Bailey Miller, Kai Yan, Ioannis Gkioulekas, and Shuang Zhao. "Path-Space Differentiable Rendering". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 39.4 (July 2020). DOI: 10.1145/3386569.3392383.

[153] Csaba Kelemen, László Szirmay-Kalos, György Antal, and Ferenc Csonka. "A Simple and Robust Mutation Strategy for the Metropolis Light Transport Algorithm". In: *Computer Graphics Forum* 21.3 (2002). DOI: 10.1111/1467-8659.t01-1-00703.

[154] Guillaume Loubet, Nicolas Holzschuch, and Wenzel Jakob. "Reparameterizing discontinuous integrands for differentiable rendering". In: *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 38.6 (Dec. 2019). DOI: 10.1145/3355089.3356510.

[155] John Paisley, David M. Blei, and Michael I. Jordan. "Variational Bayesian Inference with Stochastic Search". In: *Proc. International Conference on Machine Learning (ICML)*. 2012.

[156] Diederik P. Kingma and Max Welling. "Auto-Encoding Variational Bayes". In: *Proc. International Conference on Learning Representations (ICLR), Conference Track*. 2014.

[157] Ronald J Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine Learning* 8.3-4 (May 1992). DOI: 10.1007/978-1-4615-3618-5_2.

[158] A de Lataillade, S Blanco, Y Clergent, J.L Dufresne, M El Hafi, and R Fournier. "Monte Carlo method and sensitivity estimations". In: *Journal of Quantitative Spectroscopy and Radiative Transfer* 75.5 (2002). DOI: 10.1016/S0022-4073(02)00027-4.

[159] Osborne Reynolds. *Papers on mechanical and physical subjects: the sub-mechanics of the universe, Vol. 3*. The University Press, 1903.

[160] Cheng Zhang, Lifan Wu, Changxi Zheng, Ioannis Gkioulekas, Ravi Ramamoorthi, and Shuang Zhao. "A Differential Theory of Radiative Transfer". In: *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 38.6 (Nov. 2019). DOI: 10.1145/3355089.3356522.

[161] Eric Heitz, Jonathan Dupuy, Stephen Hill, and David Neubelt. "Real-Time Polygonal-Light Shading with Linearly Transformed Cosines". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 35.4 (July 2016). DOI: 10.1145/2897824.2925895.

[162] Eric Heitz and Stephen Hill. "Real-Time Line- and Disk-Light Shading with Linearly Transformed Cosines". In: *ACM SIGGRAPH Courses*. 2018.

[163] Cheng Zhang, Zihan Yu, and Shuang Zhao. "Path-Space Differentiable Rendering of Participating Media". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 40.4 (2021). DOI: 10.1145/3450626.3459782.

[164] Kai Yan, Christoph Lassner, Brian Budge, Zhao Dong, and Shuang Zhao. "Efficient Estimation of Boundary Integrals for Path-Space Differentiable Rendering". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 41.4 (July 2022). DOI: 10.1145/3528223.3530080.

[165] Eric Paquette, Pierre Poulin, and George Drettakis. "A Light Hierarchy for Fast Rendering of Scenes with Many Lights". In: *Comp. Graph. Forum (Proc. Eurographics)* 17.3 (1998). DOI: 10.1111/1467-8659.00254.

[166] Bruce Walter, Sebastian Fernandez, Adam Arbree, Kavita Bala, Michael Donikian, and Donald P. Greenberg. "Lightcuts: A Scalable Approach to Illumination". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 24.3 (July 2005). DOI: 10.1145/1073204.1073318.

[167] Alejandro Conty Estevez and Christopher Kulla. "Importance Sampling of Many Lights with Adaptive Tree Splitting". In: *Proc. ACM Comput. Graph. Interact. Tech.* 1.2 (Aug. 2018). DOI: 10.1145/3233305.

[168] Benedikt Bitterli, Chris Wyman, Matt Pharr, Peter Shirley, Aaron Lefohn, and Wojciech Jarosz. "Spatiotemporal reservoir resampling for real-time ray tracing with dynamic direct lighting". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 39.4 (July 2020). DOI: 10.1145/3386569.3392481.

[169] Aaron Hertzmann and Denis Zorin. "Illustrating Smooth Surfaces". In: *SIGGRAPH Comput. Graph.* 2000. DOI: 10.1145/344779.345074.

[170] Shinji Ogaki and Iliyan Georgiev. "Production Ray Tracing of Feature Lines". In: *SIGGRAPH Asia Technical Briefs*. 2018. DOI: 10.1145/3283254.3283273.

[171] Rex West. "Physically-based Feature Line Rendering". In: *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 40.6 (Dec. 2021). DOI: 10.1145/3478513.3480550.

[172] Sai Bangaru, Jesse Michel, Kevin Mu, Gilbert Bernstein, Tzu-Mao Li, and Jonathan Ragan-Kelley. "Systematically Differentiating Parametric Discontinuities". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 40.107 (2021). DOI: 10.1145/3450626.3459775.

[173] Yuting Yang, Connelly Barnes, Andrew Adams, and Adam Finkelstein. "Aδ: Autodiff for Discontinuous Programs - Applied to Shaders". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 41.4 (July 2022).

[174] Luke Anderson, Tzu-Mao Li, Jaakko Lehtinen, and Fr'edo Durand. "Aether: An Embedded Domain Specific Sampling Language for Monte Carlo Rendering". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 36.4 (2017). DOI: 10.1145/3072959.3073704.

[175] Matthew M. Loper and Michael J. Black. "OpenDR: An Approximate Differentiable Renderer". In: *European Conference on Computer Vision (ECCV)*. 2014. DOI: 10.1007/978-3-319-10584-0_11.

[176] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. "Neural 3D Mesh Renderer". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018. DOI: 10.1109/CVPR.2018.00411.

[177] Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. "Soft Rasterizer: A Differentiable Renderer for Image-based 3D Reasoning". In: *IEEE International Conference on Computer Vision (ICCV)* (Oct. 2019). DOI: 10.1109/ICCV.2019.00780.

[178] Samuli Laine, Janne Hellsten, Tero Karras, Yeongho Seol, Jaakko Lehtinen, and Timo Aila. "Modular Primitives for High-Performance Differentiable Rendering". In: *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 39.6 (2020). DOI: 10.1145/3414685.3417861.

[179] Forrester Cole, Kyle Genova, Avneesh Sud, Daniel Vlasic, and Zhoutong Zhang. "Differentiable Surface Rendering via Non-Differentiable Sampling". In: *IEEE International Conference on Computer Vision (ICCV)* (2021). DOI: 10.1109/ICCV48922.2021.00603.

[180] Yang Zhou, Lifan Wu, Ravi Ramamoorthi, and Ling-Qi Yan. "Vectorization for Fast, Analytic, and Differentiable Visibility". In: *ACM Trans. Graph.* 40.3 (July 2021). DOI: 10.1145/3452097.

[181] Shree K. Nayar, Katsushi Ikeuchi, and Takeo Kanade. "Shape from interreflections". In: *Proceedings Third International Conference on Computer Vision*. 1990. DOI: 10.1007/BF00115695.

[182] M.K. Chandraker, F. Kahl, and D.J. Kriegman. "Reflections on the generalized bas-relief ambiguity". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Vol. 1. 2005.

[183] Daniel Hauagge, Scott Wehrwein, Kavita Bala, and Noah Snavely. "Photometric Ambient Occlusion for Intrinsic Image Decomposition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 38.4 (2016). DOI: 10.1109/TPAMI.2015.2453959.

[184] Merlin Nimier-David, Zhao Dong, Wenzel Jakob, and Anton Kaplanyan. "Material and Lighting Reconstruction for Complex Indoor Scenes with Texture-space Differentiable Rendering". In: *EGSR - DL-only track*. 2021.

[185] Fujun Luan, Shuang Zhao, Kavita Bala, and Zhao Dong. "Unified Shape and SVBRDF Recovery using Differentiable Monte Carlo Rendering". In: *Computer Graphics Forum (Proc. EGSR)* 40.4 (2021). DOI: 10.1111/cgf.14344.

[186] Guangyan Cai, Kai Yan, Zhao Dong, Ioannis Gkioulekas, and Shuang Zhao. "Physics-Based Inverse Rendering using Combined Implicit and Explicit Geometries". In: *Computer Graphics Forum (Proc. EGSR)* 41.4 (2022). DOI: 10.1111/cgf.14592.

[187] Jacob Munkberg, Jon Hasselgren, Tianchang Shen, Jun Gao, Wenzheng Chen, Alex Evans, Thomas Mueller, and Sanja Fidler. "Extracting Triangular 3D Models, Materials, and Lighting From Images". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022. arXiv: 2111.12503.

[188] Jiahui Lyu, Bojian Wu, Dani Lischinski, Daniel Cohen-Or, and Hui Huang. "Differentiable Refraction-Tracing for Mesh Reconstruction of Transparent Objects". In: *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 39.6 (2020). DOI: 10.1145/3414685.3417815.

[189] Zhengqin Li, Yu-Ying Yeh, and Manmohan Chandraker. "Through the Looking Glass: Neural 3D Reconstruction of Transparent Shapes". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020. DOI: 10.1109/CVPR42600.2020.00134.

[190] Mojtaba Bemana, Karol Myszkowski, Jeppe Revall Frisvad, Hans-Peter Seidel, and Tobias Ritschel. "Eikonal Fields for Refractive Novel-View Synthesis". In: *Proc. SIGGRAPH (Conference track)*. 2022. DOI: 10.1145/3528233.3530706.

[191] Marc Kassubeck, Moritz Kappel, Susana Castillo, and Marcus Magnor. *N-SfC: Robust and Fast Shape Estimation from Caustic Images*. 2021. arXiv: 2112.06705.

[192] Shinyoung Yi, Donggun Kim, Kiseok Choi, Adrian Jarabo, Diego Gutierrez, and Min H. Kim. "Differentiable Transient Rendering". In: *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 40.6 (2021). DOI: 10.1145/3478513.3480498.

[193] Lifan Wu, Guangyan Cai, Ravi Ramamoorthi, and Shuang Zhao. "Differentiable Time-Gated Rendering". In: *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 40.6 (2021). DOI: 10.1145/3478513.3480489.

[194] Andreas Velten, Thomas Willwacher, Otkrist Gupta, Ashok Veeraraghavan, Moungi G. Bawendi, and Ramesh Raskar. "Recovering three-dimensional shape around a corner using ultrafast time-of-flight imaging". In: *Nature Communications* 3.745 (Mar. 2012). DOI: 10.1038/ncomms1747.

[195] Stanley Osher and James A. Sethian. "Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations". In: *Journal of Computational Physics* 79.1 (1988). DOI: 10.1016/0021-9991(88)90002-2.

[196] Brian Curless and Marc Levoy. "A Volumetric Method for Building Complex Models from Range Images". In: *Annual Conference Series (Proc. SIGGRAPH)*. 1996. DOI: 10.1145/237170.237269.

[197] Hong-Kai Zhao, S. Osher, and R. Fedkiw. "Fast surface reconstruction using the level set method". In: *Proceedings IEEE Workshop on Variational and Level Set Methods in Computer Vision*. 2001.

[198] Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. "KinectFusion: Real-time dense surface mapping and tracking". In: *IEEE International Symposium on Mixed and Augmented Reality*. 2011. DOI: 10.1109/ISMAR.2011.6092378.

[199] Matthias Nießner, Michael Zollhöfer, Shahram Izadi, and Marc Stamminger. "Real-Time 3D Reconstruction at Scale Using Voxel Hashing". In: *ACM Trans. Graph.* 32.6 (Nov. 2013). DOI: 10.1145/2508363.2508374.

[200] Yen-Hsi Richard Tsai, Li-Tien Cheng, Stanley Osher, Paul Burchard, and Guillermo Sapiro. "Visibility and its dynamics in a PDE based implicit framework". In: *Journal of Computational Physics* 199.1 (2004). DOI: 10.1016/j.jcp.2004.02.015.

[201] Pau Gargallo, Emmanuel Prados, and Peter Sturm. "Minimizing the Reprojection Error in Surface Reconstruction from Images". In: *IEEE International Conference on Computer Vision (ICCV)* (2007). DOI: 10.1109/ICCV.2007.4409003.

[202] Yue Jiang, Dantong Ji, Zhizhong Han, and Matthias Zwicker. "SDFDiff: Differentiable Rendering of Signed Distance Fields for 3D Shape Optimization". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020. DOI: 10.1109/CVPR42600.2020.00133.

[203] Shaohui Liu, Yinda Zhang, Songyou Peng, Boxin Shi, Marc Pollefeys, and Zhaopeng Cui. "DIST: Rendering Deep Implicit Signed Distance Function with Differentiable Sphere Tracing". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020. DOI: 10.1109/CVPR42600.2020.00209.

[204] John C. Hart. "Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces". In: *The Visual Computer* 12.10 (Jan. 1996). DOI: 10.1007/s003710050084.

[205] Benjamin Keinert, Henry Schäfer, Johann Korndörfer, Urs Ganse, and Marc Stamminger. "Enhanced Sphere Tracing". In: *Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference*. 2014.

[206] Dario Seyb, Alec Jacobson, Derek Nowrouzezahrai, and Wojciech Jarosz. "Nonlinear sphere tracing for rendering deformed signed distance fields". In: *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 38.6 (Nov. 2019). DOI: 10.1145/3355089.3356502.

[207] Lior Yariv, Yoni Kasten, Dror Moran, Meirav Galun, Matan Atzmon, Basri Ronen, and Yaron Lipman. "Multiview Neural Surface Reconstruction by Disentangling Geometry and Appearance". In: *Advances in Neural Information Processing Systems (NeurIPS)* 33 (2020).

[208] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. "Differentiable Volumetric Rendering: Learning Implicit 3D Representations without 3D Supervision". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020. DOI: 10.1109/CVPR42600.2020.00356.

[209] A. Laurentini. "The visual hull concept for silhouette-based image understanding". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 16.2 (1994). DOI: 10.1109/34.273735.

[210] Ishit Mehta, Manmohan Chandraker, and Ravi Ramamoorthi. "A Level Set Theory for Neural Implicit Evolution under Explicit Flows". In: *European Conference on Computer Vision (ECCV)*. 2022.

[211] William E. Lorensen and Harvey E. Cline. "Marching Cubes: A High Resolution 3D Surface Construction Algorithm". In: *Computer Graphics (Proc. SIGGRAPH)*. 1987. DOI: 10.1145/280811.281026.

[212] Akio Doi and Akio Koide. "An Efficient Method of Triangulating Equi-Valued Surfaces by Using Tetrahedral Cells". In: *IEICE Transactions on Information and Systems* 1.74 (1991).

[213] Edoardo Remelli, Artem Lukoianov, Stephan Richter, Benoit Guillard, Timur Bagautdinov, Pierre Baque, and Pascal Fua. "MeshSDF: Differentiable Iso-Surface Extraction". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 33. 2020.

[214] Tianchang Shen, Jun Gao, Kangxue Yin, Ming-Yu Liu, and Sanja Fidler. "Deep Marching Tetrahedra: a Hybrid Representation for High-Resolution 3D Shape Synthesis". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2021.

[215] Sai Praveen Bangaru, Michaël Gharbi, Tzu-Mao Li, Fujun Luan, Kalyan Sunkavalli, Miloš Hašan, Sai Bi, Zexiang Xu, Gilbert Bernstein, and Frédo Durand. *Differentiable Rendering of Neural SDFs through Reparameterization*. 2022. arXiv: 2206.05344.

[216] Shuang Zhao, Lifan Wu, Frédo Durand, and Ravi Ramamoorthi. "Downsampling Scattering Parameters for Rendering Anisotropic Media". In: *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 35.6 (2016). DOI: 10.1145/2980179.2980228.

[217] Zdravko Velinov, Marios Papas, Derek Bradley, Paulo Gotardo, Parsa Mirdehghan, Steve Marschner, Jan Novák, and Thabo Beeler. "Appearance Capture and Modeling of Human Teeth". In: *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 37.6 (Dec. 2018). DOI: 10.1145/3272127.3275098.

[218] Xi Deng, Fujun Luan, Bruce Walter, Kavita Bala, and Steve Marschner. "Reconstructing Translucent Objects Using Differentiable Rendering". In: *Proc. SIGGRAPH (Conference track)*. 2022. DOI: 10.1145/3528233.3530714.

[219] Chengqian Che, Fujun Luan, Shuang Zhao, Kavita Bala, and Ioannis Gkioulekas. *Inverse Transport Networks*. 2018. arXiv: 1809.10820.

[220] Milovš Hašan and Ravi Ramamoorthi. "Interactive Albedo Editing in Path-Traced Volumetric Materials". In: *ACM Trans. Graph.* 32.2 (Apr. 2013). DOI: 10.1145/2451236.2451237.

[221] Oliver Klehm, Ivo Ihrke, Hans-Peter Seidel, and Elmar Eisemann. "Property and Lighting Manipulations for Static Volume Stylization Using a Painting Metaphor". In: *IEEE Transactions on Visualization and Computer Graphics* 20.7 (2014). DOI: 10.1109/TVCG.2014.13.

[222] Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. "Neural Volumes: Learning Dynamic Renderable Volumes from Images". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 38.4 (July 2019). DOI: 10.1145/3306346.3323020.

[223] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis". In: *European Conference on Computer Vision (ECCV)*. 2020. DOI: 10.1145/3503250.

[224] Alex Yu, Sara Fridovich-Keil, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. *Plenoxels: Radiance Fields without Neural Networks*. 2021. arXiv: 2112.05131.

[225] Cheng Sun, Min Sun, and Hwann-Tzong Chen. "Direct Voxel Grid Optimization: Super-fast Convergence for Radiance Fields Reconstruction". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022.

[226] Animesh Karnewar, Tobias Ritschel, Oliver Wang, and Niloy J. Mitra. "ReLU Fields: The Little Non-linearity That Could". In: *Proc. SIGGRAPH (Conference track)* 41.4 (July 2022). DOI: 10.1145/3528233.3530707.

[227] Sai Bi, Zexiang Xu, Kalyan Sunkavalli, Miloš Hašan, Yannick Hold-Geoffroy, David Kriegman, and Ravi Ramamoorthi. "Deep Reflectance Volumes: Relightable Reconstructions from Multi-view Photometric Images". In: *European Conference on Computer Vision (ECCV)*. 2020. DOI: 10.1007/978-3-030-58580-8_18.

[228] Sai Bi, Zexiang Xu, Pratul Srinivasan, Ben Mildenhall, Kalyan Sunkavalli, Miloš Hašan, Yannick Hold-Geoffroy, David Kriegman, and Ravi Ramamoorthi. *Neural Reflectance Fields for Appearance Acquisition*. 2020. arXiv: 2008.03824 [cs.CV].

[229] Julien N.P. Martel, David B. Lindell, Connor Z. Lin, Eric R. Chan, Marco Monteiro, and Gordon Wetzstein. "ACORN: Adaptive Coordinate Networks for Neural Representation". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* (2021). DOI: 10.1145/3450626.3459785.

[230] Towaki Takikawa, Joey Litalien, Kangxue Yin, Karsten Kreis, Charles Loop, Derek Nowrouzezahrai, Alec Jacobson, Morgan McGuire, and Sanja Fidler. "Neural Geometric Level of Detail: Real-time Rendering with Implicit 3D Shapes". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021. DOI: 10.1109/CVPR46437.2021.01120.

[231] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. "Instant Neural Graphics Primitives with a Multiresolution Hash Encoding". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 41.4 (July 2022). DOI: 10.1145/3528223.3530127.

[232] Yiheng Xie, Towaki Takikawa, Shunsuke Saito, Or Litany, Shiqin Yan, Numair Khan, Federico Tombari, James Tompkin, Vincent Sitzmann, and Srinath Sridhar. "Neural Fields in Visual Computing and Beyond". In: *Computer Graphics Forum (Proc. Eurographics - State of the Art Reports)* (2021). DOI: 10.1111/cgf.14505. arXiv: 2111.11426.

[233] Lior Yariv, Jiatao Gu, Yoni Kasten, and Yaron Lipman. *Volume Rendering of Neural Implicit Surfaces*. 2021. arXiv: 2106.12052.

[234] Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. "NeuS: Learning Neural Implicit Surfaces by Volume Rendering for Multi-view Reconstruction". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2021.

[235] Wang Yifan, Felice Serena, Shihao Wu, Cengiz Öztireli, and Olga Sorkine-Hornung. "Differentiable Surface Splatting for Point-based Geometry Processing". In: *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 38.6 (2019). DOI: 10.1145/3355089.3356513.

[236] Darius Rückert, Linus Franke, and Marc Stamminger. "ADOP: Approximate Differentiable One-Pixel Point Rendering". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 41.4 (July 2022). DOI: 10.1145/3528223.3530122.

[237] Qiangeng Xu, Zexiang Xu, Julien Philip, Sai Bi, Zhixin Shu, Kalyan Sunkavalli, and Ulrich Neumann. "Point-NeRF: Point-Based Neural Radiance Fields". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2022.

[238] Christoph Lassner and Michael Zollhöfer. "Pulsar: Efficient Sphere-based Neural Rendering". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2021. DOI: 10.1109/CVPR46437.2021.00149.

[239] Zongling Li, Qingyu Hou, Zhipeng Wang, Fanjiao Tan, Jin Liu, and Wei Zhang. "End-to-end learned single lens design using fast differentiable ray tracing". In: *Opt. Lett.* 46.21 (Nov. 2021). DOI: 10.1364/OL.442870.

[240] Ethan Tseng, Ali Mosleh, Fahim Mannan, Karl St-Arnaud, Avinash Sharma, Yifan Peng, Alexander Braun, Derek Nowrouzezahrai, Jean-Francois Lalonde, and Felix Heide. "Differentiable Compound Optics and Processing Pipeline Optimization for End-to-end Camera Design". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 40.2 (2021). DOI: 10.1145/3446791.

[241] Marios Papas, Wojciech Jarosz, Wenzel Jakob, Szymon Rusinkiewicz, Wojciech Matusik, and Tim Weyrich. "Goal-Based Caustics". In: *Comp. Graph. Forum (Proc. Eurographics)* 30.2 (June 2011). DOI: 10.1111/j.1467-8659.2011.01876.x.

[242] Yonghao Yue, Kei Iwasaki, Bing-Yu Chen, Yoshinori Dobashi, and Tomoyuki Nishita. "Poisson-Based Continuous Surface Generation for Goal-Based Caustics". In: *ACM Trans. Graph.* 33.3 (June 2014). DOI: 10.1145/2580946.

[243] Yuliy Schwartzburg, Romain Testuz, Andrea Tagliasacchi, and Mark Pauly. "High-contrast Computational Caustic Design". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 33.4 (July 2014). DOI: 10.1145/2601097.2601200.

[244] Anurag Sharma, D Vizia Kumar, and Ajoy K Ghatak. "Tracing rays through graded-index media: a new method". In: *Applied Optics* 21.6 (1982). DOI: 10.1364/AO.21.000984.

[245] Du T. Nguyen, Cameron Meyers, Timothy D. Yee, Nikola A. Dudukovic, Joel F. Destino, Cheng Zhu, Eric B. Duoss, Theodore F. Baumann, Tayyab Suratwala, James E. Smay, and Rebecca Dylla-Spears. "3D-Printed Transparent Glass". In: *Advanced Materials* 29.26 (2017). DOI: 10.1002/adma.201701181.

[246] Arjun Teh, Matthew O'Toole, and Ioannis Gkioulekas. "Adjoint Nonlinear Ray Tracing". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 41.4 (July 2022). DOI: 10.1145/3528223.3530077.

[247] Oskar Elek, Denis Sumin, Ran Zhang, Tim Weyrich, Karol Myszkowski, Bernd Bickel, Alexander Wilkie, and Jaroslav Křivánek. "Scattering-aware Texture Reproduction for 3D Printing". In: *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 36.6 (Nov. 2017). DOI: 10.1145/3130800.3130890.

[248] Denis Sumin, Tobias Rittig, Vahid Babaei, Tim Weyrich, Thomas Nindel, Piotr Didyk, Bernd Bickel, Jaroslav Křivánek, Alexander Wilkie, and Karol Myszkowski. "Geometry-Aware Scattering Compensation for 3D Printing". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* (2019). DOI: 10.1145/3306346.3322992.

[249] Thomas Nindel, Tomáš Iser, Tobias Rittig, Alexander Wilkie, and Jaroslav Křivánek. "A Gradient-Based Framework for 3D Print Appearance Optimization". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 40.4 (July 2021). DOI: 10.1145/3450626.3459844.

[250] Quan Zheng, Vahid Babaei, Gordon Wetzstein, Hans-Peter Seidel, Matthias Zwicker, and Gurprit Singh. "Neural Light Field 3D Printing". In: *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 39.6 (2020). DOI: 10.1145/3414685.3417879.

[251] Carole K. Hayakawa, Jerome Spanier, Frédéric Bevilacqua, Andrew K. Dunn, Joon S. You, Bruce J. Tromberg, and Vasan Venugopalan. "Perturbation Monte Carlo methods to solve inverse photon migration problems in heterogeneous tissues". In: *Opt. Lett.* 26.17 (Sept. 2001). DOI: 10.1364/OL.26.001335.

[252] Ahmad Addoum, Olivier Farges, and Fatmir Asllanaj. "Optical properties reconstruction using the adjoint method based on the radiative transfer equation". In: *Journal of Quantitative Spectroscopy and Radiative Transfer* 204 (2018). DOI: 10.1016/j.jqsrt.2017.09.015.

[253] C. T. Germer, A. Roggan, J. P. Ritz, C. Isbert, D. Albrecht, G. Müller, and H. J. Buhr. "Optical properties of native and coagulated human liver tissue and liver metastases in the near infrared range". In: *Lasers Surg Med* 23.4 (1998). DOI: 10.1002/(SICI)1096-9101(1998)23:4%3C194::AID-LSM2%3E3.0.CO;2-6.

[254] Adithya Pediredla, Matteo Giuseppe Scopellit, Srinivasa Narasimhan, Maysamreza Chamanzar, and Ioannis Gkioulekas. "Optimized Virtual Optical Waveguides Enhance Light Throughput in Scattering Media". In: *under review* (2021). DOI: 10.21203/rs.3.rs-778793/v1.

[255] JM Tregan, S Blanco, J Dauchet, M Hafi, R Fournier, L Ibarrart, P Lapeyre, and N Villefranque. *Convergence issues in derivatives of Monte Carlo null-collision integral formulations: a solution.* 2019. DOI: 10.1016/j.jcp.2020.109463. arXiv: 1903.06508.

[256] Michael B. Giles. *Monte Carlo evaluation of sensitives in computational finance.* Tech. rep. Oxford-Man Institute of Quantitative Finance, 2007.

[257] Homan Igehy. "Tracing Ray Differentials". In: *Annual Conference Series (Proc. SIGGRAPH)*. 1999. DOI: 10.1145/311535.311555.

[258] Greg Ward and Paul Heckbert. *Irradiance gradients*. Tech. rep. Apr. 1992. DOI: 10.1145/1401132.1401225.

[259] Min Chen and James Arvo. "Theory and Application of Specular Path Perturbation". In: *ACM Trans. Graph.* 19.4 (Oct. 2000). DOI: 10.1145/380666.380670.

[260] Ravi Ramamoorthi, Dhruv Mahajan, and Peter Belhumeur. "A first-order analysis of lighting, shading, and shadows". In: *ACM Trans. Graph.* 26.1 (2007). DOI: 10.1145/1189762.1189764.

[261] Don Mitchell and Pat Hanrahan. "Illumination from curved reflectors". In: *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*. 1992. DOI: 10.1145/142920.134082.

[262] Tizian Zeltner, Iliyan Georgiev, and Wenzel Jakob. "Specular Manifold Sampling for Rendering High-Frequency Caustics and Glints". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 39.4 (July 2020). DOI: 10.1145/3386569.3392408.

[263] Binh-Son Hua, Adrien Gruson, Victor Petitjean, Matthias Zwicker, Derek Nowrouzezahrai, Elmar Eisemann, and Toshiya Hachisuka. "A Survey on Gradient-Domain Rendering". In: *Computer Graphics Forum* 38.2 (2019). DOI: 10.1111/cgf.13652.

[264] Tzu-Mao Li, Jaakko Lehtinen, Ravi Ramamoorthi, Wenzel Jakob, and Frédo Durand. "Anisotropic Gaussian Mutations for Metropolis Light Transport through Hessian-Hamiltonian Dynamics". In: *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 34.6 (Nov. 2015). DOI: 10.1145/2816795.2818084.

[265] Fujun Luan, Shuang Zhao, Kavita Bala, and Ioannis Gkioulekas. "Langevin Monte Carlo Rendering with Gradient-based Adaptation". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 39.4 (2020). DOI: 10.1145/3386569.3392382.

[266] Merlin Nimier-David, Sébastien Speierer, Benoît Ruiz, and Wenzel Jakob. "Radiative Backpropagation: An Adjoint Method for Lightning-Fast Differentiable Rendering". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 39.4 (July 2020). DOI: 10.1145/3386569.3392406.

[267] Milan Jaroš, Lubomír Říha, Petr Strakoš, and Matěj Špeťko. "GPU Accelerated Path Tracing of Massive Scenes". In: *ACM Trans. Graph.* 40.2 (Apr. 2021). DOI: 10.1145/3447807.

[268] Marco Manzi, Markus Kettunen, Frédo Durand, Matthias Zwicker, and Jaakko Lehtinen. "Temporal Gradient-Domain Path Tracing". In: *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 35.6 (Nov. 2016). DOI: 10.1145/2980179.2980256.

[269] Daqi Lin, Markus Kettunen, Benedikt Bitterli, Jacopo Pantaleoni, Cem Yuksel, and Chris Wyman. "Generalized Resampled Importance Sampling: Foundations of ReSTIR". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 41.4 (July 2022). DOI: 10.1145/3528223.3530158.

[270] Merlin Nimier-David, Thomas Müller, Alexander Keller, and Wenzel Jakob. "Unbiased Inverse Volume Rendering with Differential Trackers". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 41.4 (July 2022). DOI: 10.1145/3528223.3530073.

[271] Jos Stam and Ryan Schmidt. "On the Velocity of an Implicit Surface". In: *ACM Trans. Graph.* 30.3 (May 2011). DOI: 10.1145/1966394.1966400.

[272] Matan Atzmon, Niv Haim, Lior Yariv, Ofer Israelov, Haggai Maron, and Yaron Lipman. "Controlling neural level sets". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2019.

[273] José Gomes and Olivier Faugeras. "Reconciling Distance Functions and Level Sets". In: *Journal of Visual Communication and Image Representation* 11.2 (2000). DOI: 10.1006/jvci.1999.0439.

[274] Chunming Li, Chenyang Xu, Changfeng Gui, and M.D. Fox. "Level set evolution without re-initialization: a new variational formulation". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Vol. 1. 2005. DOI: 10.1109/CVPR.2005.213.

[275] David Adalsteinsson and James A. Sethian. "A Fast Level Set Method for Propagating Interfaces". In: *Journal of Computational Physics* 118.2 (1995). DOI: 10.1006/jcph.1995.1098.

[276] James A. Sethian. "A fast marching level set method for monotonically advancing fronts". In: *Proceedings of the National Academy of Sciences* 93.4 (1996). DOI: 10.1073/pnas.93.4.1591.

[277] James A. Sethian. "Fast Marching Methods". In: *SIAM Review* 41.2 (1999). DOI: 10.1007/978-3-540-70529-1_379.

[278] Hongkai Zhao. "A fast sweeping method for eikonal equations". In: *Mathematics of Computation* 74 (250 2004). DOI: 10.1090/S0025-5718-04-01678-3.

[279] Miles Detrixhe, Frédéric Gibou, and Chohong Min. "A parallel fast sweeping method for the Eikonal equation". In: *Journal of Computational Physics* 237 (2013). DOI: 10.1016/j.jcp.2012.11.042.

[280] Ken Museth. "Novel Algorithm for Sparse and Parallel Fast Sweeping: Efficient Computation of Sparse Signed Distance Fields". In: *ACM SIGGRAPH Talks*. 2017. DOI: 10.1145/3084363.3085093.

[281] David Adalsteinsson and James A. Sethian. "The Fast Construction of Extension Velocities in Level Set Methods". In: *Journal of Computational Physics* 148.1 (1999). DOI: 10.1006/jcph.1998.6090.

[282] Shingyu Leung and Jianliang Qian. "An adjoint state method for three-dimensional transmission traveltime tomography using first-arrivals". In: *Communications in Mathematical Sciences* 4.1 (2006). DOI: 10.4310/CMS.2006.v4.n1.a10.

[283] Miroslava Slavcheva, Maximilian Baust, and Slobodan Ilic. "SobolevFusion: 3D Reconstruction of Scenes Undergoing Free Non-rigid Motion". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018. DOI: 10.1109/CVPR.2018.00280.

[284] Baptiste Nicolet, Alec Jacobson, and Wenzel Jakob. "Large Steps in Inverse Rendering of Geometry". In: *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 40.6 (Dec. 2021). DOI: 10.1145/3478513.3480501.

[285] Jonathan Dupuy, Eric Heitz, and Eugene d'Eon. "Additional Progress towards the Unification of Microfacet and Microflake Theories". In: *EGSR Experimental Ideas and Implementations*. 2016.

[286] David Luebke, Martin Reddy, Jonathan D. Cohen, Amitabh Varshney, Benjamin Watson, and Robert Huebner. *Level of Detail for 3D Graphics*. Morgan Kaufmann Publishers Inc., 2002.

[287] Alexander Schwank, Callum James James, and Tony Micilotta. "The Trees of The Jungle Book". In: *ACM SIGGRAPH Talks*. 2016. DOI: 10.1145/2897839.2927428.

[288] Anton Kaplanyan, Anton Sochenov, Thomas Leimkuehler, Mikhail Okunev, Todd Goodall, and Rufo Gizem. "DeepFovea: Neural Reconstruction for Foveated Rendering and Video Compression using Learned Statistics of Natural Videos". In: *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 38.4 (2019). DOI: 10.1145/3355089.3356557.

[289] Hugues Hoppe. "Progressive Meshes". In: *SIGGRAPH Comput. Graph.* 1996. DOI: 10.1145/237170.237216.

[290] Guillaume Loubet and Fabrice Neyret. "Hybrid Mesh-Volume LoDs for All-Scale Pre-Filtering of Complex 3D Assets". In: *Computer Graphics Forum* 36.2 (2017). DOI: 10.1111/cgf.13138.

[291] Adrian Jarabo, Carlos Aliaga, and Diego Gutierrez. "A Radiative Transfer Framework for Spatially-Correlated Materials". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 37.4 (July 2018). DOI: 10.1145/3197517.3201282.

[292] Benedikt Bitterli, Srinath Ravichandran, Thomas Müller, Magnus Wrenninge, Jan Novák, Steve Marschner, and Wojciech Jarosz. "A radiative transfer framework for non-exponential media". In: *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 37.6 (2018). DOI: 10.1145/3272127.3275103.

[293] Marc Olano and Dan Baker. "LEAN Mapping". In: *Proc. Symposium on Interactive 3D Graphics and Games (I3D)*. 2010. DOI: 10.1145/1730804.1730834.

[294] Jonathan Dupuy, Eric Heitz, Jean-Claude Iehl, Pierre Poulin, Fabrice Neyret, and Victor Ostromoukhov. "Linear Efficient Antialiased Displacement and Reflectance Mapping". In: *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 32.6 (Nov. 2013). DOI: 10.1145/2508363.2508422.

[295] Lifan Wu, Shuang Zhao, Ling-Qi Yan, and Ravi Ramamoorthi. "Accurate Appearance Preserving Prefiltering for Rendering Displacement-Mapped Surfaces". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 38.4 (July 2019). DOI: 10.1145/3306346.3322936.

[296] Junqiu Zhu, Sizhe Zhao, Lu Wang, Yanning Xu, and Yan Ling-Qi. "Practical Level-of-Detail Aggregation of Fur Appearance". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 41.4 (2022). DOI: 10.1145/3528223.3530105.

[297] Jonathan T. Moon, Bruce Walter, and Stephen R. Marschner. "Rendering Discrete Random Media Using Precomputed Scattering Solutions". In: *Rendering Techniques (Proc. EG Workshop on Rendering)*. 2007.

[298] Johannes Meng, Marios Papas, Ralf Habel, Carsten Dachsbacher, Steve Marschner, Markus Gross, and Wojciech Jarosz. "Multi-scale Modeling and Rendering of Granular Materials". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 34.4 (July 2015). DOI: 10.1145/2766949.

[299] Thomas Müller, Marios Papas, Markus Gross, Wojciech Jarosz, and Jan Novák. "Efficient Rendering of Heterogeneous Polydisperse Granular Media". In: *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 35.6 (Dec. 2016). DOI: 10.1145/2980179.2982429.

[300] Jon Hasselgren, Jacob Munkberg, Jaakko Lehtinen, Miika Aittala, and Samuli Laine. "Appearance-Driven Automatic 3D Model Simplification". In: *EGSR - DL-only track*. 2021.

[301] Cyril Crassin, Fabrice Neyret, Miguel Sainz, Simon Green, and Elmar Eisemann. "Interactive indirect illumination using voxel cone tracing". In: *Computer Graphics Forum* 30.7 (2011). DOI: 10.1111/j.1467-8659.2011.02063.x.

[302] Cyril Crassin, Fabrice Neyret, Sylvain Lefebvre, and Elmar Eisemann. "GigaVoxels: Ray-Guided Streaming for Efficient and Detailed Voxel Rendering". In: *Proc. Symposium on Interactive 3D Graphics and Games (I3D)*. 2009. DOI: 10.1145/1507149.1507152.

[303] Lisa M. Sobierajski and Ricardo S. Avila. "A hardware acceleration method for volumetric ray tracing". In: *Proceedings Visualization '95*. 1995. DOI: 10.1109/VISUAL.1995.480792.

[304] Nate Morrical, Will Usher, Ingo Wald, and Valerio Pascucci. "Efficient Space Skipping and Adaptive Sampling of Unstructured Volumes Using Hardware Accelerated Ray Tracing". In: *Proc. VIS (short paper)*. 2019. DOI: 10.1109/VISUAL.2019.8933539.

[305] Johanna Beyer, Markus Hadwiger, and Hanspeter Pfister. "State-of-the-Art in GPU-Based Large-Scale Volume Visualization". In: *Computer Graphics Forum* 34.8 (Dec. 2015). DOI: 10.1111/cgf.12605.

[306] Eric Heitz and Fabrice Neyret. "Representing Appearance and Pre-Filtering Subpixel Data in Sparse Voxel Octrees". In: *Proc. High-Performance Graphics*. 2012.

[307] Eugene d'Eon. "A Reciprocal Formulation of Nonexponential Radiative Transfer. 1: Sketch and Motivation". In: *Journal of Computational and Theoretical Transport* 47.1-3 (2018). DOI: 10.1080/23324309.2018.1481433.

[308] Eugene d'Eon. "A Reciprocal Formulation of Nonexponential Radiative Transfer. 2: Monte Carlo Estimation and Diffusion Approximation". In: *Journal of Computational and Theoretical Transport* 48.6 (2019). DOI: 10.1080/23324309.2019.1677717.

[309] Eugene d'Eon. *A Reciprocal Formulation of Nonexponential Radiative Transfer. 3: Binary Mixtures*. 2019. arXiv: 1903.08783 [cond-mat.stat-mech].

[310] Anthony Davis and Mark Mineev-Weinstein. "Radiation propagation in random media: From positive to negative correlations in high-frequency fluctuations". In: *Journal of Quantitative Spectroscopy and Radiative Transfer* 112 (Mar. 2011). DOI: 10.1016/j.jqsrt.2010.10.001.

[311] Jie Guo, Yanjun Chen, Bingyang Hu, Ling-Qi Yan, Yanwen Guo, and Yuntao Liu. "Fractional Gaussian Fields for Modeling and Rendering of Spatially-Correlated Media". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 38.4 (July 2019). DOI: 10.1145/3306346.3323031.

[312] Thomas Camminady, Martin Frank, and Edward W. Larsen. "Nonclassical particle transport in heterogeneous materials". In: *International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering*. 2017.

[313] Emanuele Caglioti and Francois Golse. "On the Distribution of Free Path Lengths for the Periodic Lorentz Gas III". In: *Communications in Mathematical Physics* 236.2 (May 2003). DOI: 10.1007/s00220-003-0825-5.

[314] Iliyan Georgiev, Zackary Misso, Toshiya Hachisuka, Derek Nowrouzezahrai, Jaroslav Křivánek, and Wojciech Jarosz. "Integral formulations of volumetric transmittance". In: *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 38.6 (2019). DOI: 10.1145/3355089.3356559.

[315] Fakir S. Nooruddin and Greg Turk. "Simplification and Repair of Polygonal Models Using Volumetric Techniques". In: *IEEE Transactions on Visualization and Computer Graphics* 9.2 (2003). DOI: 10.1109/TVCG.2003.1196006.

[316] Patrick Min. *binvox*. 2004 - 2022. URL: http://www.patrickmin.com/binvox (Accessed on 30/08/2022).

Delio Vicini

Rue de Lausanne 49F, 1020 Renens, Switzerland | +41 78 853 68 76 | delio.vicini@gmail.com | dvicini.github.io

Education

Sep. 2017 – Oct. 2022	**Swiss Federal Institute of Technology in Lausanne (EPFL)** PhD in computer science Thesis: Efficient and Accurate Physically-Based Differentiable Rendering
2015 — 2017	**Swiss Federal Institute of Technology in Zurich (ETH Zurich)** M. Sc. in computer science (visual computing focus track) GPA: 5.92 / 6.00 (graduation with distinction)
2012 — 2015	**University of Bern** B. Sc. in computer science and mathematics, minor in history, GPA: 5.91 / 6.00 (Summa Cum Laude)

Core experience

Sep. 2017 – Oct. 2022	**PhD student, Realistic Graphics Lab (EPFL)** Research on physically-based differentiable rendering, volumetric scene representation, geometry reconstruction and machine learning for rendering. Additionally, I also contributed significantly to the Mitsuba 2/3 open-source research renderer. Supervised by Prof. Wenzel Jakob.
2019	**Research intern, Facebook Reality Labs (5 months)** Internship in the FRL graphics team in Redmond, WA. Work on volumetric scene representations for AR/VR with Anton Kaplanyan. This internship resulted in a SIGGRAPH publication.
2017	**Master thesis, Disney Research / ETH Zurich (6 months)** Master thesis on gradient-domain volumetric path tracing, with Jan Novák and Fabrice Rousselle and supervised by Prof. Markus Gross (grade 6.0/6.0).
2016/2017	**Research intern, Walt Disney Animation Studios / Disney Research (3 months)** Internship on denoising rendered deep images, supervised by Brent Burley and David Adler. This project resulted in a Computer Graphics Forum paper and a patent.
2016	**Research intern, Disney Research (3 months)** Internship on denoising for Monte Carlo rendering using local regression methods, supervised by Jan Novák and Fabrice Rousselle.
2015	**Bachelor thesis, Computer Graphics Group, University of Bern (6 months)** Bachelor thesis "Image Filtering using Halide and a new Denoising Algorithm for Gradient-Domain Rendering", supervised by Prof. Matthias Zwicker and Marco Manzi (grade 6.0/6.0). The work done in this thesis resulted in a Eurographics paper.

Additional experience

2019 – present	**Reviewer** SIGGRAPH 2022, SIGGRAPH Asia 2019/2020/2021, Transactions on Graphics, Computer Graphics Forum, The Visual Computer, Computers & Graphics, MCQMC 2021
2017 – 2021	**Teaching assistant, EPFL** Teaching assistant for «Numerical Methods for Visual Computing» and «Advanced Computer Graphics». Supervision of student projects on: denoising for differentiable rendering, neural path guiding, Monte Carlo PDE solvers, direct light sampling hierarchies, Disney BSDF, and geometry instancing.
2014/2015	**Teaching assistant, University of Bern** Teaching assistant for Analysis 1, Analysis 2 and Computer Architecture

Expertise

Analytical	Computer graphics, physically-based and differentiable rendering, volume rendering, Monte Carlo methods, optimization, denoising, neural networks, differential geometry, real-time rendering
Programming	C++, Python, CUDA, PyBind11, PyTorch, Tensorflow, CMake, MATLAB, Halide, OpenGL, GLSL, Java, C#, HTML, CSS
Tools	Git, Linux, Blender, Maya, Photoshop, Illustrator, Premiere, Nuke, LaTeX
Languages	English (proficient), German (native speaker), French (intermediate)

Honors and awards

Invited speaker at VIS conference (2019, 2021), EPFL EDIC Fellowship (2017), Google Hash Code programming competition finalist (2016), 1st place physically-based simulation project competition (ETH Zurich, 2015), 2nd place rendering competition (ETH Zurich, 2015)

Publications

2022	Differentiable Signed Distance Function Rendering **D. Vicini**, S. Speierer, W. Jakob Transactions on Graphics (Proc. of SIGGRAPH 2022)
2022	Dr.Jit: A Just-In-Time Compiler for Differentiable Rendering W. Jakob, S. Speierer, N. Roussel, **D. Vicini** Transactions on Graphics (Proc. of SIGGRAPH 2022)
2021	Path Replay Backpropagation: Differentiating Light Paths using Constant Memory and Linear Time **D. Vicini**, S. Speierer, W. Jakob Transactions on Graphics (Proc. of SIGGRAPH 2021)
2021	A Non-Exponential Transmittance Model for Volumetric Scene Representations **D. Vicini**, W. Jakob, A. Kaplanyan Transactions on Graphics (Proc. of SIGGRAPH 2021)
2019	Mitsuba 2: A Retargetable Forward and Inverse Renderer M. Nimier-David*, **D. Vicini***, T. Zeltner, W. Jakob (*joint first authors) Transactions on Graphics (Proc. of SIGGRAPH Asia)
2019	A Learned Shape-Adaptive Subsurface Scattering Model **D. Vicini**, V. Koltun, W. Jakob Transactions on Graphics (Proc. of SIGGRAPH 2019)
2018	Denoising Deep Monte Carlo Renderings **D. Vicini**, D. Adler, J. Novák, F. Rousselle, B. Burley Computer Graphics Forum, 2018 (presented at Eurographics 2019)
2016	Regularizing Image Reconstruction for Gradient-Domain Rendering with Feature Patches M. Manzi, **D. Vicini**, M. Zwicker Computer Graphics Forum (Proc. of Eurographics 2016)

Personal details

Born: 30.9.1993 | **Civil status:** single | **Nationality:** Swiss