

Inverse Rendering: From Surfaces to Continuous Formulations

Presented on 10th June 2026

School of Computer and Communication Sciences
Realistic Graphics Lab
Doctoral program in Computer and Communication Sciences

for the award of the degree of Docteur ès Sciences (PhD)

by

Ziyi ZHANG

Accepted on the jury's recommendation

Prof. M. Pauly, jury president
Prof. W. A. Jakob, thesis director
Prof. T.-M. Li, examiner
Prof. S. Zhao, examiner
Prof. B. Bickel, examiner

Abstract

Recent progress in inverse rendering has been driven in large part by methods based on emissive volumes. These methods are robust and scalable, but they place only weak constraints on the physical meaning of the reconstruction. At the other extreme, inverse rendering with explicit surfaces and physically based light transport aims to recover geometry, materials, and lighting with direct physical meaning, but is much harder because visibility changes are discontinuous, Monte Carlo estimates are noisy, and unknown scene factors are tightly coupled.

This thesis focuses on that harder setting, while studying it in relation to nearby formulations that are easier to optimize. It views inverse rendering along two continua: geometry representations, ranging from volumes to explicit surfaces, and image formation models, ranging from stored radiance to physically based transport.

First, the thesis revisits derivatives for physically based surface rendering and derives a simple local formulation of visibility-related terms. This leads to Projective Sampling, which reuses primal transport samples to guide boundary sampling and improves the efficiency of geometry derivatives.

Second, it introduces a distribution-over-surfaces viewpoint that relaxes geometric optimization without giving up a surface output. In the physically based setting, Many-Worlds Inverse Rendering extends surface derivatives into space by evaluating hypothetical surface patches as competing explanations of the observations. In the emissive setting, Radiance Surfaces replaces image-space supervision of radiance volumes with a 5D radiance field loss that more directly encourages convergence to explicit surfaces while retaining much of the speed and robustness of NeRF-style optimization.

Third, the thesis explores intermediate formulations between emissive and fully physically based inversion. Radiance Caching for Differentiable Path Tracing uses a spatially varying blend between cached and material-based estimators to improve robustness for material estimation under unknown illumination. Together, these results suggest that inverse rendering is best understood not as a choice between disconnected methods, but as a continuum of trade-offs. By clarifying derivative transport and developing algorithms that move along these continua in controlled ways, this thesis makes physically meaningful inverse rendering more robust in practice.

Résumé

Les progrès récents du rendu inverse ont été largement portés par des méthodes fondées sur des volumes émissifs. Ces méthodes sont robustes et passent bien à l'échelle, mais elles n'imposent que de faibles contraintes quant à la signification physique de la reconstruction. À l'autre extrême, le rendu inverse avec des surfaces explicites et un transport lumineux physiquement fondé vise à retrouver une géométrie, des matériaux et un éclairage ayant une interprétation physique directe, mais il est nettement plus difficile, car les changements de visibilité sont discontinus, les estimations de Monte Carlo sont bruitées et les facteurs de scène inconnus sont fortement couplés.

Cette thèse se concentre sur ce cadre plus difficile, tout en l'étudiant par rapport à des formulations voisines plus faciles à optimiser. Elle envisage le rendu inverse selon deux continuités : les représentations géométriques, qui vont des volumes aux surfaces explicites, et les modèles de formation de l'image, qui vont de la radiance stockée au transport physiquement fondé. Premièrement, la thèse réexamine les dérivées du rendu surfacique physiquement fondé et établit une formulation locale simple des termes liés à la visibilité. Cela conduit à Projective Sampling, qui réutilise les échantillons primaires de transport pour guider l'échantillonnage des frontières et améliore l'efficacité des dérivées géométriques.

Deuxièmement, elle introduit un point de vue distributionnel sur les surfaces, qui assouplit l'optimisation géométrique sans renoncer à une sortie surfacique. Dans le cadre physiquement fondé, Many-Worlds Inverse Rendering étend les dérivées de surface dans l'espace en évaluant des patches de surface hypothétiques comme explications concurrentes des observations. Dans le cadre émissif, Radiance Surfaces remplace la supervision des volumes de radiance dans l'espace image par une perte sur un champ de radiance 5D, ce qui favorise plus directement la convergence vers des surfaces explicites tout en conservant une grande partie de la vitesse et de la robustesse des optimisations de type NeRF.

Troisièmement, la thèse explore des formulations intermédiaires entre l'inversion émissive et l'inversion entièrement physiquement fondée. Radiance Caching for Differentiable Path Tracing utilise un mélange spatialement variable d'estimateurs mis en cache et d'estimateurs fondés sur les matériaux afin d'améliorer la robustesse de l'estimation des matériaux sous éclairage inconnu. Pris ensemble, ces résultats suggèrent que le rendu inverse se comprend mieux non pas comme un choix entre des méthodes disjointes, mais comme un continuum de compromis. En clarifiant le transport des dérivées et en développant des algorithmes qui se déplacent le long de ces continuités de manière contrôlée, cette thèse rend le rendu inverse physiquement interprétable plus robuste en pratique.

Acknowledgements

First and foremost, I would like to thank my advisor, Wenzel Jakob, for his guidance, trust, and support throughout my Ph.D. I am grateful for the freedom he gave me to explore difficult ideas, and for the care with which he helped turn those ideas into research. Working with him has shaped not only the technical direction of this thesis, but also my understanding of what careful research should look like.

I am also grateful to my thesis jury, Mark Pauly, Bernd Bickel, Tzu-Mao Li, and Shuang Zhao, for their time, feedback, and thoughtful questions. Their work has influenced many parts of the field in which this thesis is situated, and it is a privilege to have had them evaluate this dissertation.

I feel very lucky to have been part of the Realistic Graphics Lab. The lab has been a place of intense paper deadlines, long debugging sessions, unexpected ideas, and many conversations that made the work better and the process more enjoyable. I would like to thank Baptiste Nicolet, Benjamin Chislett, Christian Döring, Delio Vicini, Ekrem Yilmazer, Lovro Nuic, Mandy Xia, Mariia Soroka, Merlin Nimier-David, Miguel Crespo, Nicolas Roussel, Rami Tabbara, Sébastien Speierer, Tizian Zeltner, and Vishal Pani for making RGL such a stimulating and friendly environment. I am particularly thankful to Pauline Raffestin for her help with the administrative side of lab life, which quietly makes so many things possible.

Much of the research presented here was made possible by wonderful collaborators. I am especially thankful to Nicolas Roussel, whose collaboration was central to several of the projects in this thesis. My internship at NVIDIA Research grew into the Radiance Surfaces project, and I am grateful to Thomas Müller, Tizian Zeltner, Merlin Nimier-David, and Fabrice Rousselle for that collaboration. My experience at Google led to the radiance-caching project, and I am grateful to Delio Vicini, Sebastian Winberg, and Stephan Garbin for the ideas, support, and engineering effort that made it possible. I would also like to thank Lovro Nuic, Korbinian Sager, Zichen Wang, Xi Deng, and Steve Marschner for the projects, discussions, and ideas we shared.

I am also grateful to my earlier collaborators, Daniele Panozzo, Teseo Schneider, Zhongshi

Jiang, Yixin Hu, Denis Zorin, and Naoya Yamaguchi, whose work with me before the Ph.D. helped shape my path into research.

I am deeply grateful to my partner, Dongqing Wang, for her love, patience, and support throughout this journey, especially during the stressful weeks leading up to SIGGRAPH deadlines.

Finally, I would like to thank my family for their unconditional love and support. Their encouragement made it possible for me to pursue this path far from home, and I am deeply grateful for the trust they placed in me.

Funding. This thesis received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No. 948846).

Third-party materials. Several figures in this thesis build on third-party scenes, meshes, datasets, images, and textures. I gratefully acknowledge the creators, curators, and distributors of these materials; the applicable licenses are those distributed with the corresponding assets.

Scenes.

Country Kitchen. The scene used in Figures 2.1, 2.10, and 3.1 is based on the scene by Jay-Artist distributed through Benedikt Bitterli’s Rendering Resources and/or the Mitsuba scene collection; its geometry and materials were modified for the figures.

Chapter 7 benchmark scenes. These scenes are based on third-party scenes distributed through Benedikt Bitterli’s Rendering Resources and/or the Mitsuba scene collection, including, where applicable, Bedroom by SlykDrako, Contemporary Bathroom by Mareck, Salle de bain by nacimus, Country Kitchen by Jay-Artist, Modern Living Room / Breakfast Room / Grey & White Room / Wooden Staircase by Wig42, Modern Hall by NewSee2l035, and Veach Ajar / Veach Bidir / Veach MIS by Benedikt Bitterli.

Veach Ajar. The scene used in 7.8 is credited to Benedikt Bitterli.

Meshes.

Stanford Bunny. The model used in Figures 3.3, and related silhouette or geometry examples, is from the Stanford Computer Graphics Laboratory / Stanford 3D Scanning Repository, and was reconstructed from range scans by Greg Turk and Marc Levoy.

Stanford Dragon. Any Stanford Dragon model used in this thesis is likewise credited to the Stanford Computer Graphics Laboratory / Stanford 3D Scanning Repository, unless otherwise noted in the corresponding scene assets.

Fertility. The model used in Figures 4.4 and 5.6, and related examples, is provided courtesy of UU / Utrecht University by the AIM@SHAPE Shape Repository.

Acknowledgements

Filigree. The model used in Figures 4.4, and 4.12, and related examples, is provided courtesy of SensAble Technologies by the AIM@SHAPE Shape Repository.

Dancing Children. The model used in Figure 4.12 and related examples is provided courtesy of IMATI-GE by the AIM@SHAPE Shape Repository.

Botijo. The model used in Figure 4.6 and related examples is from the AIM@SHAPE Shape Repository.

Neptune. The model used in Figures 4.6 and 5.6, and related examples, is provided courtesy of Laurent Saboret, IMATI/INRIA, by the AIM@SHAPE Shape Repository.

Heptoroid. The model used in Figure 5.6 is from the Princeton Suggestive Contour Gallery, where it is listed as originating from the UC Berkeley Rapid Prototyping Project.

Images and datasets.

Monet painting texture. The texture used in Figure 7.9 is based on Claude Monet's *Bridge over a Pond of Water Lilies*, 1899. The image is sourced from The Metropolitan Museum of Art via Wikimedia Commons, photographed/uploaded by Daniel Schwen, and marked as public domain.

Captured-scene datasets. The Chapter 6 examples use the Mip-NeRF 360 dataset by Barron et al. for Figure 6.6 and related examples, the DTU multi-view stereo dataset by Jensen et al. / Aanæs et al., and the BlendedMVS dataset by Yao et al., distributed under the CC BY 4.0 license.

Additional scene assets. Some figures also use third-party textures, environment maps, or mesh assets included with the corresponding scene distributions; where applicable, these materials are credited according to the license files distributed with the original assets.

To my family

Contents

Abstract

Acknowledgements ii

1 Introduction 1

1.1 Motivation	1
1.2 Summary of contributions	3
1.3 List of works	5

2 Physically based rendering 7

2.1 Scope, assumptions, and notation	8
2.2 Radiometry and local light transport	10
2.2.1 Radiance and projected solid angle	10
2.2.2 Pixel measurements	10
2.2.3 Surface scattering and free-space propagation	11
2.2.4 Light sources and endpoint measures	12
2.3 Geometry terms and path space	12
2.3.1 From solid angle to area	13
2.3.2 Visibility and the geometry term	13
2.3.3 Unrolling local transport into paths	14
2.3.4 Path measure and path contribution	15
2.3.5 Path densities and coordinate changes	16
2.3.6 Multiple parameterizations of the same path	17
2.3.7 Smooth and delta path sets	18
2.4 Monte Carlo integration for light transport	18
2.4.1 Basic estimator and error measures	18
2.4.2 Importance sampling	20
2.4.3 Emitter sampling and PDF conversion	20
2.4.4 Multiple importance sampling	22
2.5 Path tracing estimators	22
2.5.1 Throughput and recursive estimation	23
2.5.2 Next-event estimation	23
2.5.3 Russian roulette	24
2.5.4 Path-sampling strategies as path-space proposals	24
2.6 Surface scattering models	26

2.6.1	What a BSDF measures	26
2.6.2	Diffuse, glossy, and ideal specular components	27
2.6.3	BSDF sampling, mixtures, and physical constraints	28
2.7	Volume rendering	29
2.7.1	Medium coefficients, phase functions, and radiative transfer	29
2.7.2	Transmittance and emission–absorption rendering	30
2.7.3	Scattering media and volumetric path tracing	31
2.7.4	Homogeneous and heterogeneous media	32
2.8	Microflake media	33
2.9	Stochastic surfaces	36
2.10	Implications for inverse rendering	38
3	Differentiable PBR	39
3.1	Preliminaries: derivatives of path tracing	39
3.1.1	What does “differentiable rendering” mean here?	40
3.1.2	Challenges in differentiating path tracing	40
3.1.3	A common misconception: differentiable ray tracing	44
3.2	Automatic differentiation in rendering	44
3.2.1	Automatic differentiation	45
3.2.2	Forward and backward modes for renderers	46
3.3	Path replay backpropagation	48
3.3.1	Path contributions and local derivatives	48
3.3.2	Replay-based reverse-mode implementation	50
3.4	A derivative transport viewpoint	53
3.4.1	Differential radiance as a transport quantity	54
3.4.2	Adjoint radiance	56
3.4.3	Path replay backpropagation revisited	58
3.5	Design space in the differential problem	59
3.5.1	Differential sampling strategies	59
3.5.2	Differential light path parameterization	64
3.5.3	A practical implementation viewpoint	70
3.6	Visibility derivatives in physically based rendering	74
3.6.1	A simple example	75
3.6.2	Reynolds’ transport theorem	77
3.6.3	Recursive transport integrals	78
3.6.4	Boundary motion	79
3.6.5	Boundary sampling	81
3.6.6	Boundary derivatives as differential transport	83
3.6.7	Boundary integral reparameterization	84
3.6.8	Other approaches	86
3.6.9	Summary	88
3.7	Optimization bottlenecks beyond derivatives	89

Contents

4	Projective Sampling for Differentiable Rendering of Geometry	91
4.1	A local boundary integral	92
4.2	Method	95
4.2.1	Guiding distribution	96
4.2.2	Projection	99
4.2.3	Fiber curves	103
4.2.4	Implicit functions	103
4.3	Results	105
4.3.1	Experimental setup	105
4.3.2	Derivative estimation	106
4.3.3	End-to-end optimization	106
4.3.4	Implicit and indirect effects	108
4.4	Conclusion	114
5	Many-Worlds Inverse Rendering	115
5.1	Method	118
5.1.1	Motivation	118
5.1.2	Many-worlds derivative transport	119
5.1.3	Primal rendering	122
5.2	Discussion	123
5.2.1	Relation to surface derivatives	123
5.2.2	Relation to volume rendering	125
5.3	Results	128
5.3.1	Core reconstruction behavior	128
5.3.2	Limitations and robustness	131
5.4	Conclusion	133
6	Radiance Surfaces: Surface Optimization with a 5D Radiance Field Loss	135
6.1	Introduction	137
6.2	Method	138
6.2.1	Non-local surface perturbation	138
6.2.2	Radiance field loss	139
6.2.3	Stochastic background surface	142
6.2.4	Volume relaxation	143
6.2.5	Implementation	143
6.3	Results	144
6.3.1	Novel view synthesis	144
6.3.2	Geometry reconstruction	146
6.4	Discussion	147
6.4.1	Choice of background distribution	147
6.4.2	Relation to many-worlds inverse rendering	147
6.4.3	Limitations and future work	148
6.5	Conclusion	153

7	Radiance Caching for Differentiable Path Tracing	155
7.1	Introduction	157
7.2	Related work	159
7.2.1	Radiance caching	159
7.2.2	Differentiable path tracing	159
7.2.3	Radiance caching in inverse rendering	160
7.3	Method	161
7.3.1	Consistency losses	162
7.3.2	Separated estimators and losses	164
7.3.3	Optimizing the blending field	165
7.3.4	Connection to Hadadan et al. [1]	168
7.4	Results	168
7.4.1	Unknown lighting material optimization	169
7.4.2	Design choices	172
7.4.3	Known lighting material optimization	174
7.4.4	Variance-aware optimization of $\alpha(\mathbf{x})$	175
7.4.5	Optimization progress	175
7.4.6	Performance analysis	177
7.5	Conclusion	180
8	Conclusion	181
A	Appendix for Projective Sampling	183
A.1	Derivation of the local formulation	183
A.1.1	Change of variables in 2D	183
A.1.2	Normal velocity in 3D	184
A.1.3	Local formulation: perimeter term	186
A.1.4	Local formulation: interior term	186
A.1.5	Step 1: $dI(\mathbf{x}_c) \rightarrow d\mathbf{u}$ and normal velocity	188
A.1.6	Step 2: $ds \rightarrow d\mathbf{v}$	188
A.1.7	Step 3: $dt \rightarrow d\phi$	189
A.1.8	Assembling the parts	190
B	Appendix for Radiance Surfaces	192
B.1	Derivation of our loss in NeRF form	192
B.2	Design space of the background distribution	195
B.3	Additional experiments and results	196
B.4	Volume relaxation	200
B.5	Implementation details	201
C	Appendix for Radiance Caching	204
C.1	Decorrelated gradient estimator	204

Contents

C.2 Alpha optimization via blended termination losses	205
C.3 Termination strategies for separated losses	206
Bibliography	220
Curriculum Vitae	221

1 Introduction

1.1 Motivation

Rendering is the process of predicting images from a description of a scene. In computer graphics, that description may include geometry, materials, lights, media, and a camera; a rendering algorithm then models how light and visibility determine pixel values from those quantities. The classical use of rendering is the forward problem: choose a scene, compute an image. Inverse rendering turns this process around. Given one or more observed images, it asks which scene parameters could have produced them.

This inverse problem is valuable because images are easy to capture, whereas the quantities often needed downstream—shape, reflectance, illumination, and sometimes participating media—are not directly observed. Successful inverse rendering would make captured scenes editable, relightable, measurable, and usable in simulation, rather than merely viewable from recorded camera positions. Modern methods usually pose the problem as gradient-based optimization: a renderer predicts images from a parameterized scene, and the scene parameters are adjusted until those predictions match observations or task-specific targets. The forward model can be as simple as rasterizing colored primitives or as rich as physically based rendering (PBR), which attempts to account for shadows, interreflections, caustics, and other global-illumination effects.

The difficulty is that different explanations of a scene can produce similar images. A dark patch in an observation might be a change in diffuse albedo, a cast shadow, a glossy reflection of a dark object, a dimmer emitter, or attenuation by participating media. Geometry is similarly ambiguous: a missing contour might be explained by moving a surface, adding a new surface, or changing the occlusion relationship between objects. These ambiguities are already present in simple settings, and they become much harder when light transport couples geometry, materials, and illumination recursively.

A helpful way to organize existing methods is by two modeling choices. The *geometry representation* ranges from volume-like fields, which assign density or occupancy throughout space, to explicit surfaces such as meshes or level sets. The *image-formation model* ranges from directly stored or baked radiance to physically based transport, where color is explained by emitted

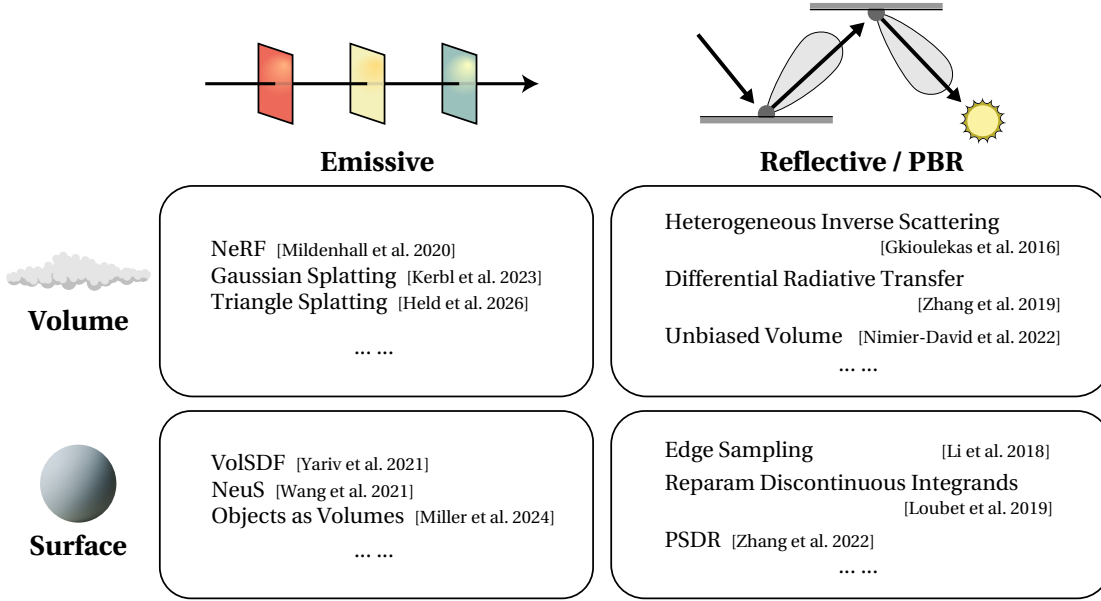


Figure 1.1: A coarse map of inverse rendering methods. One axis describes geometry, ranging from volume-like representations to explicit surfaces. The other describes image formation, ranging from directly stored radiance to reflective, physically based transport. The categories are only approximate, but they capture the trade-offs discussed in this thesis.

light interacting with materials and geometry. Figure 1.1 places representative methods in this design space. This map is not exhaustive, but it captures the trade-offs that shape current inverse rendering methods.

1. **Geometry.** In general, volume representations are easier to optimize than explicit surfaces, primarily due to the nature of their optimization landscape. A volume can place partial density at many locations along a ray, so gradient descent can explore several possible depths and shapes before committing to one. In contrast, a surface usually contributes only where the current interface is already visible or near a visibility boundary. This makes surface gradients local and sensitive to initialization. Surface optimization also struggles when topology must change: creating or removing connected components, opening holes, or merging separate parts cannot be achieved by a small, smooth displacement of the current surface. Volumes are generally easier to differentiate for the same reason: their image formation varies smoothly along rays, while hard surfaces introduce discontinuous occlusion events.
2. **Image formation.** Some methods bake appearance directly into the representation, whereas others explain it physically through light transport. The former approach is typically cheaper and yields lower-variance gradients, leading to cleaner loss estimates and more stable optimization. In PBR, by contrast, pixel values emerge from a tightly coupled process involving geometry, materials, lighting, and possibly media. A color change can be explained by several different physical causes, and those causes interact through shadows, interreflec-

tions, and other indirect effects. This coupling makes optimization slower, noisier, and significantly more non-convex.

For these reasons, much of the recent practical success of inverse rendering can be understood as progress in the upper-left corner of that diagram. Methods that model scenes as emissive volumes, such as NeRF [2] and Gaussian Splatting [3], have become prevalent in many real-world applications of inverse rendering. They are highly effective when the goal is novel view synthesis or when an emissive radiance field is an acceptable final representation. Their success, however, also reflects the simplifications they make. When the task requires explicit surfaces, physically meaningful materials, or indirect cues such as shadows and interreflections, the emissive approximation is no longer sufficient, and one must confront the much harder problem of physically based inverse rendering.

This dissertation is centered on the difficult end of that design space: inverse rendering with explicit surfaces and physically based transport. It also explores the surrounding space along both axes: temporary relaxations of transport, such as radiance caches that store outgoing radiance at surface positions, and temporary reliance on non-explicit surfaces, such as stochastic surfaces. The common question is how far transport or geometry can be relaxed during optimization without giving up physically meaningful outputs.

From this viewpoint, the thesis has three parts. One part revisits derivative computation for physically based surface rendering, where visibility changes and recursive transport make even correct gradients difficult. Two further parts develop algorithmic strategies for moving through the design space in a controlled way: along the surface–volume axis, by retaining the robustness of volumetric optimization while still recovering explicit surfaces, and along the emissive–physically based axis, by introducing intermediate formulations between directly stored radiance and full transport.

1.2 Summary of contributions

This thesis makes three main contributions.

First, we revisit derivative computation in physically based rendering. In emissive volume methods, it is often possible to differentiate the forward simulation more or less directly using automatic differentiation. The surface case is much harder: visibility changes introduce discontinuities, and recursive light transport makes those discontinuities appear throughout path space. We develop a transport view of these geometry derivatives, separating ordinary, smooth path derivatives from visibility-boundary contributions. This leads to *Projective Sampling*, a method that reuses importance-sampled paths from primal rendering as seeds for derivative computation and projects them onto nearby visibility boundaries. The resulting estimator improves the efficiency of geometry derivatives in physically based rendering.

Second, we study how to move along the surface–volume axis without giving up the advantages

Chapter 1. Introduction

of either representation. In practice, volume-based methods are often more robust than surface-based ones, even in applications where the final object is clearly meant to be a surface. To address this, we introduce a viewpoint based on a *distribution over surfaces*, in which optimization is performed on a relaxed representation while the final output remains surface-based. We develop this idea in two settings. In the physically based case, *Many-Worlds Inverse Rendering* extends surface derivatives into space by treating hypothetical surface patches as independent, competing explanations of the observations. In the emissive case, *Radiance Surfaces* applies the same intuition in a simpler setting by supervising a 5D radiance field directly, rather than through a 2D image-space projection. Together, these chapters show that one can retain much of the robustness of volumetric optimization while still targeting explicit surfaces.

Third, we explore the axis between emissive and physically based inverse rendering. The two paradigms form a continuum of representations. A path tracer that terminates at depth k and queries cached radiance already defines a discrete family of intermediate models: depth 0 reduces to emissive rendering, while large values of k approach full PBR. This suggests using intermediate formulations that preserve some of the optimization behavior of emissive methods while still recovering physically meaningful parameters. As a first step in that direction, *Radiance Caching for Differentiable Path Tracing* introduces a spatially varying blending field that interpolates between a learned cache estimator and a standard material estimator. This approach reduces variance and improves conditioning in material estimation under unknown illumination, and it provides a practical mechanism to move between easy-to-optimize radiance representations and fully recursive light transport.

1.3 List of works

This thesis is primarily based on four publications, each included as a separate chapter.

Core thesis publications

- 2023 **Projective Sampling for Differentiable Rendering of Geometry** [4]
Ziyi Zhang, Nicolas Roussel, and Wenzel Jakob.
ACM Transactions on Graphics (SIGGRAPH Asia), 2023.
- 2025 **Many-Worlds Inverse Rendering** [5]
Ziyi Zhang, Nicolas Roussel, and Wenzel Jakob.
ACM Transactions on Graphics (SIGGRAPH Asia), 2025.
- 2025 **Radiance Surfaces: Optimizing Surface Representations with a 5D Radiance Field Loss** [6]
Ziyi Zhang, Nicolas Roussel, Thomas Muller, Tizian Zeltner, Merlin Nimier-David, Fabrice Rousselle, and Wenzel Jakob.
SIGGRAPH Conference Papers, 2025.
- 2026 **Radiance Caching for Differentiable Path Tracing** [7]
Ziyi Zhang, Delio Vicini, Sebastian Winberg, Stephan Garbin, and Wenzel Jakob.
ACM Transactions on Graphics (SIGGRAPH), 2026.

Other publications

During the Ph.D., the author was also involved in the following projects, which are related to inverse rendering but are not included as chapters.

- 2024 **A Simple Approach to Differentiable Rendering of SDFs** [8]
Zichen Wang, Xi Deng, Ziyi Zhang, Wenzel Jakob, and Steve Marschner.
SIGGRAPH Asia Conference Papers, 2024.
- 2026 **Inverse Rendering for Discrete X-Ray Computed Tomography** [9]
Lovro Nuic, Ziyi Zhang, Korbinian Sager, and Wenzel Jakob.
ACM Transactions on Graphics (SIGGRAPH), 2026.

2 Physically based rendering

This chapter introduces the forward rendering model used throughout the thesis. Its purpose is not to give a complete treatment of physically based rendering, but to establish the precise concepts that later chapters differentiate, relax, or approximate. For comprehensive accounts of primal rendering, see standard references such as *Physically Based Rendering: From Theory to Implementation* by Pharr, Jakob, and Humphreys [10] and Veach’s thesis [11].

The chapter first defines how radiance is measured and transported locally and then rewrites these local relations as path-space integrals and Monte Carlo estimators. This order matters for inverse rendering: derivatives act on the same measures, visibility terms, sampling densities, and local path factors that define the forward estimator.

The differentiable rendering methods in Chapter 3 keep this same forward model fixed. They ask for derivatives of the pixel values with respect to scene parameters, rather than for the pixel values alone. For that reason, this chapter pays special attention to measures, sampling densities, visibility, and path factorizations: these details are exactly where differentiating a renderer becomes subtle. Figure 2.1 summarizes the forward map that this chapter formalizes.



Figure 2.1: **Forward rendering maps scene parameters to pixel values.** Geometry, materials, media, emitters, and the sensor define a radiance field. A renderer evaluates light transport through the scene and then applies a sensor measurement to obtain pixel values.

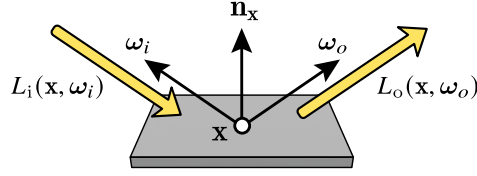


Figure 2.2: **Direction convention.** Both incident and outgoing directions are represented as vectors pointing away from the scattering point. This convention keeps BSDF arguments, transport equations, and path-space formulas consistent.

2.1 Scope, assumptions, and notation

We work in the standard *ray-optics* regime. Light travels along rays, interactions are local, and transport is in steady state: the radiance field does not depend on time. We ignore diffraction, interference, polarization, fluorescence, and transient effects. These approximations are the usual ones behind path tracing and are sufficient for the phenomena considered in this thesis, including shadows, interreflections, caustics, volumetric attenuation, and multiple scattering.

A scene consists of four kinds of components. A *sensor* measures incoming radiance and turns it into pixels. *Emitters* inject radiance into the scene. *Surfaces* scatter light according to BSDFs. When present, *participating media* absorb, emit, and scatter light along ray segments. Most equations are written for a single wavelength or a single color channel. In practice, the same estimators can be evaluated independently for RGB channels or embedded in a full spectral formulation.

Direction convention. Positions are denoted by $\mathbf{x} \in \mathbb{R}^3$ and directions by unit vectors $\boldsymbol{\omega} \in \mathcal{S}^2$. Unless stated otherwise, both are expressed in world coordinates. At a surface point $\mathbf{x}$, the geometric normal is $\mathbf{n}_x$. The directions $\boldsymbol{\omega}_i$ and $\boldsymbol{\omega}_o$ denote incident and outgoing directions at the same point. Both directions are represented as vectors pointing *away from* that point. Thus $L_i(\mathbf{x}, \boldsymbol{\omega}_i)$ means radiance arriving at $\mathbf{x}$ from the side indicated by $\boldsymbol{\omega}_i$; equivalently, that light traveled in direction $-\boldsymbol{\omega}_i$ just before reaching $\mathbf{x}$. The outgoing radiance $L_o(\mathbf{x}, \boldsymbol{\omega}_o)$ leaves $\mathbf{x}$ in direction $\boldsymbol{\omega}_o$.

This convention is common in rendering because the incoming and outgoing directions at a surface can then be drawn in the same local coordinate frame. It also avoids a recurring source of sign errors in path-space formulas: at an interior path vertex, both adjacent directions point away from the vertex, one toward the previous vertex and one toward the next vertex. Figure 2.2 illustrates this convention at a surface interaction.

Symbol	Meaning	Measure / unit
<i>Geometry and directions</i>		
$\mathbf{x}, \mathbf{y}$	Points in space; usually surface points.	$\mathbb{R}^3$
$\mathbf{n}_{\mathbf{x}}$	Geometric normal at $\mathbf{x}$.	unit vector
$\boldsymbol{\omega}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o$	Unit directions; incident and outgoing directions point away from the event.	$\mathcal{S}^2$
$d\boldsymbol{\omega}$	Solid-angle element.	steradian
$d\boldsymbol{\omega}_{\mathbf{x}}^{\perp}$	Projected solid angle, $ \mathbf{n}_{\mathbf{x}} \cdot \boldsymbol{\omega} d\boldsymbol{\omega}$.	projected steradian
$dA(\mathbf{x})$	Surface area element.	m^2
<i>Radiance, scattering, and measurement</i>		
L, L_i, L_o	Radiance; incoming and outgoing variants.	$\text{W m}^{-2} \text{sr}^{-1}$
L_e, L_s	Emitted and surface-scattered radiance.	radiance
$W_e^{(j)}$ or W_j	Sensor response for pixel j ; adjoint importance.	scene-dependent weight
$f_s(\mathbf{x}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o)$	BSDF at $\mathbf{x}$.	sr^{-1} , smooth case
$f_p(\boldsymbol{\omega}_i, \boldsymbol{\omega}_o)$	Medium phase function.	sr^{-1}
P_e	Total emitted power, i.e. radiant flux.	W
<i>Path space and Monte Carlo</i>		
$V(\mathbf{x}, \mathbf{y})$	Visibility between $\mathbf{x}$ and $\mathbf{y}$.	$\{0, 1\}$
$G(\mathbf{x}, \mathbf{y})$	Geometry and visibility segment factor.	m^{-2}
$\bar{\mathbf{x}}$	Complete path, e.g. $(\mathbf{x}_0, \dots, \mathbf{x}_k)$.	variable dimension
$\mathcal{P}_k, \mathcal{P}$	Fixed-length and full path spaces.	disjoint union
μ_k, μ	Corresponding path measures.	area products
$p(\bar{\mathbf{x}})$	Path density with respect to μ .	inverse path measure
p_A, p_{ω}	Area and solid-angle densities.	$\text{m}^{-2} / \text{sr}^{-1}$
β	Path throughput.	weighted product
<i>Participating media</i>		
μ_a, μ_s, μ_t	Absorption, scattering, and extinction; $\mu_t = \mu_a + \mu_s$.	m^{-1}
$T(\mathbf{x}, \mathbf{y})$	Transmittance from $\mathbf{x}$ to $\mathbf{y}$.	dimensionless
ω	Single-scattering albedo, μ_s / μ_t .	$[0, 1]$
α_i	Interval opacity in alpha compositing.	$[0, 1]$
<i>Surface-volume bridge models</i>		
$\alpha(\mathbf{x})$	Occupancy or candidate-existence probability.	$[0, 1]$

Table 2.1: **Notation.** Symbols are grouped by role; densities are always defined with respect to the measure shown or stated in the surrounding text. The symbol α is also used later for cache blending fields; the local context distinguishes opacity, occupancy, and estimator blending.

2.2 Radiometry and local light transport

2.2.1 Radiance and projected solid angle

The basic transported quantity is *radiance*. Intuitively, radiance measures how much light travels through a point in a particular direction. More precisely, at a surface point it is power per unit projected area and per unit solid angle, with units $\text{W m}^{-2} \text{sr}^{-1}$. The word *projected* is important: a surface tilted away from a direction presents a smaller cross-section to the light. For a parallel beam crossing a small surface patch $dA(\mathbf{x})$ in direction $\boldsymbol{\omega}$, the effective area seen by the beam is $|\mathbf{n}_{\mathbf{x}} \cdot \boldsymbol{\omega}| dA(\mathbf{x})$. This geometric projection is the source of the cosine factors that appear throughout rendering equations.

Let $d\boldsymbol{\omega}$ denote differential solid angle and let $dA(\mathbf{x})$ be differential surface area at $\mathbf{x}$. The projected solid-angle measure at $\mathbf{x}$ is

$$d\boldsymbol{\omega}_{\mathbf{x}}^{\perp} := |\mathbf{n}_{\mathbf{x}} \cdot \boldsymbol{\omega}| d\boldsymbol{\omega}. \quad (2.1)$$

The cosine factor is therefore part of the measure rather than part of the integrand. For an opaque one-sided surface, the integration domain is the hemisphere where the relevant cosine is positive. The absolute value in Equation (2.1) keeps the notation compact for two-sided surfaces, transmission, and path-space formulas in which orientations may vary. Figure 2.3 separates the radiometric quantities used below: flux is total power, irradiance is incident power per unit area after integrating directions, and radiance is the density before either the area or direction differential has been integrated out.

2.2.2 Pixel measurements

The measurement equation gives the scalar value stored for a pixel as an integral over sensor position and incident direction. This separates the radiance field from the camera model: radiance is the physical quantity being transported through the scene, while the sensor specifies which part of that field contributes to a particular stored value. For pixel j , let I_j be its value and let A be the sensor surface. We denote the sensor response function for that pixel by $W_e^{(j)}(\mathbf{x}, \boldsymbol{\omega}_i)$. It accounts for the pixel footprint, reconstruction filter, exposure, lens mapping, aperture effects, and any conversion from incident radiance to the stored image value. The same function is also called the pixel's *importance* function when it is used in an adjoint or reverse transport interpretation: it tells how strongly radiance arriving at sensor position $\mathbf{x}$ from direction $\boldsymbol{\omega}_i$ contributes to I_j . The measurement equation is

$$I_j = \int_A \int_{\mathcal{S}^2} W_e^{(j)}(\mathbf{x}, \boldsymbol{\omega}_i) L_i(\mathbf{x}, \boldsymbol{\omega}_i) d\boldsymbol{\omega}_{\mathbf{x}}^{\perp} dA(\mathbf{x}). \quad (2.2)$$

This equation is useful for two reasons. First, it makes clear that the pixel value is a continuous measurement rather than a point sample of radiance. Second, it shows that a pixel is a linear functional of the incident radiance field. The importance interpretation of $W_e^{(j)}$ will

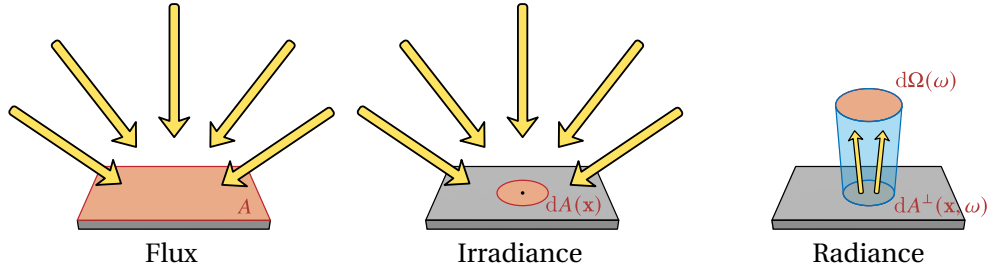


Figure 2.3: **Flux, irradiance, and radiance.** Flux is total power. Irradiance integrates incoming radiance over directions at a receiving point. Radiance keeps both differentials: projected area and solid angle. Pixel measurements later add a sensor response and integrate the same radiance field over the sensor.

be used again after path-space reciprocity is introduced. Later chapters often abbreviate the same sensor-side importance as W_j , or as W_i when it is paired with incident radiance in boundary-integral formulas.

2.2.3 Surface scattering and free-space propagation

A surface converts incident radiance into outgoing radiance through a bidirectional scattering distribution function (BSDF) f_s , defined formally in Section 2.6. At a point $\mathbf{x}$, the surface-scattered radiance toward ω_o is

$$L_s(\mathbf{x}, \omega_o) = \int_{\mathcal{S}^2} f_s(\mathbf{x}, \omega_i, \omega_o) L_i(\mathbf{x}, \omega_i) d\omega_{\mathbf{x}}^\perp. \quad (2.3)$$

Informally, the BSDF describes how strongly light arriving from each incident direction contributes to the chosen outgoing direction. For opaque reflection, the effective domain is a hemisphere; for transmission or two-sided materials, it is convenient to write the integral over the full sphere.

Some surfaces are themselves emitters. The outgoing radiance is then the sum of emitted and scattered radiance:

$$L_o(\mathbf{x}, \omega_o) = L_e(\mathbf{x}, \omega_o) + L_s(\mathbf{x}, \omega_o). \quad (2.4)$$

Here L_e is radiance produced locally by an emitter, while L_s is radiance redirected from other parts of the scene.

In vacuum, radiance is constant along an unoccluded ray segment. Let $r(\mathbf{x}, \omega)$ be the first visible scene point reached by tracing a ray from $\mathbf{x}$ in direction ω . If the ray misses all geometry, r is understood to refer to an environment emitter at infinity. Then the incident radiance at $\mathbf{x}$ equals the outgoing radiance at the first hit point in the opposite direction:

$$L_i(\mathbf{x}, \omega) = L_o(r(\mathbf{x}, \omega), -\omega). \quad (2.5)$$

Together, Equations (2.3)–(2.5) state the recursive rendering equation: outgoing radiance at one surface depends on incident radiance from other surfaces, and that incident radiance in turn depends on their outgoing radiance.

2.2.4 Light sources and endpoint measures

Emitters are most naturally described by emitted radiance L_e . For an area emitter occupying a surface A_L , the total emitted power (radiant flux) is

$$P_e = \int_{A_L} \int_{\mathcal{S}^2} L_e(\mathbf{x}, \boldsymbol{\omega}) d\boldsymbol{\omega}_{\mathbf{x}}^{\perp} dA(\mathbf{x}), \quad (2.6)$$

where the projected solid-angle measure accounts for the cosine between the emitting surface normal and the emitted direction. Because the integral is written over the full sphere with the absolute projected measure, a one-sided emitter is represented by setting L_e to zero on the non-emitting side. Equivalently, one may integrate only over the emitting hemisphere for such lights. This expression is the emitter-side analogue of the sensor measurement equation: a sensor integrates arriving radiance weighted by its response, while an emitter integrates outgoing radiance to produce power.

The measure used at an emitter endpoint depends on the type of light source. An area light is sampled in surface-area measure $dA(\mathbf{x})$ and direction measure $d\boldsymbol{\omega}_{\mathbf{x}}^{\perp}$. An environment light is usually sampled by direction on the sphere. Point lights, directional lights, and ideal laser-like emitters are singular sources: they are represented by delta measures rather than ordinary densities with respect to area or solid angle. This distinction is not only a matter of implementation. It changes which paths have nonzero measure and therefore which probability densities are valid in a path-space estimator.

When a renderer samples a light, it must report a PDF with respect to the same measure used by the estimator. For example, sampling a point $\mathbf{y}$ on an area light gives an area density $p_A(\mathbf{y})$. Sampling an environment direction gives a solid-angle density $p_{\omega}(\boldsymbol{\omega})$. Sampling a point light usually gives a discrete probability of choosing that light, together with a deterministic geometric connection from the shading point to the light position. Later, when several proposals are combined by multiple importance sampling, these quantities must all be converted to a common measure before their densities can be compared.

2.3 Geometry terms and path space

The previous section described transport locally: a sensor measures radiance, surfaces scatter radiance, emitters produce radiance, and ray tracing transports radiance between visible points. The same light transport can be written in several equivalent coordinate systems. Directional form is natural for BSDFs and phase functions. Area form is natural for surface endpoints and light sampling. Path-space form composes all local factors into a single integral

over complete transport histories.

This section moves through those forms in that order. The goal is not to introduce a different physical model, but to make the measures explicit before Monte Carlo estimators are introduced. This is the bookkeeping that makes later statements about path densities, multiple importance sampling, and differentiable rendering precise.

2.3.1 From solid angle to area

Rendering algorithms frequently sample points on surfaces rather than directions. For example, next-event estimation samples a point on a light source, while BSDF sampling samples a direction that may later hit that light. These two procedures can describe the same geometric segment, but their probability densities are expressed in different measures. The bridge between them is the area–solid-angle change of variables.

For two surface points $\mathbf{x}$ and $\mathbf{y}$, define the direction from $\mathbf{x}$ to $\mathbf{y}$ as

$$\omega_{\mathbf{x} \rightarrow \mathbf{y}} := \frac{\mathbf{y} - \mathbf{x}}{\|\mathbf{y} - \mathbf{x}\|}. \quad (2.7)$$

A differential patch $dA(\mathbf{y})$ subtends a solid angle

$$d\omega = \frac{|\mathbf{n}_{\mathbf{y}} \cdot (-\omega_{\mathbf{x} \rightarrow \mathbf{y}})|}{\|\mathbf{x} - \mathbf{y}\|^2} dA(\mathbf{y}). \quad (2.8)$$

Combining this with the projected-solid-angle factor at $\mathbf{x}$ gives

$$d\omega_{\mathbf{x}}^{\perp} = \frac{|\mathbf{n}_{\mathbf{x}} \cdot \omega_{\mathbf{x} \rightarrow \mathbf{y}}| |\mathbf{n}_{\mathbf{y}} \cdot (-\omega_{\mathbf{x} \rightarrow \mathbf{y}})|}{\|\mathbf{x} - \mathbf{y}\|^2} dA(\mathbf{y}). \quad (2.9)$$

Equation (2.9) is the source of the familiar “cosine times cosine divided by squared distance” factor. It is also the simplest example of a principle that will appear repeatedly: changing the coordinates used to describe the same path changes the density by a Jacobian. Figure 2.4 visualizes the geometry behind this change of variables.

2.3.2 Visibility and the geometry term

Let $V(\mathbf{x}, \mathbf{y}) \in \{0, 1\}$ indicate whether the open segment between $\mathbf{x}$ and $\mathbf{y}$ is unobstructed. It is convenient to group visibility, orientation, and distance into the symmetric geometry term

$$G(\mathbf{x}, \mathbf{y}) = V(\mathbf{x}, \mathbf{y}) \frac{|\mathbf{n}_{\mathbf{x}} \cdot \omega_{\mathbf{x} \rightarrow \mathbf{y}}| |\mathbf{n}_{\mathbf{y}} \cdot (-\omega_{\mathbf{x} \rightarrow \mathbf{y}})|}{\|\mathbf{x} - \mathbf{y}\|^2}. \quad (2.10)$$

The geometry term is a physical segment factor in an area-measure path contribution. In forward rendering, the visibility factor is evaluated during ray traversal; it sets blocked segments to zero before being multiplied into the same segment factor. Its nonsmooth dependence on

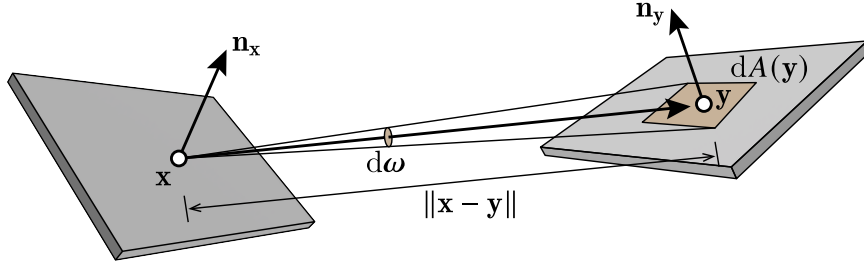


Figure 2.4: **Converting between area and solid angle.** A differential patch $dA(y)$ viewed from x subtends a solid angle $d\omega$. The projected area factors at both endpoints and the inverse-square distance form the geometry term $G(x, y)$. This conversion is essential when combining emitter sampling in area measure with BSDF sampling in solid-angle measure.

geometry is deferred to the differentiable-rendering chapter. Figure 2.5 previews how these segment factors combine with endpoint and scattering factors along complete paths.

2.3.3 Unrolling local transport into paths

The rendering equation is recursive. Starting from the measurement equation and substituting the surface scattering equation once yields direct sensor–light contributions plus terms with one intermediate scattering vertex. Substituting again produces terms with two scattering vertices. Repeating this process yields an infinite series of integrals of increasing dimension. Path space is a compact notation for this entire series.

A light path is written

$$\bar{\mathbf{x}} = (\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_k), \quad (2.11)$$

where $\mathbf{x}_0$ lies on the sensor, $\mathbf{x}_k$ lies on an emitter, and the intermediate vertices are scattering events. The integer k is the number of path segments. Thus a length-one path directly connects sensor and emitter, a length-two path has one interior scattering event, and longer paths have more bounces. For surface-only transport, all vertices lie on surfaces; with participating media, some interior vertices are integrated with respect to volume measure instead.

For a fixed number of segments k , define the fixed-length path space

$$\mathcal{P}_k = \{(\mathbf{x}_0, \dots, \mathbf{x}_k) \mid \mathbf{x}_0 \in A, \mathbf{x}_k \in A_L, \mathbf{x}_1, \dots, \mathbf{x}_{k-1} \in \mathcal{M}\}, \quad (2.12)$$

where A is the sensor surface, A_L denotes emitting surfaces, and $\mathcal{M}$ denotes the scene surfaces. This smooth definition covers finite area emitters. Environment lights, point lights, and other singular sources are incorporated by adding directional, discrete, or delta components to the path measure. The full path space is the disjoint union over all path lengths,

$$\mathcal{P} = \bigsqcup_{k=1}^{\infty} \mathcal{P}_k. \quad (2.13)$$

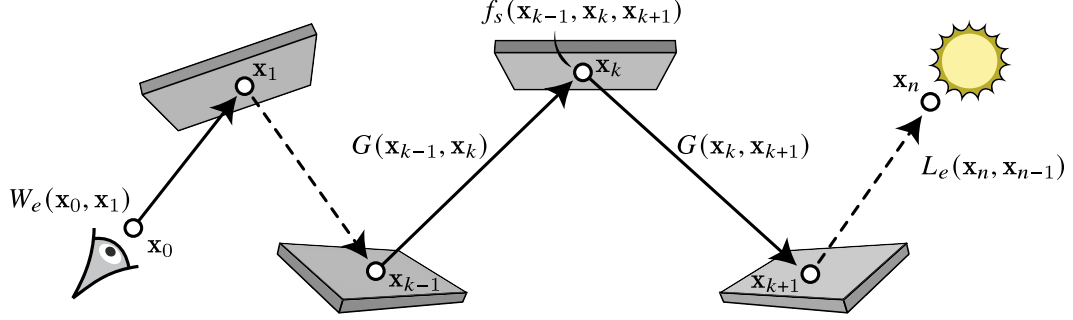


Figure 2.5: **Local factorization of a path contribution.** A path $\bar{\mathbf{x}} = (\mathbf{x}_0, \dots, \mathbf{x}_k)$ contributes through a product of local terms: sensor response at the camera end, BSDF or phase-function evaluations at interior vertices, geometric coupling and visibility between adjacent vertices, optional transmittance through media, and emission at the light end. Rendering algorithms differ mainly in how they sample such paths, not in the physical factors being multiplied.

This union is conceptually important: path space is not a single fixed-dimensional Euclidean domain. It is a collection of domains of different dimensions, one for each possible number and type of interactions.

2.3.4 Path measure and path contribution

For a surface-only path of fixed length k , the natural area measure is

$$d\mu_k(\bar{\mathbf{x}}) = dA(\mathbf{x}_0) dA(\mathbf{x}_1) \cdots dA(\mathbf{x}_k). \quad (2.14)$$

The full path measure combines the fixed-length measures, which lets us write

$$\int_{\mathcal{P}} f_j(\bar{\mathbf{x}}) d\mu(\bar{\mathbf{x}}) := \sum_{k=1}^{\infty} \int_{\mathcal{P}_k} f_{j,k}(\mathbf{x}_0, \dots, \mathbf{x}_k) d\mu_k(\mathbf{x}_0, \dots, \mathbf{x}_k). \quad (2.15)$$

The subscript k on $f_{j,k}$ simply indicates that paths of different lengths have different numbers of arguments. Most of the time we suppress it and write $f_j(\bar{\mathbf{x}})$.

For a surface path, define

$$\boldsymbol{\omega}_i := \boldsymbol{\omega}_{\mathbf{x}_i \rightarrow \mathbf{x}_{i+1}} = \frac{\mathbf{x}_{i+1} - \mathbf{x}_i}{\|\mathbf{x}_{i+1} - \mathbf{x}_i\|}, \quad i = 0, \dots, k-1. \quad (2.16)$$

At an interior vertex $\mathbf{x}_i$, the direction toward the next vertex is $\boldsymbol{\omega}_i$, while the direction toward the previous vertex is $-\boldsymbol{\omega}_{i-1}$. A typical area-measure contribution for pixel j factors as

$$f_j(\bar{\mathbf{x}}) = W_e^{(j)}(\mathbf{x}_0, \boldsymbol{\omega}_0) \left[\prod_{i=1}^{k-1} f_s(\mathbf{x}_i, \boldsymbol{\omega}_i, -\boldsymbol{\omega}_{i-1}) \right] \times \left[\prod_{i=0}^{k-1} G(\mathbf{x}_i, \mathbf{x}_{i+1}) \right] L_e(\mathbf{x}_k, -\boldsymbol{\omega}_{k-1}). \quad (2.17)$$

Chapter 2. Physically based rendering

Equation (2.17) is written for smooth surface vertices and area-measure endpoints. Delta events still use the same surface path domain, but impose local directional constraints rather than ordinary smooth BSDF factors. Point lights and medium vertices require discrete or volume-measure factors, but the same product structure remains. The endpoint factors are the sensor response $W_e^{(j)}$ and emission L_e . The vertex factors are BSDFs or, when a vertex lies in a medium, phase functions. The segment factors are geometry, visibility, and, in media, transmittance. The path contribution is therefore a product of local terms, and the path integral sums this product over all possible vertex locations.

The pixel value is now

$$I_j = \int_{\mathcal{P}} f_j(\bar{\mathbf{x}}) d\mu(\bar{\mathbf{x}}) = \sum_{k=1}^{\infty} \int_{\mathcal{P}_k} f_{j,k}(\bar{\mathbf{x}}) d\mu_k(\bar{\mathbf{x}}). \quad (2.18)$$

Equation (2.18) is the path-space rendering equation in the form used throughout this thesis. It is not a new physical law; it packages measurement, scattering, transport, and emission into one path integral.

2.3.5 Path densities and coordinate changes

A Monte Carlo estimator samples paths according to a density $p(\bar{\mathbf{x}})$. This density is meaningful only relative to the measure in Equation (2.18); there is no measure-independent PDF for a path. If the integral is written in area measure, then $p(\bar{\mathbf{x}})$ must be an area-measure path density. If the algorithm samples directions, its directional PDFs must be converted to area densities before they can be interpreted as a density over $\mathcal{P}_k$.

Consider a transition from $\mathbf{x}_i$ to $\mathbf{x}_{i+1}$ produced by sampling a direction $\boldsymbol{\omega}_i$ from a solid-angle PDF $p_{\omega}(\boldsymbol{\omega}_i | \mathbf{x}_i)$. If that ray hits the surface at $\mathbf{x}_{i+1}$, Equation (2.8) gives the corresponding area density

$$p_A(\mathbf{x}_{i+1} | \mathbf{x}_i) = p_{\omega}(\boldsymbol{\omega}_i | \mathbf{x}_i) \frac{|\mathbf{n}_{\mathbf{x}_{i+1}} \cdot (-\boldsymbol{\omega}_i)|}{\|\mathbf{x}_{i+1} - \mathbf{x}_i\|^2}. \quad (2.19)$$

Conversely, if a point $\mathbf{x}_{i+1}$ is sampled in area measure with density $p_A(\mathbf{x}_{i+1} | \mathbf{x}_i)$, the induced directional density at $\mathbf{x}_i$ is

$$p_{\omega}(\boldsymbol{\omega}_i | \mathbf{x}_i) = p_A(\mathbf{x}_{i+1} | \mathbf{x}_i) \frac{\|\mathbf{x}_{i+1} - \mathbf{x}_i\|^2}{|\mathbf{n}_{\mathbf{x}_{i+1}} \cdot (-\boldsymbol{\omega}_i)|}. \quad (2.20)$$

This is the same Jacobian that appears in emitter sampling. It is a common source of errors in rendering and differentiable-rendering estimators: two proposals cannot be compared using multiple importance sampling until their PDFs are expressed in the same measure.

For a path generated sequentially from the sensor, the path density is a product of conditional densities for each local sampling decision, after all factors have been converted to the target

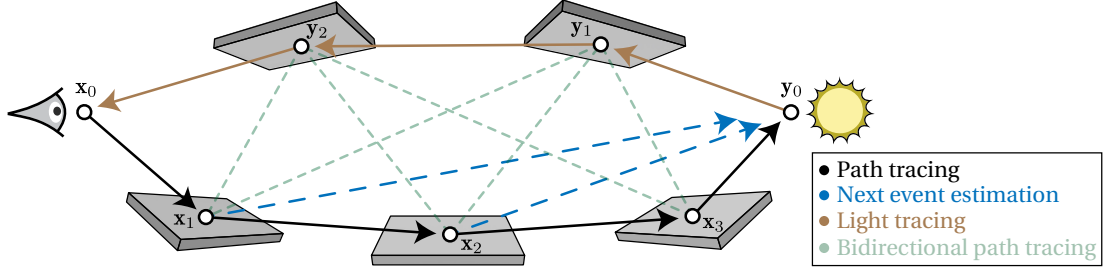


Figure 2.6: **Different ways of constructing light paths.** Unidirectional path tracing grows paths from the sensor, light tracing grows them from emitters, and bidirectional methods connect subpaths from both ends. These are different proposal densities over the same path-space integral, not different physical transport models.

path measure. In a simplified surface-only path tracer,

$$p(\tilde{\mathbf{x}}) = p_{\text{cam},A}(\mathbf{x}_0, \mathbf{x}_1) \prod_{i=1}^{k-1} p_A(\mathbf{x}_{i+1} | \mathbf{x}_i), \quad (2.21)$$

where $p_{\text{cam},A}$ is the area-measure density of the sensor endpoint and first scene intersection, including the camera ray’s direction-to-area conversion. The factors p_A may have originated from directional BSDF sampling through Equation (2.19). Real renderers add discrete probabilities for choosing BSDF components or lights, survival probabilities from Russian roulette, and special handling of delta events. The important point is that every random choice used to construct the path appears in the density. Figure 2.6 shows three common proposal families for constructing such paths.

2.3.6 Multiple parameterizations of the same path

A path is a geometric object, but an estimator represents it using coordinates. The same path may be described by surface positions, by directions and distances, by random numbers pushed through sampling maps, or by a split into sensor and emitter subpaths. Changing coordinates changes the measure and therefore the PDF. In ordinary forward rendering this is mostly careful bookkeeping. In differentiable rendering, it becomes more consequential because the sampling map and its Jacobian may depend on scene parameters.

This observation explains why later chapters often distinguish the physical path contribution from the estimator that samples it. Two primal estimators may have the same expectation while exposing different computational graphs to differentiation. For example, differentiating an estimator that samples a light endpoint can behave differently from differentiating an algebraically equivalent estimator written in solid-angle form, because the sampling map and its Jacobian depend on different scene quantities.

2.3.7 Smooth and delta path sets

The area-measure path space above is easiest to understand for smooth surface scattering and finite area lights. Realistic scenes also contain singular components. An ideal mirror or ideal refractive event does not sample an arbitrary outgoing direction; it imposes a deterministic constraint between incoming and outgoing directions. A point light imposes a singular endpoint constraint. Such paths may lie on lower-dimensional manifolds inside a larger smooth path space. Rendering systems handle them by mixing continuous densities, discrete probabilities, and constraint-aware sampling strategies. Thus Equation (2.18) should be read as the smooth part of a mixed path measure. Delta lights and ideal specular events add discrete or constrained components rather than ordinary area densities.

Reciprocity and sensor–light duality. For nonmagnetic media and reciprocal BSDFs, light transport is reciprocal: the physical weight of a path is unchanged when the path is traversed in the opposite direction, provided that the same endpoint measures are used [11]. In Equation (2.17), this symmetry is visible in the segment factors $G(\mathbf{x}_i, \mathbf{x}_{i+1})$ and in reciprocal scattering kernels. Reversing the path therefore exchanges the sensor response $W_e^{(j)}$ and the emitter radiance L_e without changing the underlying transport contribution. Equivalently, one may start an adjoint transport problem at the sensor with boundary value $W_e^{(j)}$ instead of starting a radiance transport problem at the emitters with boundary value L_e . The propagated adjoint quantity is called *importance*: its value at a scene point and direction measures how strongly radiance inserted there would contribute to pixel j . This interpretation does not change the path integral; it only reverses which endpoint is treated as the source of the transported quantity. Later chapters use the same duality when discussing adjoint radiance and reverse-mode derivative transport.

2.4 Monte Carlo integration for light transport

2.4.1 Basic estimator and error measures

Rendering integrals are difficult for two separate reasons. Even a fixed-length path integral is continuous: the vertices may move over surfaces or through a volume. The full path integral is also variable-dimensional because paths can contain an arbitrary number of scattering events. Grid-based quadrature is therefore impractical for general scenes. Monte Carlo integration estimates such integrals using random samples drawn from the integration measure, avoiding a fixed grid over the full path domain.

Consider a generic integral

$$I = \int_{\Omega} g(u) \, du, \quad (2.22)$$

where Ω is the integration domain, $u \in \Omega$ is a sample, and $g(u)$ is the integrand. If samples u_n are drawn from a probability density $p(u)$ with respect to the same measure du , the standard

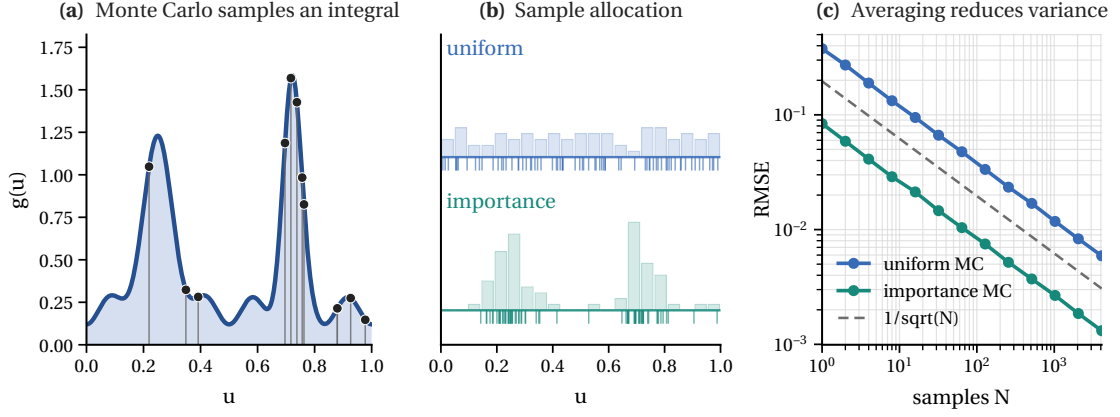


Figure 2.7: **Monte Carlo estimation and variance reduction.** Random samples estimate the shaded integral in a one-dimensional example. The middle panel compares sample allocation: importance sampling places more samples in the high-contribution regions of $g(u)$. The right panel shows that this reduces variance, lowering RMSE while preserving unbiasedness.

Monte Carlo estimator is

$$\hat{I}_N = \frac{1}{N} \sum_{n=1}^N \frac{g(u_n)}{p(u_n)}, \quad u_n \sim p(u). \quad (2.23)$$

The factor $1/p(u_n)$ compensates for the non-uniform sampling density. If $p(u) > 0$ wherever $g(u) \neq 0$, then the estimator is unbiased:

$$\mathbb{E}[\hat{I}_N] = I.$$

For independent samples, its variance is

$$\text{Var}[\hat{I}_N] = \frac{1}{N} \left(\int_{\Omega} \frac{g(u)^2}{p(u)} du - I^2 \right). \quad (2.24)$$

The dimension of Ω does not appear explicitly in this expression; the variance is controlled by how much the weighted integrand g/p fluctuates under the sampling density. Equation (2.24) also shows that the variance of the average decreases as $\mathcal{O}(1/N)$ and the root-mean-square error decreases as $\mathcal{O}(1/\sqrt{N})$. For any estimator $\hat{I}_N$, mean-squared error separates random variation from systematic bias:

$$\mathbb{E}[(\hat{I}_N - I)^2] = \text{Var}[\hat{I}_N] + \text{Bias}[\hat{I}_N]^2, \quad \text{Bias}[\hat{I}_N] = \mathbb{E}[\hat{I}_N] - I. \quad (2.25)$$

Figure 2.7 illustrates the same one-dimensional integral using unbiased Monte Carlo estimators with different variances.

Applying the same idea to Equation (2.18) yields

$$\hat{I}_{j,N} = \frac{1}{N} \sum_{n=1}^N \frac{f_j(\tilde{\mathbf{x}}_n)}{p(\tilde{\mathbf{x}}_n)}, \quad \tilde{\mathbf{x}}_n \sim p(\tilde{\mathbf{x}}), \quad (2.26)$$

Chapter 2. Physically based rendering

where $p(\bar{\mathbf{x}})$ is the path-sampling density induced by the rendering algorithm. The density must be defined with respect to the same path measure μ used in Equation (2.18). This apparently small bookkeeping requirement becomes crucial when differentiating practical rendering code, because changing variables or canceling terms can preserve the primal value while changing the estimator seen by automatic differentiation.

Because Equation (2.26) is the same Monte Carlo construction applied to path space, it inherits the same slow convergence. Many estimators in this thesis are designed to preserve unbiasedness while reducing variance, because both biased values and noisy gradients can destabilize inverse rendering.

2.4.2 Importance sampling

Equation (2.24) shows that Monte Carlo efficiency is controlled by the fluctuation of the sample weight g/p . Importance sampling tries to reduce this fluctuation by choosing p so that high-contribution regions are sampled more often. In rendering, most randomly chosen paths carry little energy to a particular pixel, while a small number of paths may carry large contributions through a light source, caustic, or specular chain. The unattainable ideal for a nonnegative integrand is $p(u) \propto g(u)$, which would make the sample weight $g(u)/p(u)$ constant.

Of course, the full integrand is unknown before rendering. Practical algorithms therefore sample according to partial information. A glossy BSDF suggests directions near its specular lobe. An area light suggests directions toward the emitter. A learned cache or guiding distribution may suggest directions that were useful in previous samples. All of these are proposal distributions for the same underlying transport integral. Figure 2.8 illustrates the same idea on a simple multimodal integrand.

At a surface point, a one-sample estimator of the scattered radiance in Equation (2.3) is

$$\hat{L}_s(\mathbf{x}, \boldsymbol{\omega}_o) = \frac{f_s(\mathbf{x}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o) L_i(\mathbf{x}, \boldsymbol{\omega}_i) |\mathbf{n}_x \cdot \boldsymbol{\omega}_i|}{p(\boldsymbol{\omega}_i)}, \quad \boldsymbol{\omega}_i \sim p(\boldsymbol{\omega}_i), \quad (2.27)$$

where $p(\boldsymbol{\omega}_i)$ is a density with respect to solid angle. The estimator becomes noisy if high-contribution directions have small probability, because the ratio in Equation (2.27) then becomes large.

2.4.3 Emitter sampling and PDF conversion

Emitter sampling is the most common practical instance of the path-density bookkeeping discussed in Section 2.3.5. At a shading point $\mathbf{x}$, the direct-light integral can be parameterized either by an incident direction or by a point on an emitter. Both parameterizations describe the same physical paths, but they use different measures.

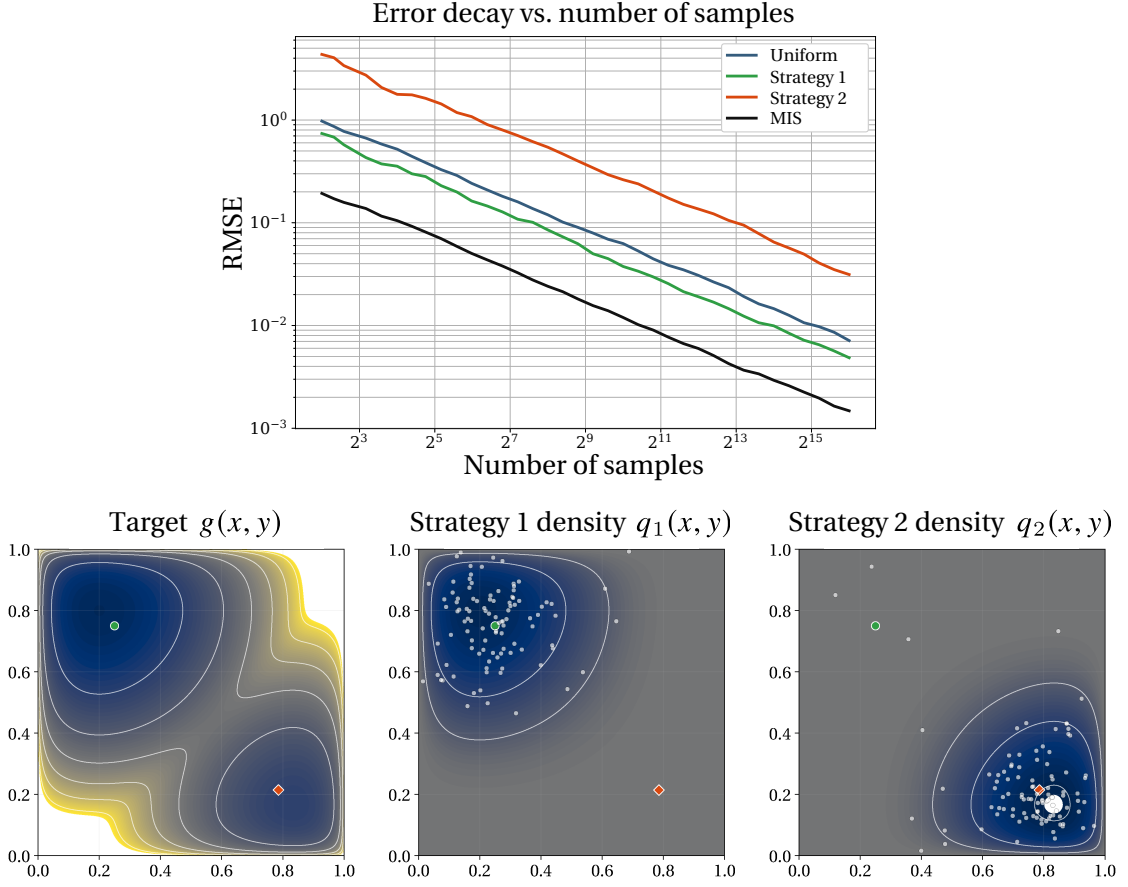


Figure 2.8: **Importance sampling on a multimodal integrand.** The bottom row shows a target integrand $g(x, y)$ together with two proposal densities $q_1(x, y)$ and $q_2(x, y)$, each adapted to a different high-contribution region. The top plot reports the resulting root-mean-square error (RMSE) as the sample count increases. A single proposal misses part of the important mass and converges slowly, while multiple importance sampling combines both proposals and achieves lower error over the full integrand.

Suppose a point $\mathbf{y}$ is sampled on an emitter with area density $p_A(\mathbf{y})$. At $\mathbf{x}$, this sample induces the direction $\boldsymbol{\omega}_i = \boldsymbol{\omega}_{\mathbf{x} \rightarrow \mathbf{y}}$. The corresponding solid-angle density is given by Equation (2.20):

$$p_{\omega}(\boldsymbol{\omega}_i) = p_A(\mathbf{y}) \frac{\|\mathbf{x} - \mathbf{y}\|^2}{|\mathbf{n}_{\mathbf{y}} \cdot (-\boldsymbol{\omega}_i)|}. \quad (2.28)$$

Conversely, the solid-angle density of a BSDF-sampled direction that hits the emitter can be converted to an area density using Equation (2.19). These conversions are necessary because the numerator and denominator of a Monte Carlo estimator must describe the same event with respect to the same measure.

This point is especially important for MIS between BSDF sampling and emitter sampling. If a direct-light path can be produced by either method, then both PDFs must be evaluated with

Chapter 2. Physically based rendering

respect to a common measure before applying a weight such as Equation (2.30). One may compare both PDFs with respect to the solid-angle measure at the shading point, or both with respect to the area measure at the emitter; the result is the same only if the Jacobian is applied consistently.

Light-source selection adds another discrete factor. If the renderer first chooses an emitter ℓ with probability $p(\ell)$ and then samples a point on that emitter with density $p_A(\mathbf{y} | \ell)$, then the full density is

$$p_A(\mathbf{y}) = p(\ell)p_A(\mathbf{y} | \ell). \quad (2.29)$$

Forgetting this discrete factor is equivalent to pretending that the light was sampled more often than it actually was, which biases the estimate. This same discrete–continuous bookkeeping also appears for BSDF lobe selection, Russian roulette, and bidirectional connection strategies.

This point reappears in Chapter 3, where differentiating through implicit changes of variables can otherwise produce incorrect gradients. A primal estimator may remain algebraically correct after canceling or rewriting factors, but the corresponding differentiated estimator can change if the sampling map and its density are not kept explicit.

2.4.4 Multiple importance sampling

No single proposal is reliable in all scenes. For direct lighting, BSDF sampling follows the material lobe but may rarely hit a small emitter. Emitter sampling finds the light but may choose directions where the BSDF is weak. Multiple importance sampling (MIS) combines such proposals so that each can contribute where it is effective [12].

Assume technique i draws n_i samples from density $p_i(u)$. The balance heuristic assigns a sample from technique i the weight

$$w_i(u) = \frac{n_i p_i(u)}{\sum_m n_m p_m(u)}. \quad (2.30)$$

With one sample per technique, this reduces to $p_i(u) / \sum_m p_m(u)$. The exact heuristic is less important here than the principle: BSDF sampling, light sampling, bidirectional connection strategies, and later differential proposals are all different ways of sampling the same integral, and their densities must be combined consistently.

2.5 Path tracing estimators

The previous sections described rendering in terms of an integral and defined the densities that appear in Monte Carlo estimates of that integral. A renderer implements the same estimator incrementally. It constructs one path vertex at a time, updates a product of local factors divided by sampling densities, and adds contributions when the partial path reaches or explicitly samples an emitter. The notation below connects that implementation view back

to the path-space formula above.

2.5.1 Throughput and recursive estimation

A unidirectional path tracer constructs a path from the sensor into the scene. After sampling the camera ray, the algorithm repeatedly intersects the current ray with the scene, evaluates local emission or direct lighting, samples a continuation direction, and updates a multiplicative weight called the *path throughput*.

Let a sampled path have vertices $\mathbf{x}_0, \mathbf{x}_1, \dots$ ordered from the sensor into the scene, and let ω_j be the sampled direction from $\mathbf{x}_j$ to $\mathbf{x}_{j+1}$. Ignoring media for the moment, a typical directional-throughput update after scattering at $\mathbf{x}_j$ is

$$\beta_i = \prod_{j=1}^i \frac{f_s(\mathbf{x}_j, \omega_j, -\omega_{j-1}) |\mathbf{n}_{\mathbf{x}_j} \cdot \omega_j|}{p_j(\omega_j)}, \quad \beta_0 := 1, \quad (2.31)$$

where $p_j(\omega_j)$ is the PDF used to sample the continuation direction at $\mathbf{x}_j$. The numerator is the physical scattering-and-cosine factor; the denominator is the Monte Carlo weight. In a full renderer, β_i also includes transmittance through media, refractive-index factors, MIS weights when appropriate, and color or spectral components.

A simplified emission-accumulation estimator can be written as

$$\hat{I}_j = \sum_{k=1}^n L_e(\mathbf{x}_k, -\omega_{k-1}) \beta_{k-1}. \quad (2.32)$$

This equation assumes that camera sampling and pixel filtering have already been folded into the initial weight. It also omits explicit direct-light sampling for readability. Practical renderers usually decompose the contribution into emitted radiance at visited vertices, next-event estimation for direct lighting, and stochastic continuation for indirect transport. The important structural fact is the product form: local factors multiply along a path, and later derivative estimators exploit exactly this factorization.

2.5.2 Next-event estimation

Relying on random BSDF continuation alone to hit emitters is often inefficient. A path tracer therefore usually performs *next-event estimation* (NEE): at a non-specular vertex, it explicitly samples a light source and tests visibility to it. This samples direct-light paths explicitly rather than relying only on chance. Figure 2.9 shows the practical effect at one sample per pixel: explicit light sampling turns rare direct-light hits into regular direct-light estimates.

At a shading point $\mathbf{x}$, suppose we sample a point $\mathbf{y}$ on an emitter and define

$$\omega_l = \omega_{\mathbf{x} \rightarrow \mathbf{y}} = \frac{\mathbf{y} - \mathbf{x}}{\|\mathbf{y} - \mathbf{x}\|}.$$

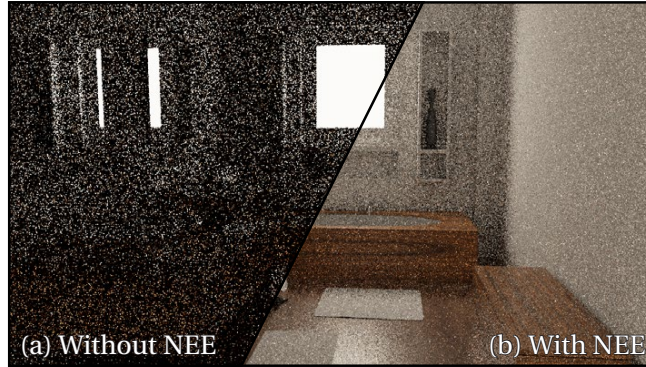


Figure 2.9: **Next-event estimation reduces direct-light variance.** Both images show the same scene rendered at one sample per pixel. **(a)** Without explicit emitter sampling, direct illumination appears mainly when BSDF continuation happens to find a light, producing sparse high-variance samples. **(b)** With NEE, each non-specular vertex samples an emitter directly, giving denser direct-light estimates.

Let $p_\omega(\omega_l)$ be the induced directional PDF from Equation (2.20). A one-sample direct-light estimator in directional measure is

$$\hat{L}_{\text{dir}}(\mathbf{x}, \omega_o) = \frac{f_s(\mathbf{x}, \omega_l, \omega_o) L_e(\mathbf{y}, -\omega_l) V(\mathbf{x}, \mathbf{y}) |\mathbf{n}_x \cdot \omega_l|}{p_\omega(\omega_l)}. \quad (2.33)$$

Equivalently, in area measure the numerator contains the geometry term $G(\mathbf{x}, \mathbf{y})$ and the denominator is $p_A(\mathbf{y})$. NEE is usually combined with BSDF sampling using MIS. For inverse rendering, it is valuable because it reduces image noise and produces denser, less noisy gradient estimates for direct illumination.

2.5.3 Russian roulette

The rendering equation expands into paths of unbounded length. A hard maximum depth would bias the result by discarding all longer paths. Instead, path tracers commonly use *Russian roulette*: after a minimum depth, the path is continued with probability q and terminated with probability $1 - q$. If it survives, its throughput is divided by q . This preserves the expected contribution while avoiding substantial work on paths that are likely to contribute little. If C denotes the continuation contribution, the replacement is unbiased because

$$\mathbb{E} \left[\mathbf{1}_{\text{survive}} \frac{C}{q} \right] = q \frac{C}{q} = C.$$

2.5.4 Path-sampling strategies as path-space proposals

It is useful to describe common rendering algorithms as proposal distributions over the same path-space integral. The physical contribution $f_j(\tilde{\mathbf{x}})$ does not change when the algorithm

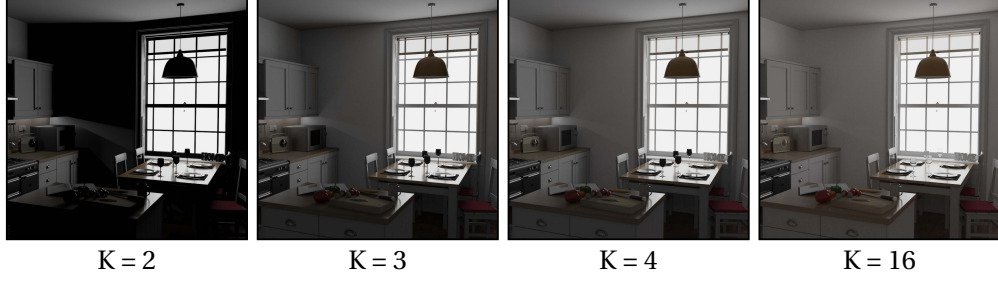


Figure 2.10: **Rendering with increasing path-length cutoff K .** A depth cutoff of K keeps only the first K terms of the path-space expansion. Direct paths capture visible emitters and direct lighting; longer paths add indirect illumination, color bleeding, and other multi-bounce effects.

changes. The algorithm changes the path density $p(\bar{\mathbf{x}})$, the set of paths that can be generated efficiently, and therefore the variance of the Monte Carlo estimator. This viewpoint provides the common language behind path tracing, NEE, bidirectional rendering, manifold methods, and the differentiable-rendering methods studied later in the thesis.

Unidirectional path tracing. A path tracer samples the sensor endpoint and then grows the path forward by repeatedly sampling local scattering directions. In path-space terms, it gives high probability to paths that are easy to generate from the camera according to local BSDFs and phase functions. For a surface path written with respect to area measure, a simplified camera-path density has the form

$$p_{\text{PT}}(\bar{\mathbf{x}}) = p_{\text{cam},A}(\mathbf{x}_0, \mathbf{x}_1) \prod_{i=1}^{k-1} \left[p_{\omega,i}(\omega_i | \mathbf{x}_i) \frac{|\mathbf{n}_{\mathbf{x}_{i+1}} \cdot (-\omega_i)|}{\|\mathbf{x}_{i+1} - \mathbf{x}_i\|^2} \right] \prod_i q_i, \quad (2.34)$$

where $p_{\text{cam},A}$ is the camera-sampling density expressed with respect to the area measure on $(\mathbf{x}_0, \mathbf{x}_1)$, $p_{\omega,i}$ is the local continuation PDF, and q_i denotes discrete probability factors such as lobe-selection probabilities or Russian-roulette survival factors. The bracketed Jacobian converts each sampled direction to an area density for the next vertex. The exact implementation may store the same information in throughput updates rather than in one large product, but the path-space density is the conceptual object being tracked. A finite path-length cutoff K replaces Equation (2.18) by the partial sum

$$I_j^{(\leq K)} = \sum_{k=1}^K \int_{\mathcal{P}_k} f_{j,k}(\bar{\mathbf{x}}) d\mu_k(\bar{\mathbf{x}}). \quad (2.35)$$

This partial sum is useful for understanding and debugging path tracing: increasing K adds successively longer indirect transport, as illustrated in Figure 2.10. Using a hard cutoff as the final estimator is biased unless the omitted tail is negligible or is replaced by an unbiased termination scheme such as Russian roulette.

Next-event estimation as a second proposal. NEE should not be viewed as an extra physical lighting term. It is a second proposal for paths whose final segment reaches an emitter. Instead of waiting for BSDF continuation to hit a light, it samples the light endpoint explicitly and connects it to the current camera subpath. The path contribution is the same as for a path that would have reached the light by BSDF sampling; the PDF is different. For this reason, NEE and BSDF continuation must be combined with MIS when both techniques can produce the same path. This is the simplest example of “one integral, multiple path proposals.”

Light tracing. Light tracing samples paths from emitters and tries to connect them to the sensor or splat them onto it. It can be effective when important transport is easy to generate from the light side but difficult from the sensor side, as with certain caustics. Path-space reciprocity ensures that these light-generated paths estimate the same measurement, provided that the sensor response and density conversions are handled correctly. Light tracing therefore changes the proposal density, not the physical contribution.

Bidirectional path tracing. Bidirectional path tracing samples a camera subpath and a light subpath, then connects pairs of vertices to create complete paths [11]. A connection strategy is often indexed by (s, t) , where s is the number of vertices taken from the light subpath and t is the number of vertices taken from the camera subpath. Different (s, t) strategies can create the same complete geometric path with different probabilities. MIS combines these strategies by evaluating all relevant path densities in a common measure. This is where the path-space view is particularly clarifying: the (s, t) strategies are not separate rendering equations but different parameterizations and proposals for the same path contribution.

The same viewpoint will be used again later for differentiable rendering. Once the target quantity is a derivative rather than a primal image, efficiency still depends on choosing useful coordinates and proposal densities for the relevant integral.

2.6 Surface scattering models

The BSDF is the local scattering model that appears in every surface path contribution. Because later chapters differentiate paths through surfaces, it is useful to understand a BSDF not merely as a material name, but as a measure-valued scattering kernel. Given an incident direction and an outgoing direction, it describes how incident radiance density is redistributed with respect to projected solid angle.

2.6.1 What a BSDF measures

At a surface point $\mathbf{x}$, the BSDF $f_s(\mathbf{x}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o)$ is defined by the differential relation

$$dL_o(\mathbf{x}, \boldsymbol{\omega}_o) = f_s(\mathbf{x}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o) L_i(\mathbf{x}, \boldsymbol{\omega}_i) d\omega_{\mathbf{x}}^\perp. \quad (2.36)$$

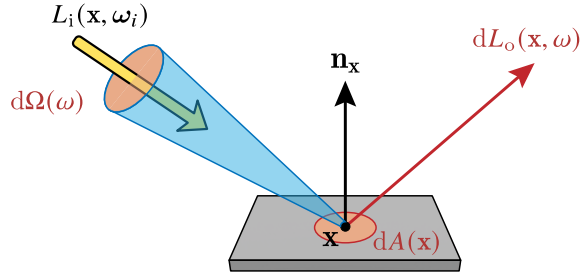


Figure 2.11: **Differential definition of a BSDF.** With the thesis direction convention, both ω_i and ω_o point away from the surface event. Incident radiance arriving along the reverse ray contributes to the outgoing direction in proportion to the BSDF and to the projected incident solid angle.

Equivalently, for a smooth scattering component with nonzero incident radiance,

$$f_s(\mathbf{x}, \omega_i, \omega_o) = \frac{dL_o(\mathbf{x}, \omega_o)}{L_i(\mathbf{x}, \omega_i) d\omega_{\mathbf{x}}^{\perp}}. \quad (2.37)$$

Thus the BSDF is the factor that converts incident radiance from a small angular region around ω_i into outgoing radiance in the chosen direction ω_o . For ordinary smooth scattering its units are inverse steradians. The projected measure is part of the definition, which is why Equation (2.3) integrates $f_s L_i$ against $d\omega_{\mathbf{x}}^{\perp}$ rather than against ordinary solid angle with a separate cosine factor. Figure 2.11 visualizes this local differential definition.

When incident and outgoing directions lie on the same side of the surface, the corresponding component is a BRDF. When light is transmitted to the other side, it is a BTDF. The term BSDF covers both reflection and transmission, and practical materials often mix several components. A glass material, for example, may use a discrete Fresnel choice between reflection and refraction; a plastic material may mix a diffuse base with a glossy coating.

A useful way to interpret the BSDF is as a local conditional transport operator. The incident radiance field $L_i(\mathbf{x}, \cdot)$ is a function on directions. For each outgoing direction ω_o , the scattering equation integrates this function against the kernel $f_s(\mathbf{x}, \cdot, \omega_o)$. In path space, each interior surface vertex contributes exactly one such kernel evaluation. In inverse rendering, differentiating a material parameter changes this local kernel and therefore the weight of every path that visits the surface.

2.6.2 Diffuse, glossy, and ideal specular components

The simplest useful BRDF is Lambertian diffuse reflection:

$$f_r(\mathbf{x}, \omega_i, \omega_o) = \frac{\rho(\mathbf{x})}{\pi}, \quad (2.38)$$

Chapter 2. Physically based rendering

where $\rho(\mathbf{x}) \in [0, 1]$ is the diffuse albedo. The outgoing radiance is independent of ω_o , which makes the Lambertian model a useful baseline for both rendering and inverse problems. The factor $1/\pi$ normalizes the BRDF so that a perfectly white Lambertian surface reflects unit energy over the hemisphere.

Rough glossy materials are commonly modeled using microfacet theory. The model treats the surface as a distribution of tiny planar facets. Only facets whose normals are appropriately oriented reflect or refract light from ω_i to ω_o ; other terms account for Fresnel reflectance and mutual masking and shadowing among facets. For reflection, a standard microfacet BRDF has the form [13]

$$f_r(\mathbf{x}, \omega_i, \omega_o) = \frac{D(\omega_h) F(\omega_i, \omega_h) G_{\text{mf}}(\omega_i, \omega_o)}{4|\mathbf{n}_x \cdot \omega_i||\mathbf{n}_x \cdot \omega_o|}, \quad (2.39)$$

where ω_h is the half-vector, D is the microfacet normal distribution, F is the Fresnel term, and G_{mf} is the masking-shadowing term. We write G_{mf} to avoid confusion with the geometric coupling term $G(\mathbf{x}, \mathbf{y})$ in Equation (2.10). Related microfacet constructions model rough transmission as well.

At the opposite extreme are ideal specular events such as perfect mirrors and ideal dielectric refraction. Their BSDFs are delta-distributed: for a given incident direction, all energy goes into a deterministic outgoing direction determined by reflection or Snell's law. Such components are sampled as discrete events rather than as ordinary densities over solid angle. In path space, an ideal specular chain imposes constraints on the positions of neighboring vertices, which is why specialized methods such as manifold sampling can be useful for caustics and specular paths.

2.6.3 BSDF sampling, mixtures, and physical constraints

A renderer usually samples a BSDF by first choosing a component and then sampling a direction from that component. For a mixture

$$f_s = \sum_m c_m f_m, \quad c_m \geq 0, \quad (2.40)$$

where f_m are diffuse, glossy, transmissive, or delta components, the component choice is a discrete random variable and the sampled direction has a conditional density given that component. The full PDF for the resulting direction is the mixture of the component PDFs, including the discrete probabilities of selecting the components:

$$p_s(\omega_o | \mathbf{x}, \omega_i) = \sum_m q_m(\mathbf{x}, \omega_i) p_m(\omega_o | \mathbf{x}, \omega_i), \quad (2.41)$$

for all smooth components that could have produced ω_o . Delta components contribute through their discrete selection probabilities and deterministic maps rather than through finite solid-angle densities. This is the BSDF-side counterpart of choosing among different emitters or different bidirectional connection strategies.

Physically plausible BSDFs are nonnegative, energy-conserving, and reciprocal under the appropriate transport measure. Energy conservation means that a passive surface cannot scatter more power than it receives, for example

$$\int_{\mathcal{S}^2} f_s(\mathbf{x}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o) d\boldsymbol{\omega}_{o,\mathbf{x}}^\perp \leq 1 \quad (2.42)$$

for each fixed incident direction, up to the usual care needed for transmission and relative-index factors. Reciprocity relates transport along a path to transport along the reversed path and underlies bidirectional and adjoint formulations [11].

These details may seem local, but they propagate into the global estimator. A poor BSDF sampling density increases path-space variance. A missing discrete component probability gives an incorrect path PDF. A delta event changes the dimension of the set being sampled. A material derivative may change both the BSDF value and the distribution used to sample it. These are precisely the kinds of issues that become more delicate when the estimator is differentiated.

2.7 Volume rendering

2.7.1 Medium coefficients, phase functions, and radiative transfer

Participating media allow light to interact along ray segments, not only at surfaces. At a point $\mathbf{x}$, the absorption coefficient $\mu_a(\mathbf{x})$ removes radiance, the scattering coefficient $\mu_s(\mathbf{x})$ redirects radiance, and the extinction coefficient is

$$\mu_t(\mathbf{x}) = \mu_a(\mathbf{x}) + \mu_s(\mathbf{x}). \quad (2.43)$$

All three coefficients have units of inverse length. A useful derived quantity is the single-scattering albedo

$$\omega(\mathbf{x}) = \frac{\mu_s(\mathbf{x})}{\mu_t(\mathbf{x})}, \quad (2.44)$$

which, when $\mu_t(\mathbf{x}) > 0$, is the probability that an extinction event is a scattering event rather than absorption.

Angular scattering at a medium interaction is described by a phase function f_p satisfying

$$\int_{\mathcal{S}^2} f_p(\boldsymbol{\omega}_i, \boldsymbol{\omega}_o) d\boldsymbol{\omega}_o = 1. \quad (2.45)$$

In this thesis, we mainly use isotropic scattering and the Henyey–Greenstein phase function [14]. For the Henyey–Greenstein phase function, the parameter $g \in [-1, 1]$ controls whether scattering is mostly forward, isotropic, or backward. In inverse problems, g often couples strongly with density and illumination; this ambiguity is related to similarity relations in volume scattering [15].

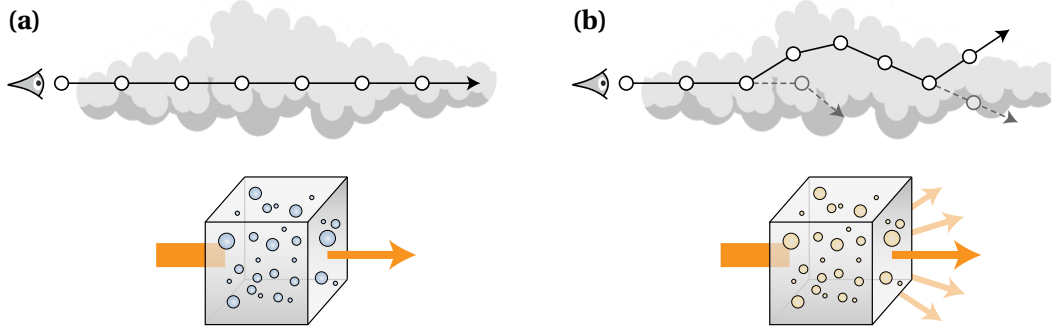


Figure 2.12: **Emission-absorption versus scattering volumes.** (a) In an emission-absorption medium, radiance along a camera ray can be accumulated by transmittance-weighted compositing. (b) When scattering is present, radiance at a point depends on illumination arriving from other directions. The transport problem becomes recursive and generally requires stochastic path tracing.

To avoid confusion between medium scattering and the surface-scattered radiance L_s , define

$$L_{\text{scat}}(\mathbf{x}, \boldsymbol{\omega}) = \int_{\mathcal{S}^2} f_p(\boldsymbol{\omega}', \boldsymbol{\omega}) L(\mathbf{x}, \boldsymbol{\omega}') d\boldsymbol{\omega}'. \quad (2.46)$$

This is the radiance scattered by the medium into the direction $\boldsymbol{\omega}$.

The differential radiative transfer equation (RTE) is usually written for radiance traveling in a fixed physical direction $\boldsymbol{\omega}$:

$$(\boldsymbol{\omega} \cdot \nabla) L(\mathbf{x}, \boldsymbol{\omega}) = -\mu_t(\mathbf{x}) L(\mathbf{x}, \boldsymbol{\omega}) + \mu_s(\mathbf{x}) L_{\text{scat}}(\mathbf{x}, \boldsymbol{\omega}) + Q(\mathbf{x}, \boldsymbol{\omega}), \quad (2.47)$$

where $Q(\mathbf{x}, \boldsymbol{\omega})$ denotes volumetric emission or another source term. Compared with surface transport, the essential change is that radiance is gained and lost continuously along a ray rather than only at isolated vertices. In the integral equations below, however, $\boldsymbol{\omega}$ remains the incident-direction argument of L_i : the point $\mathbf{x} + t\boldsymbol{\omega}$ lies on the side from which light arrives at $\mathbf{x}$, so the physical travel direction of that radiance is $-\boldsymbol{\omega}$. Source and in-scattering terms sampled along this ray are therefore evaluated at $-\boldsymbol{\omega}$.

2.7.2 Transmittance and emission-absorption rendering

Along a segment from $\mathbf{x}$ to $\mathbf{x} + t\boldsymbol{\omega}$, the transmittance is

$$T(\mathbf{x}, \mathbf{x} + t\boldsymbol{\omega}) = \exp\left(-\int_0^t \mu_t(\mathbf{x} + s\boldsymbol{\omega}) ds\right). \quad (2.48)$$

It is the attenuation accumulated along the segment. Equivalently, in a probabilistic interpretation, it is the probability that no real interaction occurs before distance t .

It is useful first to consider an emission-absorption medium, where $\mu_s = 0$. This case is closely

related to NeRF-style radiance fields because radiance along a camera ray can be computed by front-to-back compositing without angular coupling. Let $\mathbf{x}_s$ be the first surface hit, let $s = \|\mathbf{x}_s - \mathbf{x}\|$, and set $\mathbf{x}_t = \mathbf{x} + t\boldsymbol{\omega}$. With the same incident-ray convention as above, the incoming radiance along the ray is

$$L_i(\mathbf{x}, \boldsymbol{\omega}) = \int_0^s T(\mathbf{x}, \mathbf{x} + t\boldsymbol{\omega}) Q(\mathbf{x}_t, -\boldsymbol{\omega}) dt + T(\mathbf{x}, \mathbf{x}_s) L_o(\mathbf{x}_s, -\boldsymbol{\omega}). \quad (2.49)$$

Here $Q(\mathbf{x}_t, -\boldsymbol{\omega})$ denotes the source term added along the ray segment toward the sensor-side query point. If the ray exits the scene instead of hitting a surface, the terminal surface term is replaced by the environment radiance.

For a discretized ray, choose samples $0 = t_0 < t_1 < \dots < t_N = s$, interval lengths $\Delta t_i = t_{i+1} - t_i$, and sample points $\mathbf{x}_i = \mathbf{x} + t_i\boldsymbol{\omega}$. Assuming the coefficients are constant on each interval, Equation (2.49) becomes

$$L_i(\mathbf{x}, \boldsymbol{\omega}) \approx \sum_{i=0}^{N-1} T_i \frac{1 - \exp(-\mu_{t_i} \Delta t_i)}{\mu_{t_i}} Q_i + T_N L_o(\mathbf{x}_s, -\boldsymbol{\omega}), \quad (2.50)$$

where the ratio is interpreted as Δt_i when $\mu_{t_i} = 0$, and

$$T_i = \exp\left(-\sum_{j=0}^{i-1} \mu_{t_j} \Delta t_j\right), \quad \mu_{t_i} = \mu_t(\mathbf{x}_i), \quad Q_i = Q(\mathbf{x}_i, -\boldsymbol{\omega}). \quad (2.51)$$

If we write $Q(\mathbf{x}, -\boldsymbol{\omega}) = \mu_t(\mathbf{x}) c(\mathbf{x}, -\boldsymbol{\omega})$, where c is the color or radiance associated with the sample, then

$$L_i(\mathbf{x}, \boldsymbol{\omega}) \approx \sum_{i=0}^{N-1} T_i \alpha_i c_i + T_N L_o(\mathbf{x}_s, -\boldsymbol{\omega}), \quad \alpha_i = 1 - \exp(-\mu_{t_i} \Delta t_i), \quad c_i = c(\mathbf{x}_i, -\boldsymbol{\omega}). \quad (2.52)$$

This is the standard front-to-back alpha-compositing form used in NeRF-style emission-absorption rendering [2]. It is simpler than full PBR because the color stored at a point is emitted directly toward the query point rather than produced by recursive scattering from other directions. Figure 2.13 shows the corresponding discrete quantities along a ray: higher extinction within an interval produces larger opacity, and accumulated opacity reduces the remaining transmittance. Figure 2.12 contrasts this local ray integration with the recursive scattering case introduced next.

2.7.3 Scattering media and volumetric path tracing

Adding in-scattering and integrating the RTE along the ray yields the volume rendering equation

$$L_i(\mathbf{x}, \boldsymbol{\omega}) = \int_0^s T(\mathbf{x}, \mathbf{x} + t\boldsymbol{\omega}) [Q(\mathbf{x}_t, -\boldsymbol{\omega}) + \mu_s(\mathbf{x}_t) L_{\text{scat}}(\mathbf{x}_t, -\boldsymbol{\omega})] dt + T(\mathbf{x}, \mathbf{x}_s) L_o(\mathbf{x}_s, -\boldsymbol{\omega}), \quad (2.53)$$

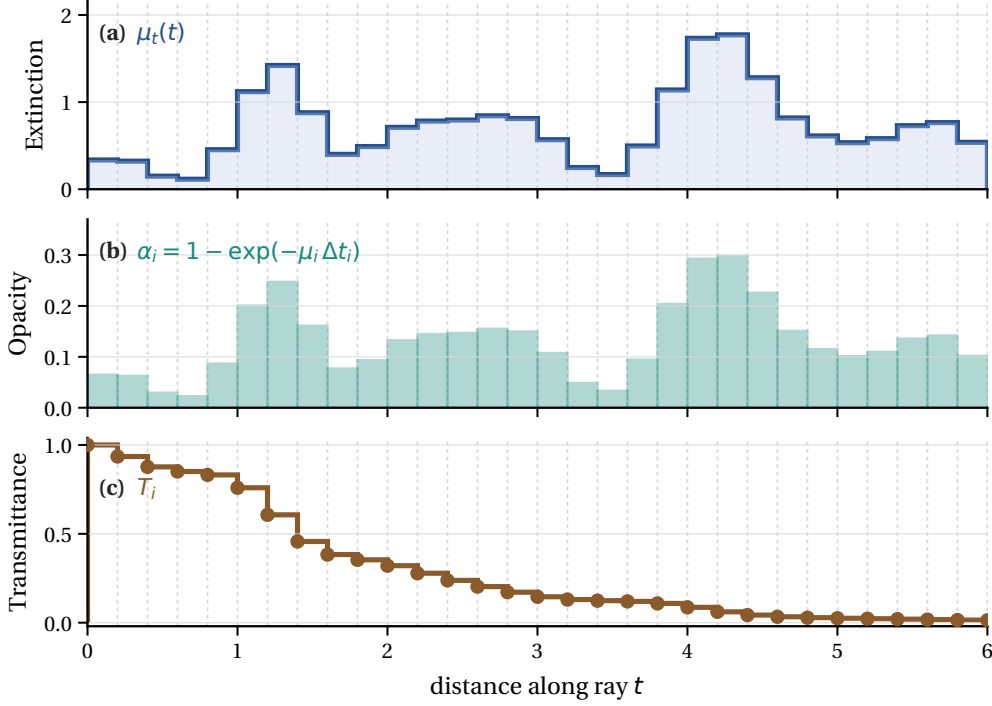


Figure 2.13: **Extinction, interval opacity, and transmittance in emission-absorption rendering.** A ray is divided into intervals with piecewise-constant extinction. Higher extinction produces larger interval opacity, and the accumulated optical depth causes transmittance to decrease along the ray.

where $\mathbf{x}_t = \mathbf{x} + t\boldsymbol{\omega}$. The term $\mu_s(\mathbf{x}_t)L_{\text{scat}}(\mathbf{x}_t, -\boldsymbol{\omega})$ redirects radiance from other directions into the travel direction $-\boldsymbol{\omega}$; this angular coupling is the key difference from emission-absorption rendering.

A volumetric path tracer alternates between surface events and medium events. Along each segment, it samples a free-flight distance, accumulates transmittance, and then either scatters in the medium using a phase function or reaches a surface and applies a BSDF. The same path-space and Monte Carlo ideas apply, with different vertex types and local factors.

2.7.4 Homogeneous and heterogeneous media

For a homogeneous medium, μ_t is constant, and Equation (2.48) reduces to the Beer-Lambert law [10],

$$T(t) = \exp(-\mu_t t). \quad (2.54)$$

The free-flight distance to the next interaction has the closed-form sampling rule

$$t = -\frac{\log(1 - \xi)}{\mu_t}, \quad \xi \sim \mathcal{U}[0, 1), \quad (2.55)$$

where ξ is a uniform random variable. If the sampled distance exceeds the distance to the next surface, the ray reaches the surface first.

For heterogeneous media, $\mu_t(\mathbf{x})$ varies spatially, so both transmittance estimation and free-flight sampling are more difficult. Two families of methods are commonly used. *Ray marching* approximates the line integral by finite steps; it is simple but biased when the discretization is too coarse. *Tracking* or *null-collision* methods sample tentative events from a majorant extinction coefficient $\bar{\mu}_t$ and classify them as real or null events; they can remain unbiased, but their efficiency depends on the quality of the majorant [16]. For inverse rendering, this bias-variance trade-off matters because both biased forward values and high-variance gradients can harm optimization.

2.8 Microflake media

The next two sections focus on models where the distinction between surfaces and volumes becomes less clearly defined. This matters later because several inverse-rendering methods use volume-like intermediate representations while still trying to recover or reason about surfaces.

The volume model described above uses scalar coefficients to describe how a medium absorbs and scatters. Microflake media add directional structure. They model the medium as a volume filled with many tiny oriented flakes whose aggregate behavior is described statistically. This makes them useful for materials such as cloth, hair, wood, and fibrous tissue, where appearance is governed by oriented internal structure rather than by spatial density alone.

Jakob et al. [17] introduced a generalized radiative-transfer framework for this setting. The key difference from ordinary isotropic media and from anisotropic media that depend only on the scattering angle is that the extinction coefficient, scattering coefficient, and phase function can depend on direction. In particular, the phase function depends on the incoming and outgoing directions separately, not only on their relative angle.

A useful intuition is that a microflake medium is a volumetric analogue of a microfacet surface model. A microfacet BRDF imagines a rough surface made of tiny facets at an interface. A microflake medium imagines tiny oriented flakes distributed through a volume. The analogy is helpful, but it is not an identity: microfacets scatter at a boundary, while microflakes also involve free flight and transmittance through the bulk. Figure 2.14 contrasts the corresponding GGX and SGGX normal distributions.

Heitz et al. [18] made this framework practical with the SGGX representation. SGGX describes the flake distribution through projected area, parameterized by an ellipsoid analogous to the GGX microfacet distribution. This representation supports closed-form operators, interpolation, prefiltering, diffuse and specular flakes, and importance sampling based on visible normals [18]. These properties are valuable when microstructural statistics must be stored in

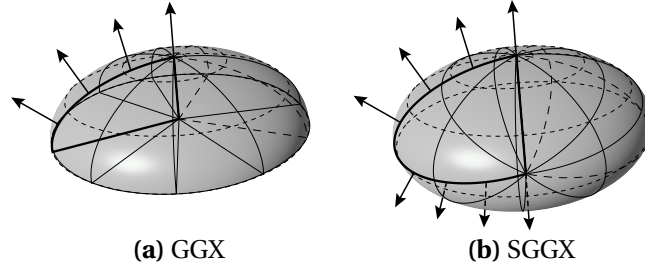


Figure 2.14: **Ellipsoidal normal distributions.** (a) A GGX distribution models normals on an ellipsoidal hemisphere for rough surfaces. (b) An SGGX distribution models flake normals on a complete ellipsoid for oriented volumetric scattering.

voxels or filtered across levels of detail.

Relation to surfaces. A dense microflake volume does not automatically become a microfacet surface. A surface BRDF concentrates all scattering at a boundary, whereas a microflake medium combines oriented scattering with attenuation and multiple interactions throughout a finite volume. This comparison is included because later chapters repeatedly move between surface and volume-like representations; the point is that visual or mathematical similarity does not make the models interchangeable.

Figure 2.15 illustrates this difference. We render a thick homogeneous microflake volume enclosed in a cube and fit a GGX BRDF on the same cube to reproduce its appearance. For the volume, we use the SGGX microflake model with extinction coefficient $\mu_t = 1000$ and parameters $S_{xx} = 0.01$, $S_{yy} = 0.01$, $S_{zz} = 1$, and $S_{xy} = S_{xz} = S_{yz} = 0$. For the surface, we optimize the GGX roughness; the best fit occurs at $\alpha_{\text{GGX}} = 0.2120$ and remains only approximate.

Dupuy et al. [19] study a specific surface-to-volume equivalence: rough surface transport can be reproduced by a semi-infinite homogeneous microflake volume with non-symmetric flakes that are reflective on one side and transparent on the other. Those one-sided flakes are meaningful because the construction still carries an orientation tied to an underlying interface. The example above lies outside those assumptions: the volume is finite, the flakes are symmetric, and the goal is to extract an explicit surface BRDF on the boundary. The connection between rough surfaces and microflake volumes is therefore real but conditional. In general, converting an optimized microflake volume into a surface BRDF is an approximation and may require a separate fitting problem.

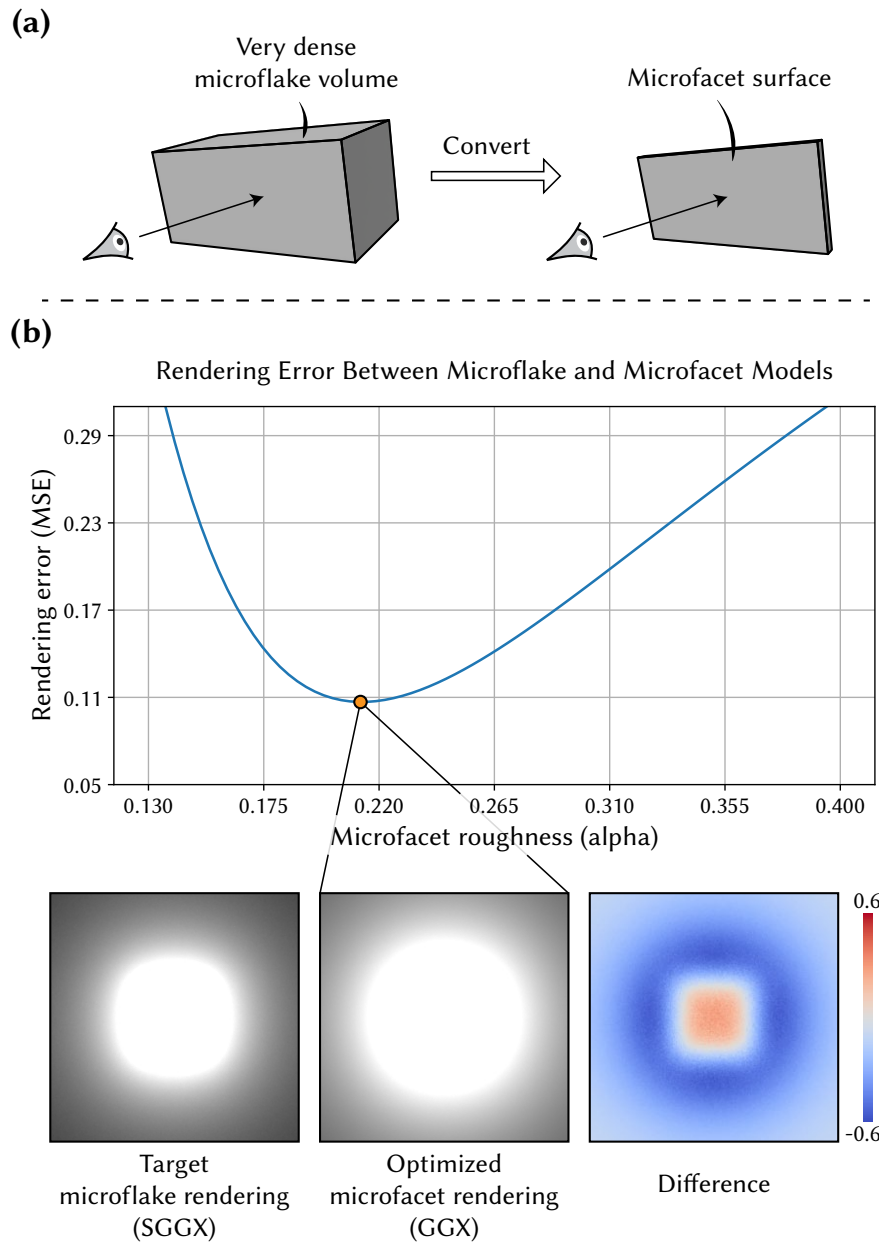


Figure 2.15: **Extracting BRDFs from a simple microflake volume.** (a) A dense homogeneous microflake volume is rendered as the reference. (b) A GGX BRDF on the boundary is optimized to mimic the same appearance. Even in this deliberately simple case, the conversion is an approximation rather than an exact analytic reduction.

2.9 Stochastic surfaces

Stochastic surfaces provide another bridge between surfaces and volumes. Instead of representing geometry as a fixed interface, they model the interface as a random object and define image formation under that random variation. This viewpoint preserves surface semantics while allowing unresolved detail or uncertainty to be represented probabilistically. It is therefore directly connected to the many-worlds formulation developed later in the thesis.

Let S denote one possible surface realization and let $R_j(S)$ be the rendered value of pixel j for that realization. A stochastic-surface image is the expectation

$$I_j = \mathbb{E}_{S \sim p_{\text{surf}}} [R_j(S)], \quad (2.56)$$

where p_{surf} is a probability distribution over surfaces. A deterministic surface is the special case where all probability mass is placed on a single surface. The benefit of the stochastic formulation is that it can soften geometric optimization while still reasoning about surfaces rather than only about densities.

One way to define a distribution over surfaces is to start from a random implicit function

$$\phi \sim \text{GP}(\mu, k), \quad (2.57)$$

where GP denotes a Gaussian process, $\mu(\mathbf{x})$ is the mean function, and $k(\mathbf{x}, \mathbf{y})$ is the covariance kernel. A Gaussian process is a distribution over scalar fields: sampled functions have correlated values at nearby points, with the correlations determined by k [20]. In Gaussian-process implicit surfaces (GPIS), a surface realization is the zero level set

$$S = \{\mathbf{x} \in \mathbb{R}^3 \mid \phi(\mathbf{x}) = 0\}. \quad (2.58)$$

Because $\phi(\mathbf{x})$ is Gaussian at each fixed point $\mathbf{x}$, it also defines an occupancy probability

$$\alpha(\mathbf{x}) = \Pr[\phi(\mathbf{x}) \leq 0]. \quad (2.59)$$

This occupancy is spatially correlated through the underlying random field, unlike a collection of independent voxel opacities [21]. Figure 2.16 illustrates this construction in two dimensions.

Seyb et al. argue that stochastic implicit surfaces provide a transport formulation that connects several regimes usually treated separately [22]. When the random geometry is tightly concentrated around one interface, transport behaves like ordinary surface rendering. When possible interfaces are spread throughout space, behavior becomes more volume-like.

This bridge is not an equivalence in all cases. Miller et al. show that a stochastic solid reduces to standard exponential volumetric transport only under specific assumptions [23]. In general, spatial correlations in the random geometry induce non-exponential free-flight statistics: after a ray has avoided an interface for some distance, the probability of future intersections is not

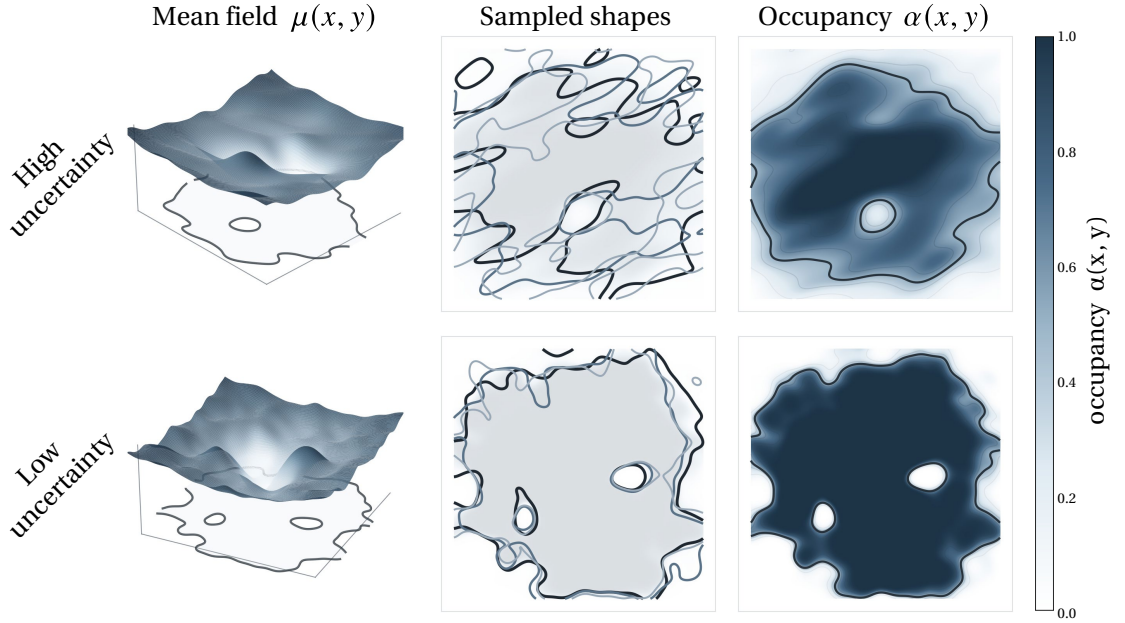


Figure 2.16: **2D illustration of stochastic implicit surfaces at two uncertainty levels.** Each row shows a Gaussian-process implicit surface in two dimensions. **(a)** The left column visualizes the mean field $\mu(x, y)$, whose zero level set defines the mean interface. **(b)** The middle column overlays sampled zero level sets $\phi(x, y) = 0$. **(c)** The right column shows the occupancy probability $\alpha(x, y) = \Pr[\phi(x, y) \leq 0]$. Here, higher uncertainty corresponds to increasing the marginal variance of the random field while keeping the same kernel shape; this spreads the sampled interfaces and makes the occupancy transition more diffuse. In 3D, the same construction produces random surfaces rather than random curves.

the same as in a memoryless participating medium. Thus uncertain geometry cannot always be replaced by a conventional volume without changing the transport model.

Classical GPIS is conceptually useful but can be computationally expensive. Xu et al. show that sparse convolution noise can make GPIS-style stochastic surfaces more practical for rendering systems [24].

Connection to many-worlds derivatives. The many-worlds construction used later should not be interpreted as simply replacing a surface with an ordinary participating medium. A conventional volume changes the primal transport equation: radiance is attenuated and scattered continuously as it travels through density. A many-worlds derivative field instead evaluates many hypothetical surface interactions as alternative local explanations while keeping the forward renderer surface-based. In path-space language, it extends the derivative domain beyond the currently realized surface path space without requiring the primal image formation model to become standard exponential volume transport.

Stochastic surfaces should not be conflated with microflake media either. Both replace unre-

solved geometric detail with statistics, but they randomize different objects. Microflake media randomize oriented scattering elements distributed throughout a volume. Stochastic surfaces randomize the interface itself. In this thesis, stochastic surfaces are the more direct precursor to many-worlds inverse rendering, while microflake media provide a complementary comparison point for surface–volume relations.

2.10 Implications for inverse rendering

Physically based rendering is attractive for inverse problems because its parameters are physically meaningful. Geometry changes cast shadows and alter interreflections. Materials affect angular scattering and energy conservation. Lighting changes direct and indirect illumination. When the goal is relighting, editing, fabrication, or physical interpretation, these semantics are valuable.

The same forward model also makes inverse rendering hard. First, pixel values are high-dimensional Monte Carlo integrals, so both objective values and gradients can be noisy. Second, geometry, material, medium, and illumination parameters are strongly coupled through recursive transport. Third, visibility is discontinuous: a small geometric motion can abruptly reveal or hide a path. These three issues—variance, coupling, and nonsmooth visibility—reappear throughout the thesis.

It is useful to compare PBR with two simpler families of forward models. Rasterization with local shading is fast and usually low-noise, but it omits many global illumination effects. Emissive-volume and radiance-field models are easier to optimize because their gradients are dense along camera rays and their transport recursion is simpler. However, their recovered parameters are often less physically interpretable, and relighting consistency is weaker unless additional constraints are imposed.

Most of this thesis targets the harder setting in which the physical semantics of surfaces, materials, media, and lighting matter. The next chapter therefore keeps the forward model of this chapter fixed and asks how to differentiate it correctly and efficiently. In particular, it studies how to differentiate products of path factors, how to avoid storing full Monte Carlo traces, and how to handle visibility-induced boundary terms.

3 Differentiable physically based rendering

This chapter develops the differentiable rendering background needed for the remainder of the thesis. The central question is how to compute derivatives of the underlying light-transport problem correctly and efficiently. In physically based rendering, that question is subtle because rendering is implemented as a stochastic Monte Carlo estimator, because practical systems rewrite that estimator aggressively for efficiency, and because geometry changes can alter the set of contributing paths through visibility.

The chapter therefore follows three connected themes. First, it revisits automatic differentiation (AD) from a rendering viewpoint and explains why naïve differentiation becomes memory-limited. Second, it develops scalable and correct differentiation in the fixed-visibility regime: path replay backpropagation, local path parameterizations, and the transport viewpoint in which differentiation becomes a light-transport problem in its own right. Third, it turns to visibility changes, where the domain of contributing paths moves and the derivative acquires boundary terms that require specialized estimators. The final sections then step back from derivative correctness alone and explain why inverse rendering remains difficult even with unbiased gradients, thereby motivating the methods developed in the remainder of the thesis.

This chapter is not intended as a comprehensive survey of differentiable rendering. Instead, it is a guided route through the core challenges and ideas needed later in the thesis. Much of the presentation is informed by previous work [25, 26, 27, 28, 29, 30] and by many valuable discussions with colleagues.

3.1 Preliminaries: derivatives of path tracing

This section gives high-level intuition for why “just applying automatic differentiation” to a physically based path tracer can be infeasible or even wrong. The main point is that a Monte Carlo path tracer is *an estimator of an integral*: when we differentiate the code, we are differentiating a particular estimator implementation, which may not correspond to the derivative of the underlying light-transport model.

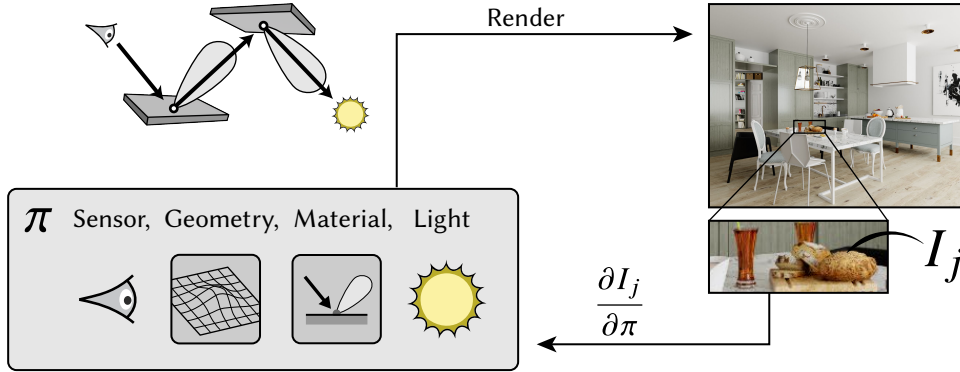


Figure 3.1: **Differentiable rendering as a map from scene parameters to image-space sensitivities.** A renderer maps scene parameters π to an image $\mathbf{I}(\pi)$. The differential query is how a rendered quantity—a pixel value I_j or a loss $\mathcal{L}(\mathbf{I}(\pi))$ —changes when a selected scene parameter π is perturbed. Typical differentiable parameters include material properties, emitter settings, geometry, camera parameters, and medium coefficients.

3.1.1 What does “differentiable rendering” mean here?

Chapter 2 introduced the forward model and emphasized a useful mental model: rendering is an integral over the space of light-transport paths, and Monte Carlo path tracing estimates that integral with random samples. Differentiable rendering keeps the same forward model, but asks for derivatives of rendered pixels with respect to scene parameters. Figure 3.1 summarizes this viewpoint: scene parameters are mapped to an image, and differentiable rendering asks how image-space quantities—or losses built from them—change under perturbations of a chosen parameter.

We collect all differentiable scene parameters into a vector $\pi \in \mathbb{R}^P$ and write $\mathbf{I}(\pi) = \mathcal{R}(\pi)$ for the rendered image. A basic query is a pixel derivative

$$\frac{\partial I_j}{\partial \pi}, \quad (3.1)$$

where π can be a material parameter (roughness, albedo, IOR, texture values), an emitter parameter (intensity, spectrum, pose), a medium parameter (density, scattering coefficients), a camera parameter (pose, intrinsics), or a geometry parameter (vertex positions, deformation controls, rigid transforms). These derivatives are the core ingredient for inverse problems: we define a loss $\mathcal{L}(\mathbf{I}(\pi))$ and optimize π using gradients.

3.1.2 Challenges in differentiating path tracing

To see why this is subtle for path tracing, it helps to return to the path-space view and reason about differentiating a pixel value.

A pixel can be written as an integral over path space (Equation (2.18)), which we estimate

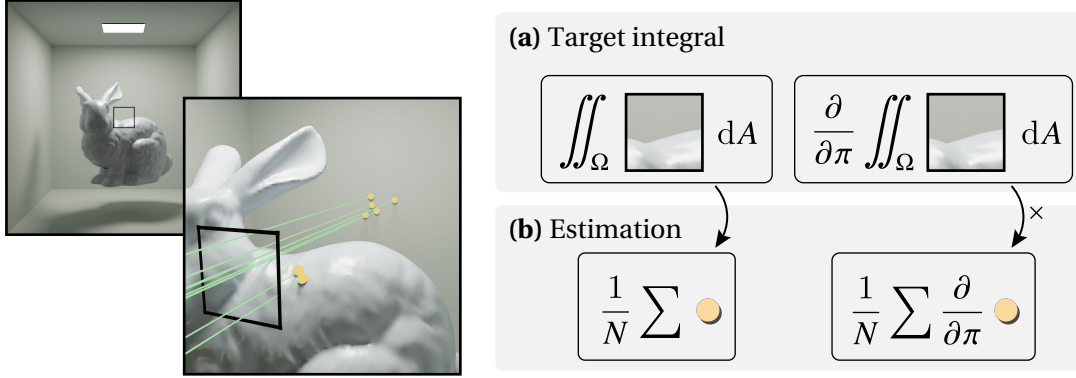


Figure 3.2: **Differentiating the integral vs. differentiating the estimator.** (a) Our mathematical goal is the analytical derivative of the underlying transport integral. (b) However, standard rendering software implements a discrete Monte Carlo estimator. Applying naïve autodiff computes the derivative of this specific programmatic estimator, which can diverge from the true derivative of the integral when estimator rewrites, measure conversions, or discontinuities are differentiated in the wrong form.

using Monte Carlo. Suppressing many details, we will write

$$I_j(\pi) = \int_{\mathcal{P}(\pi)} f_j(\bar{\mathbf{x}}, \pi) d\mu(\bar{\mathbf{x}}), \quad (3.2)$$

where $\bar{\mathbf{x}} = (\mathbf{x}_0, \dots, \mathbf{x}_n)$ is a light path, f_j is its contribution, and $\mathcal{P}(\pi)$ is the set of valid paths.

A primary challenge in inverse rendering stems from a fundamental disconnect between this continuous mathematical formulation and its implementation. In practice, we estimate the physical quantity in Equation (3.2) using a random estimator $\hat{I}_j(\pi)$ such that $\mathbb{E}[\hat{I}_j(\pi)] = I_j(\pi)$. For inverse problems, the quantity we actually want is the derivative of this expectation,

$$\partial_{\pi} I_j(\pi) = \partial_{\pi} \mathbb{E}[\hat{I}_j(\pi)]. \quad (3.3)$$

However, automatic differentiation acts mechanically on the estimator exactly as it is written in code. As illustrated in Figure 3.2, naïvely differentiating the *program* that computes $\hat{I}_j(\pi)$ produces the derivative of one concrete Monte Carlo evaluation, which may take a very different form from Equation (3.2). Derivatives computed this way are generally biased estimates of the true derivative of the underlying integral and can be very noisy.

This mechanical application of AD to the rendering code breaks down in practice for three distinct computational and mathematical reasons:

(1) Reverse-mode AD can be impractical. A path tracer is a long, branchy stochastic computation: each sample is a random walk with data-dependent length (Russian roulette, participating media, ...). Reverse-mode AD generally requires access to intermediate primal values

Chapter 3. Differentiable PBR

during the backward pass, which in turn requires recording a trace (tape), using checkpointing, or applying an equivalent program transformation. For a single path, this is possible, but the overhead is large relative to the small amount of arithmetic performed by one random walk. In practical rendering, we trace millions of paths, and the stored per-bounce state grows roughly in proportion to the total number of simulated scattering events. This storage can become prohibitive even when primal rendering itself is perfectly feasible.

Later in this chapter, we introduce methods that avoid an unbounded tape by reorganizing the computation around light-transport structure [28, 29].

(2) Differentiating primal estimators can be wrong and inefficient. It is common to write rendering code that is fully correct for primal rendering, yet becomes wrong (or incomplete) under differentiation. Practical renderers routinely rewrite estimators for efficiency and variance reduction: they cancel terms, switch measures (area vs. solid angle), or return pre-divided quantities for numerical stability. These transformations preserve the primal estimator, but they do not necessarily guarantee that the same code, when differentiated, produces an unbiased estimator for the derivative of the underlying integral.

Implicit change of variables. For two surface points $\mathbf{x}$ and $\mathbf{y}$ connected by direction $\boldsymbol{\omega} = \frac{\mathbf{y}-\mathbf{x}}{\|\mathbf{y}-\mathbf{x}\|}$, the projected measure at $\mathbf{x}$ satisfies (Equation (2.9))

$$d\omega_{\mathbf{x}}^{\perp} = \frac{|\mathbf{n}_{\mathbf{x}} \cdot \boldsymbol{\omega}| |\mathbf{n}_{\mathbf{y}} \cdot (-\boldsymbol{\omega})|}{\|\mathbf{x} - \mathbf{y}\|^2} dA(\mathbf{y}).$$

The same geometric connection $\mathbf{x} \rightarrow \mathbf{y}$ can be produced either by sampling a direction $\boldsymbol{\omega}$ (solid-angle domain) or by sampling a point $\mathbf{y}$ on an emitter (area domain, e.g. next-event estimation). The change of variables itself is not a problem; if the domain is smooth and fixed, either coordinate system can be differentiated correctly. The difficulty appears when the coordinate map has moving discontinuities or moving domain boundaries, or when an implementation hides the Jacobian through cancellations that make the differentiated program differ from the intended derivative estimator.

Pre-divided weights and “zero gradients.” A more extreme case appears in standard Monte Carlo renderer interfaces for numerical and implementation reasons. BSDF sampling routines commonly return a pre-weighted quantity that already includes division by the sampling probability [31]. With perfect importance sampling, this returned weight is often simplified to a literal constant (e.g. “1”) in code. Primal rendering remains correct, but differentiating the returned constant yields a zero derivative.

Later in this chapter, we will take a more principled approach: instead of differentiating an estimator designed for primal rendering, we will treat derivative computation as a stand-alone problem. We will also show that many variance-reduction techniques designed for primal rendering, once differentiated, can even be detrimental for derivative estimation.

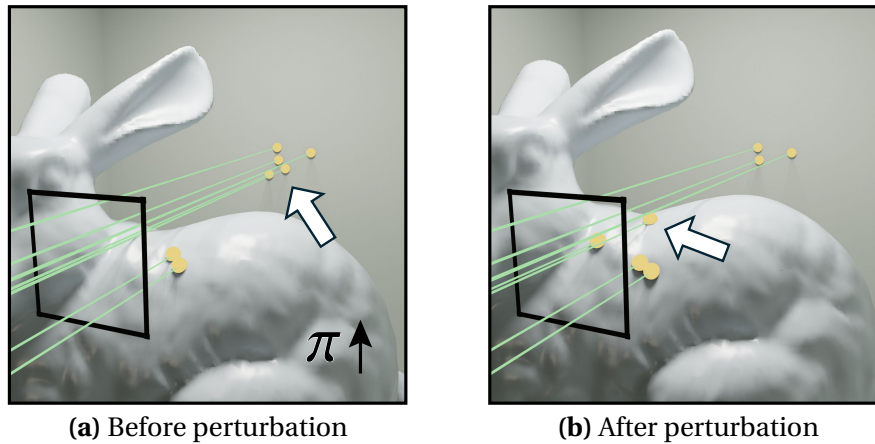


Figure 3.3: **Derivatives due to visibility changes.** A small upward perturbation π applied to the bunny causes rays near the silhouette that previously missed the geometry to become occluded. This sudden change in ray color is a critical source of derivatives, but it cannot be captured by locally differentiating an unperturbed light path that does not intersect the bunny.

(3) Discontinuous derivatives on a challenging domain. Analytic differentiation reveals a new challenge not present in primal rendering: part of the target derivative exists on a domain that differs from the standard path space. Although these derivatives may be small, they are essential for correctness and their absence can break even simple optimizations.

These derivatives arise because we are differentiating an integral with discontinuities. The missing term captures the contribution of the boundaries as they shift under differentiation; in the literature, such terms are also called “boundary derivatives”.

The most important source of discontinuity in physically based rendering is the visibility term. In Chapter 2, visibility entered explicitly through $V(\mathbf{x}, \mathbf{y}) \in \{0, 1\}$ in the geometry term $G(\mathbf{x}, \mathbf{y})$ (Equation (2.10)). When geometry moves, V can flip discontinuously: an infinitesimal perturbation can change which surface a ray intersects, or whether a connection is occluded. In the path-space picture (Equation (2.18)), this means the effective domain of contributing paths $\mathcal{P}(\pi)$ changes with π .

Such derivatives simply cannot be captured by differentiating light paths in a mechanical way. As illustrated in Figure 3.3, a tiny upward parameter perturbation π causes rays that initially miss the bunny to suddenly become occluded, leading to an abrupt change in color. However, from the perspective of an automatic differentiation system, the original unperturbed light paths never touch the bunny; the system has no local information that the geometry even exists. Consequently, a mechanical application of AD to these paths evaluates to a zero gradient with respect to the object’s position, missing the discontinuity entirely.

Later in this chapter, we will analyze the effect of these visibility-induced discontinuities on the derivative, and show how they can be handled with tailored estimators.

3.1.3 A common misconception: differentiable ray tracing

A common misconception is that the main challenge in differentiable rendering is to differentiate the ray-tracing operator, especially since production renderers often rely on highly optimized third-party kernels (e.g., Embree [32], OptiX [33], or custom BVHs).

In practice, differentiating ray-surface intersections is largely a solved local problem; the difficult parts are the estimator-level issues above (scalability and correctness with moving geometry).

Ray-triangle intersection. For triangle meshes, a typical solution is: (i) call a fast non-differentiable routine to determine the closest hit primitive, then (ii) locally re-compute the ray-triangle intersection against that one triangle with AD enabled, using a closed-form algorithm such as the Möller–Trumbore algorithm [34]. This gives correct local derivatives under a key assumption: the identity of the closest hit does not change under the infinitesimal perturbation we are differentiating through. When that assumption is violated (the hit switches), we are exactly in the visibility-discontinuity case above; the missing term is not a triangle-intersection detail, but a boundary term.

Iterative intersection solvers. For spline surfaces, subdivision surfaces, and implicit geometry, intersections are often obtained by iterative root finding. We do not need to differentiate every solver iterate. Instead, we differentiate the defining constraint of the final hit point—equivalently, the root condition that characterizes the intersection—using implicit differentiation [35]. This yields derivatives of the solution (hit point, surface parameters, normals) with respect to ray and scene parameters; in practice, it is often implemented as a separate reattachment step after a non-differentiable intersection query.

Summary. The rest of this chapter addresses the above issues: (i) scalable reverse-mode differentiation for long Monte Carlo paths, (ii) estimator design choices for correct and efficient derivatives of the transport integral, and (iii) visibility-induced discontinuities.

3.2 Automatic differentiation in rendering

This section reviews automatic differentiation (AD) and specializes the discussion to rendering. It establishes two mental models that will be used throughout this chapter: (i) AD queries Jacobians through *Jacobian products* rather than forming them explicitly, and (ii) reverse-mode AD conceptually requires a *trace* of intermediate primal values, which interacts poorly with long Monte Carlo paths.

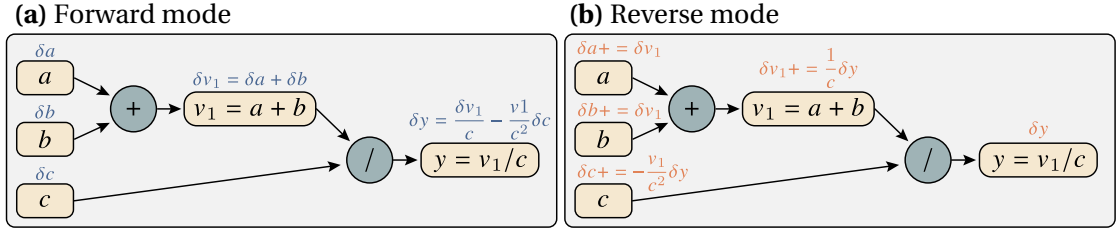


Figure 3.4: **Forward and reverse modes on a computation graph.** The program computes $v_1 = a + b$ and $y = v_1 / c$. Forward mode propagates directional derivatives from the inputs to the output, while reverse mode propagates an output adjoint back to the inputs. Both evaluate the same chain rule on the same graph, but in opposite directions.

3.2.1 Automatic differentiation

Automatic differentiation is a set of techniques for evaluating derivatives of functions. It is best understood as a systematic application of the chain rule to the sequence of elementary operations executed by the program [35]. Unlike symbolic differentiation, AD does not attempt to construct a closed-form expression. Unlike finite differences, AD does not rely on numerical perturbations and therefore avoids truncation error. Instead, it reuses the program structure and computes derivatives alongside (or after) the primal computation, with accuracy limited only by floating-point rounding.

Setup and notation. Consider a function $\mathbf{y} = f(\mathbf{x})$ implemented by a program, where $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{y} \in \mathbb{R}^m$. Its first-order behavior is described by the Jacobian $\mathbf{J}_f(\mathbf{x}) = \partial \mathbf{y} / \partial \mathbf{x} \in \mathbb{R}^{m \times n}$. In practice, we rarely form $\mathbf{J}_f$ explicitly. Instead, AD systems provide efficient ways to apply $\mathbf{J}_f$ (or its transpose) to vectors:

- **Forward mode** computes a Jacobian–vector product (JVP), $\delta \mathbf{y} = \mathbf{J}_f(\mathbf{x}) \delta \mathbf{x}$.
- **Reverse mode** computes a vector–Jacobian product (VJP), $\delta \mathbf{x} = \mathbf{J}_f(\mathbf{x})^\top \delta \mathbf{y}$.

Both modes query the same linear map differently; the choice depends on cost and the derivative information needed, as discussed below.

A concrete example. Figure 3.4 shows a tiny program with addition and division:

$$v_1 = a + b, \quad y = \frac{v_1}{c}. \quad (3.4)$$

The same computation can be differentiated in two equivalent ways.

In forward mode, we choose an input perturbation direction $(\delta a, \delta b, \delta c)$ and propagate *direc-*

tional derivatives along the graph:

$$\delta v_1 = \delta a + \delta b, \quad \delta y = \frac{\delta v_1}{c} - \frac{v_1}{c^2} \delta c. \quad (3.5)$$

In reverse mode, we seed an output sensitivity δy and propagate *adjoints* backward:

$$\delta v_1 += \frac{1}{c} \delta y, \quad \delta c += -\frac{v_1}{c^2} \delta y, \quad \delta a += \delta v_1, \quad \delta b += \delta v_1. \quad (3.6)$$

In both cases we are applying the chain rule on the same computation graph; the difference is whether we propagate information forward from inputs (JVP) or backward from outputs (VJP). Both expressions depend on the primal intermediates computed earlier: for example, the reverse update for c uses the stored value $v_1 = a + b$ from Equation (3.4).

Cost and memory trade-offs. Forward and reverse modes compute the same Jacobian, but expose it through different products. Forward mode answers: “how does the output change along a chosen input direction?” Its cost scales with the number of input directions of interest, since each forward sweep produces one JVP. Reverse mode answers the dual question: “given an output sensitivity, how should it be distributed back to the inputs?” Its cost scales with the number of output directions of interest, since each reverse sweep produces one VJP.

This is why reverse mode is the default when the objective is scalar ($m = 1$): one reverse sweep yields gradients with respect to all n inputs. The main practical downside is memory use. To propagate adjoints backward through an operation, one generally needs the primal values used during the forward evaluation, as in the toy example above. Hence reverse mode is commonly implemented as a primal pass that records a trace (tape) followed by an adjoint pass that traverses this trace in reverse [35]. With sufficient storage (or checkpointing), both modes can be implemented for the same program; the choice is primarily about the input/output dimensions and the available memory budget.

3.2.2 Forward and backward modes for renderers

We now specialize the above discussion to rendering. We treat the renderer as a function mapping scene parameters to an image:

$$\mathbf{I}(\pi) = \mathcal{R}(\pi), \quad (3.7)$$

where $\pi \in \mathbb{R}^p$ collects all differentiable scene parameters (geometry, materials, emitters, medium coefficients, camera parameters, ...), and $\mathbf{I} \in \mathbb{R}^n$ stacks the n pixel values of the rendered image (implicitly including color channels if needed). The Jacobian $\partial \mathbf{I} / \partial \pi \in \mathbb{R}^{n \times p}$ is far too large to form explicitly, so we again work with Jacobian products.

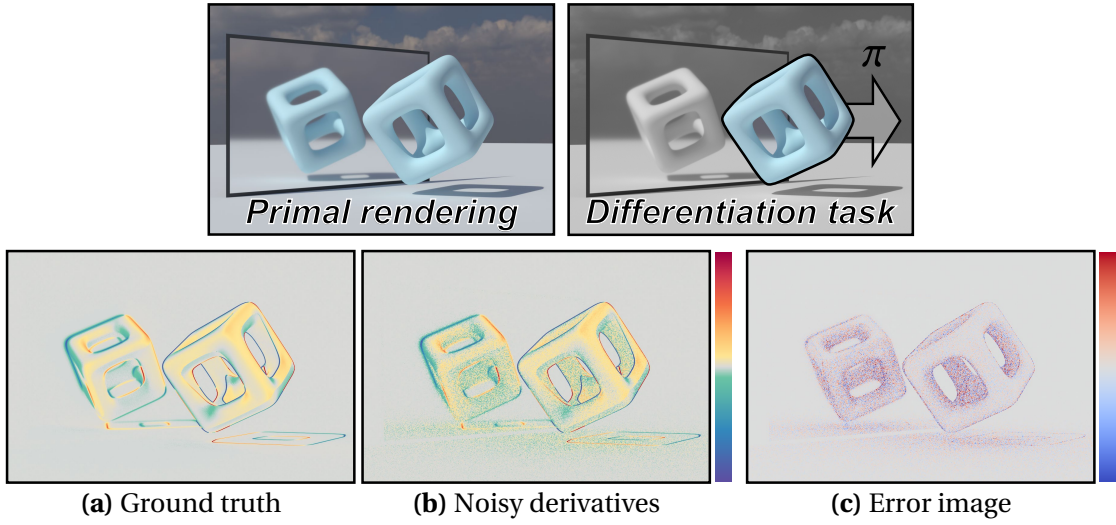


Figure 3.5: **Reading forward derivative images.** The top row shows the primal rendering and the parameter perturbation being differentiated. **(a)** A reference derivative image for that perturbation. Warm colors indicate pixels that get brighter under the perturbation, and cool colors indicate pixels that get darker. **(b)** A noisy Monte Carlo estimate of the same derivative. **(c)** Signed error with respect to the reference. Throughout this chapter, such images are used to judge both correctness and estimator efficiency: structured error indicates bias, while fine-scale noise indicates variance.

Forward mode: perturb parameters, observe a gradient image. Forward mode corresponds to the Jacobian–vector product

$$\delta \mathbf{I} = \frac{\partial \mathcal{R}(\pi)}{\partial \pi} \delta \pi. \quad (3.8)$$

Here, $\delta \pi$ is a user-chosen perturbation direction in parameter space. A common special case is choosing $\delta \pi$ as a standard basis vector, which isolates $\partial \mathbf{I} / \partial \pi_i$ for a single parameter π_i . The output $\delta \mathbf{I}$ lives in image space and can be visualized directly. This is particularly useful for debugging: gradient images are often the most interpretable way to sanity-check correctness and noise behavior before attempting any optimization.

Figure 3.5 shows how such forward derivatives are interpreted throughout this chapter.

Backward mode: start from a loss, obtain parameter gradients. In inverse problems, we typically define a scalar objective $\mathcal{L}(\mathbf{I}(\pi))$ and want $\nabla_{\pi} \mathcal{L}$. Backward mode applies the chain rule in the form

$$\nabla_{\pi} \mathcal{L} = \left(\frac{\partial \mathcal{R}(\pi)}{\partial \pi} \right)^{\top} \nabla_{\mathbf{I}} \mathcal{L}. \quad (3.9)$$

We first form an image-space adjoint signal $\delta \mathbf{I} := \nabla_{\mathbf{I}} \mathcal{L}$, and then propagate it through the renderer via a VJP to obtain gradients with respect to all scene parameters. This is the default setting for inverse rendering because π is often very high-dimensional. The corresponding diagnostic output is a parameter-space gradient rather than a derivative image: for example,

reverse mode may produce a texture-sized map indicating which texels should change to reduce the loss.

Implications for Monte Carlo path tracing. Because Equations (3.8) and (3.9) are generic Jacobian products, one could, in principle, “apply AD” to a renderer. In practice, physically based path tracing is a long, stochastic, branch-heavy computation. A naïve reverse-mode implementation would have to record the full per-path state needed for the backward pass (intersection records, BSDF values, PDFs, medium state, random choices, ...). The resulting trace grows with the total number of simulated scattering events, which is exactly the scaling issue discussed above. The next sections therefore keep the JVP/VJP viewpoint from AD but redesign the estimator to make reverse-mode differentiation scalable for Monte Carlo light transport [28, 29]; later sections address correctness under visibility-induced discontinuities.

3.3 Path replay backpropagation

The previous sections framed differentiation of a renderer as a Jacobian product and motivated reverse mode as the appropriate computational mode when the number of parameters is large. It is therefore tempting to apply reverse-mode AD to a path tracer and obtain gradients. In practice, a direct taped implementation can become memory-limited: reverse mode needs intermediate primal values during the backward pass, which usually means recording a long trace or relying on checkpointing. For Monte Carlo path tracing, the trace grows with the total number of scattering events across all simulated paths and can require storage comparable to a substantial amount of bidirectional path-tracing state.

Radiative backpropagation (RB) and path replay backpropagation (PRB) [28, 29] address this problem by reorganizing how derivatives are computed. For the detached formulation discussed here, PRB computes the same derivative as a naïve reverse-mode implementation while requiring only constant memory and linear time. This is achieved by leveraging the special structure of light transport to avoid an unbounded trace. Other attached variants differentiate more of the sampling procedure, but the replay idea described below is the part needed for the rest of this chapter.

To expose this arithmetic structure cleanly, we begin with a simplified setting: geometry and the camera are fixed, so ray intersections do not depend on the parameters we differentiate. Later sections add continuous geometry terms and visibility terms; this section focuses narrowly on the core algebraic perspective on PRB.

3.3.1 Path contributions and local derivatives

We consider a unidirectional path tracer without MIS. For exposition, it is convenient to start from a BSDF random walk that contributes when it hits an emitter. More practical estimators

introduce next-event estimation, MIS, participating media, and delta events, but the same logic remains. A single camera sample generates a path $\bar{\mathbf{x}} = (\mathbf{x}_0, \dots, \mathbf{x}_n)$, where $\mathbf{x}_0$ is the sensor vertex, $\mathbf{x}_1, \dots, \mathbf{x}_{n-1}$ are the non-emissive scattering vertices produced by the random walk, and $\mathbf{x}_n$ is the terminal emissive vertex. The resulting contribution is

$$L = \beta_{n-1} L_e(\mathbf{x}_n, -\boldsymbol{\omega}_{n-1}), \quad (3.10)$$

where the throughput is accumulated only over the scattering vertices encountered before the final emission evaluation. A typical surface-only form is

$$\beta_i = \prod_{j=1}^i t_j, \quad t_j := \frac{f_s(\mathbf{x}_j, \boldsymbol{\omega}_j, -\boldsymbol{\omega}_{j-1}) |\mathbf{n}_{\mathbf{x}_j} \cdot \boldsymbol{\omega}_j|}{p_j(\boldsymbol{\omega}_j)}, \quad i = 1, \dots, n-1, \quad (3.11)$$

with $\beta_0 := 1$.

In this setting, differentiating L amounts to differentiating the smooth local computations that appear in the factors t_j and in the terminal emission L_e . Computing those local derivatives is exactly what an AD system is well suited for: differentiating BSDF evaluation, emitter evaluation, and other smooth shader terms inside the loop body (Section 3.2).

Reverse mode does *not* need to retain the entire downstream computation at every vertex. It needs only the influence of the *rest of the sampled path* on the current local derivative. For a path contribution, that future dependence can be summarized by the suffix

$$S_i := \left(\prod_{j=i+1}^{n-1} t_j \right) L_e(\mathbf{x}_n, -\boldsymbol{\omega}_{n-1}), \quad i = 0, \dots, n-1, \quad (3.12)$$

so that

$$L = \beta_i S_i \quad (3.13)$$

for any $i \in \{0, \dots, n-1\}$. In words, S_i is simply the contribution of the path segment that starts *after* vertex $\mathbf{x}_i$.

Differentiating L then gives

$$\partial_\pi L = \sum_{i=1}^{n-1} \underbrace{(\partial_\pi t_i)}_{\text{local}} \underbrace{(\beta_{i-1} S_i)}_{\text{suffix weight}} + \underbrace{(\beta_{n-1} \partial_\pi L_e)}_{\text{local}}, \quad (3.14)$$

where $\partial_\pi t_i$ depends only on the local computation at vertex $\mathbf{x}_i$, while $\beta_{i-1} S_i$ provides the downstream weight that tells us how strongly that local perturbation affects the final path contribution.

Equation (3.14) has a useful property for reverse-mode AD: it expresses the derivative as a sum of local contributions, each multiplied by a suffix weight from the remaining path. In a generic program (e.g., a neural network), such future dependence is usually represented by a

high-dimensional adjoint state. For this scalar radiance estimator, however, it collapses to the suffix contribution S_i . With vector-valued spectra, polarization, or constrained delta transport, the suffix state can carry more structure, but the key point remains that it is far smaller than the full recorded path state.

This suffix weight can, in turn, be recovered from quantities already present in the sampled path. Since

$$L = \beta_{i-1} t_i S_i, \quad (3.15)$$

we have

$$\beta_{i-1} S_i = \frac{L}{t_i}, \quad i = 1, \dots, n-1. \quad (3.16)$$

This identity is the arithmetic core of PRB. A naïve reverse-mode implementation would store enough intermediate state so that the correct downstream weight is available when backpropagating through bounce i . PRB instead observes that, for the path that was actually sampled, this weight is already encoded in the final scalar contribution L together with the local factor t_i .

The only challenge is that L becomes known only after the full path has terminated, so the algorithm must first finish the primal path and then replay the same random walk to expose these suffix weights at the vertices where they are needed.

3.3.2 Replay-based reverse-mode implementation

Although Monte Carlo estimators are stochastic in theory, typical rendering implementations are deterministic once the pseudorandom number stream is fixed. If we reset the pseudorandom number generator (PRNG) to the same state, we reproduce the same sequence of random variates and retrace the same random walk. In practice, this can be done by storing a per-sample seed or by using a counter-based generator indexed by the sample ID.

PRB uses this determinism to build a reverse-mode algorithm whose memory use is constant with respect to path depth. The first pass traces a path and stores only a small amount of information, most importantly, the seed and the final scalar contribution L . The second pass resets the PRNG, retraces the identical path, and accumulates gradient contributions on the fly. Conceptually, the second pass is not a second estimator for a different quantity: it reconstructs, at each bounce, the same suffix weight that a taped reverse-mode implementation would have read from the recorded trace.

Listings 3.1 and 3.2 show the primal and replay/adjoint passes separately. We use a local helper `backward_grad(expr, d_expr)`, which backpropagates an adjoint signal `d_expr` through the local computation `expr` and accumulates gradients into the scene parameter vector. The primal pass stores only lightweight per-sample information, most importantly, the seed and the final scalar contribution L . The replay pass resets the PRNG to that seed, retraces the

Algorithm 3.1: **Forward pass for PRB.** The primal pass computes the scalar path contribution L while storing only lightweight per-sample replay information.

```
def sample_path(ray, rng):
    L = 0.0; beta = 1.0
    for i in range(N):
        si = intersect(ray)
        Le = eval_emitter(si, -ray.d)
        L += beta * Le

        wo, bsdf_val, bsdf_pdf = sample_bsdf(si, -ray.d, rng)
        bsdf_weight = bsdf_val / bsdf_pdf
        beta *= bsdf_weight
        ray = spawn_ray(si, wo)
    return L
```

same random walk, and accumulates gradient contributions on the fly. For readability, the pseudocode uses the common form that adds emitted radiance whenever an emissive surface is hit; in the simplified terminal-emitter setting above, all non-terminal emitter evaluations are simply zero.

The key replay step is the subtraction $L -= \text{beta} * \text{eval_emitter}(\dots)$. After the local emitter derivative has been accumulated with weight beta , this subtraction removes the current emission term from the running value L . At that point, L no longer stores the total contribution of the path, but only the contribution of the suffix that remains after the current vertex. The subsequent call to $\text{backward_grad}(\text{bsdf_weight}, dL * (L / \text{bsdf_weight}))$ then injects exactly the reverse-mode weight from Equation (3.16).

Why backpropagation can use constant memory. PRB avoids an unbounded path-length tape because the cross-bounce coupling needed by reverse mode is carried by a low-dimensional throughput-like quantity rather than by the full path state. Local shader evaluations may still use small tapes, computation graphs, or derivative code produced by program transformation; what PRB removes is the need to store the whole stochastic path-tracing loop. Memory therefore stays essentially independent of path length.

It is helpful to contrast this with a generic reverse-mode computation. Consider an iterative program ending in a scalar objective $\ell(\mathbf{s}_N)$:

$$\mathbf{s}_{i+1} = h(\mathbf{s}_i; \pi), \quad i = 0, \dots, N-1, \quad (3.17)$$

Reverse mode propagates an adjoint vector

$$\mathbf{a}_i = (\partial_{\mathbf{s}_i} h(\mathbf{s}_i; \pi))^T \mathbf{a}_{i+1}, \quad \mathbf{a}_N = \nabla_{\mathbf{s}_N} \ell, \quad (3.18)$$

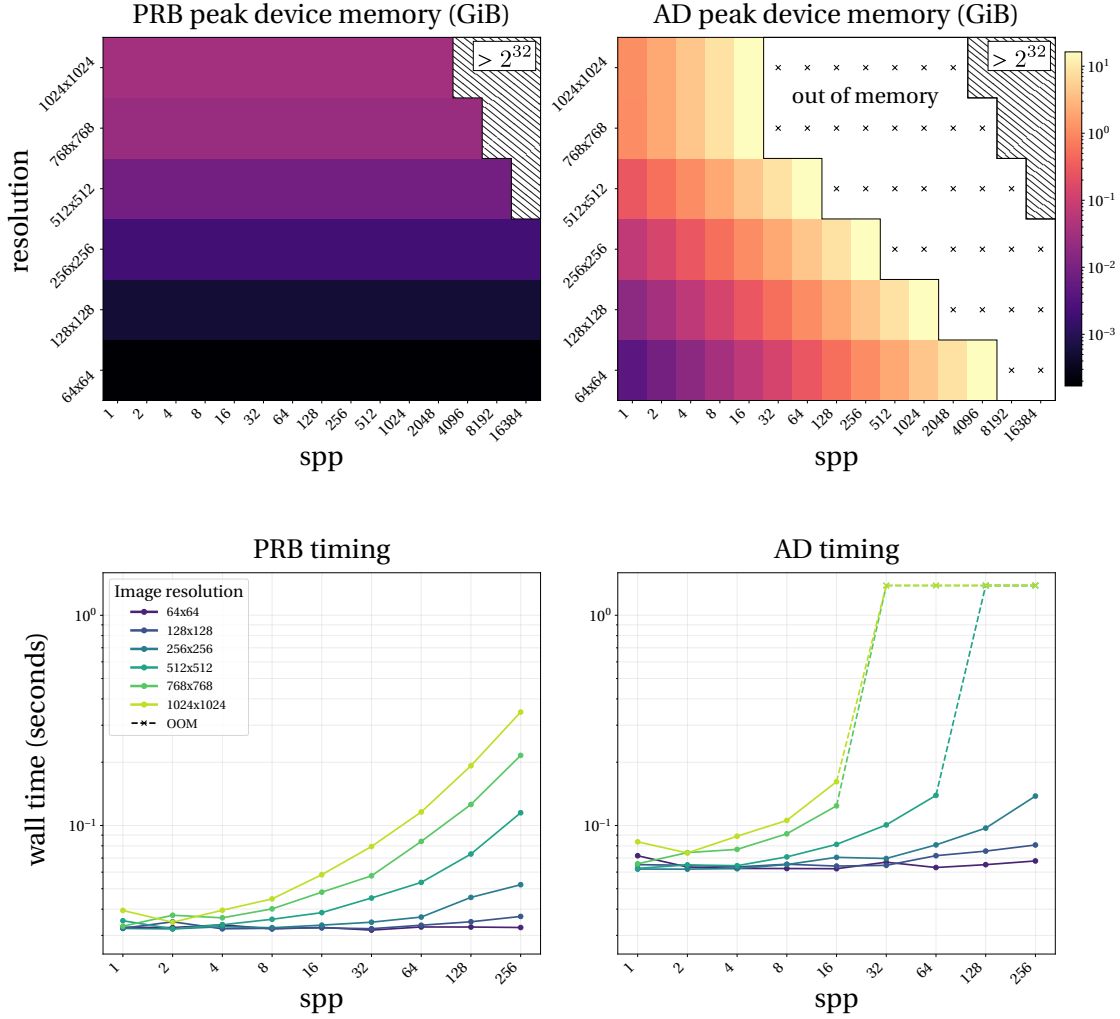


Figure 3.6: **Timing and memory comparison between conventional AD and PRB.** The plots compare a taped reverse-mode implementation and path replay backpropagation as the number of rays increases. **(top)** Conventional reverse mode becomes limited by the need to retain per-bounce state, whereas PRB replays the same sampled paths and requires a much smaller memory footprint. The shaded upper-right region corresponds to simulating more rays than the kernel-index range (2^{32}) permits in one kernel. This is not a limitation of PRB. **(bottom)** Despite the overhead of replaying paths, PRB is generally faster than the taped version when tested with the Mitsuba implementation.

Algorithm 3.2: **Replay/adjoint pass for PRB**. The replay pass retraces the same random walk and reconstructs reverse-mode suffix weights on the fly.

```
def sample_path_adjoint(ray, L, dL, rng):
    beta = 1.0
    for i in range(N):
        si = intersect(ray)
        Le = eval_emitter(si, -ray.d)
        backward_grad(Le, dL * beta)
        L -= beta * Le

        wo, bsdf_val, bsdf_pdf = sample_bsdf(si, -ray.d, rng)
        bsdf_weight = bsdf_val / bsdf_pdf
        backward_grad(bsdf_weight, dL * (L / bsdf_weight))

    beta *= bsdf_weight
    ray = spawn_ray(si, wo)
```

and accumulates parameter gradients via

$$\partial_{\pi} \ell = \sum_{i=0}^{N-1} \mathbf{a}_{i+1}^{\top} (\partial_{\pi} h(\mathbf{s}_i; \pi)). \quad (3.19)$$

In general, the adjoint $\mathbf{a}_i$ has the same dimension as the loop state $\mathbf{s}_i$. There is no constant-size record analogous to the stored final L that would allow reconstructing all $\mathbf{a}_i$ by a single replay without storing intermediate states or performing checkpointed recomputation.

Summary The main point of this section is that PRB leverages special properties of light transport to compute the same derivative as a naïve reverse-mode implementation, using memory that is constant with respect to path depth and time that is linear in the number of scattering events.

Later sections will add continuous geometry motion and visibility-induced boundary terms. Those extensions change the local derivative terms, but the arithmetic role of replay remains the same: it reconstructs the downstream weight required by reverse mode.

Our discussion in this section is deliberately focused on the arithmetic aspect of the problem. Section 3.4.3 revisits PRB from a transport viewpoint and explains how replay can be interpreted as light transport rather than as a generic program reversal.

3.4 A derivative transport viewpoint

The previous section explained PRB in arithmetic terms: a path contribution can be written as a sum of local derivative terms weighted by path suffixes, and replay reconstructs those

suffix weights without a tape. That viewpoint explains how the algorithm works. We now step back and ask why rendering has this special structure that allows reverse-mode differentiation without storing a trace.

Rather than differentiating a primal estimator, it is more principled to differentiate the transport equations themselves. Once we do that, the derivative is no longer merely “the output of AD on a renderer implementation.” It becomes a transport quantity on its own. This is the central insight behind *radiative backpropagation* (RB): reverse-mode differentiation of rendering can be formulated as an adjoint light-transport simulation rather than as the mechanical reversal of a taped primal program [28].

This viewpoint matters for two reasons. It explains why gradient computation can be made comparable in cost to primal rendering without a large tape, and it separates the design of the differential estimator from that of the primal estimator. This separation is important: the primal image estimate, the adjoint signal, and the derivative estimator do not have to reuse the same samples. Using decorrelated simulations for these roles is standard in inverse rendering and already appears in earlier heterogeneous inverse-scattering work [36]. The primal renderer remains important, but mainly as a source of light path samples and radiance values rather than as the only permissible route to the derivative. To keep the derivation focused, we begin in the fixed-geometry regime, where geometry and visibility are held fixed under differentiation. Continuous geometry motion and visibility-induced discontinuities will enter later in the chapter, but the core message of this section already appears in the simpler setting.

3.4.1 Differential radiance as a transport quantity

Consider the surface rendering equation in projected-solid-angle form:

$$L_o(\mathbf{x}, \boldsymbol{\omega}_o) = L_e(\mathbf{x}, \boldsymbol{\omega}_o) + \int_{\mathcal{S}^2} f_s(\mathbf{x}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o) L_i(\mathbf{x}, \boldsymbol{\omega}_i) d\boldsymbol{\omega}_i^\perp. \quad (3.20)$$

Differentiating with respect to a scalar scene parameter π gives

$$\begin{aligned} \partial_\pi L_o(\mathbf{x}, \boldsymbol{\omega}_o) = & \underbrace{\partial_\pi L_e(\mathbf{x}, \boldsymbol{\omega}_o)}_{(1) \text{ differential emission}} \\ & + \int_{\mathcal{S}^2} \left(\underbrace{L_i(\mathbf{x}, \boldsymbol{\omega}_i) \partial_\pi f_s(\mathbf{x}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o)}_{(2) \text{ differential scattering source}} + \underbrace{f_s(\mathbf{x}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o) \partial_\pi L_i(\mathbf{x}, \boldsymbol{\omega}_i)}_{(3) \text{ transport of differential radiance}} \right) d\boldsymbol{\omega}_i^\perp. \end{aligned} \quad (3.21)$$

This equation can be read as an energy balance for a new quantity: *differential radiance* (Figure 3.7). Three terms appear:

- (1) If an emitter depends on π , it directly emits differential radiance through $\partial_\pi L_e$.
- (2) When ordinary incident radiance L_i interacts with a material whose scattering law depends

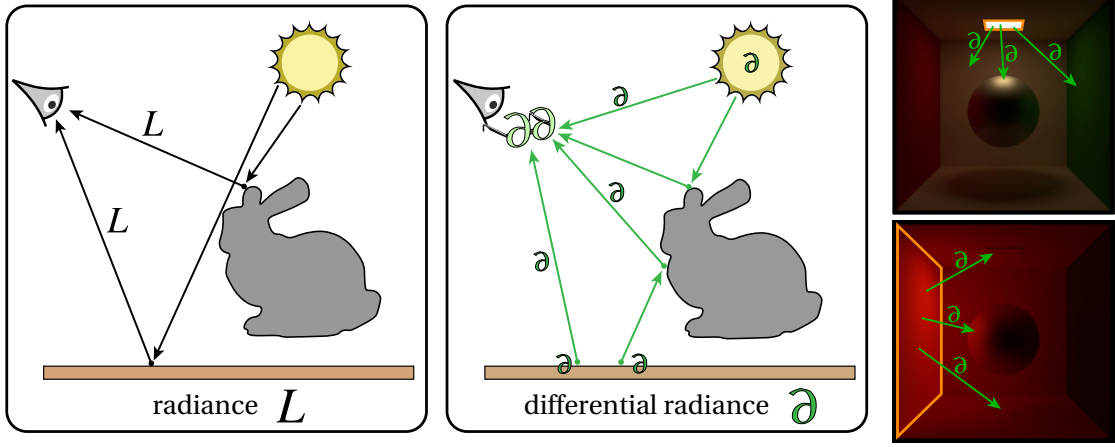


Figure 3.7: **A derivative transport viewpoint.** **Left:** primal rendering transports radiance from emitters to the sensor. **Middle:** after differentiation, parameter-dependent emission and scattering events act as sources of *differential radiance*, which is transported by the same visibility and scattering structure as ordinary light. **Right:** signed image derivatives for two example perturbations (emitter intensity and wall albedo), showing the image-space effect of those transported differential sources.

on π , part of that ordinary radiance is converted into differential radiance.

- (3) Once created, differential radiance is transported by the same propagation-and-scattering kernel as ordinary radiance: the term $f_s \partial_\pi L_i$ shows that it scatters according to the primal BSDF.

It is useful to group the local source terms into an effective differential emission

$$Q_\pi(\mathbf{x}, \boldsymbol{\omega}_o) := \partial_\pi L_e(\mathbf{x}, \boldsymbol{\omega}_o) + \int_{\mathcal{S}^2} L_i(\mathbf{x}, \boldsymbol{\omega}_i) \partial_\pi f_s(\mathbf{x}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o) d\boldsymbol{\omega}_i^\perp. \quad (3.22)$$

Then the differential transport equations become

$$\partial_\pi L_o = Q_\pi + K \partial_\pi L_i, \quad \partial_\pi L_i = G \partial_\pi L_o, \quad (3.23)$$

where, as usual,

$$(Kh)(\mathbf{x}, \boldsymbol{\omega}_o) := \int_{\mathcal{S}^2} f_s(\mathbf{x}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o) h(\mathbf{x}, \boldsymbol{\omega}_i) d\boldsymbol{\omega}_i^\perp, \quad (Gh)(\mathbf{x}, \boldsymbol{\omega}) := h(r(\mathbf{x}, \boldsymbol{\omega}), -\boldsymbol{\omega}). \quad (3.24)$$

Eliminating $\partial_\pi L_i$ yields

$$\partial_\pi L_o = Q_\pi + KG \partial_\pi L_o = (I - KG)^{-1} Q_\pi =: S Q_\pi, \quad (3.25)$$

with the usual light-transport solution operator S .

The key point is that the derivative obeys the same propagation law as ordinary radiance; only the source term changes. In the fixed-geometry regime, differentiating rendering therefore

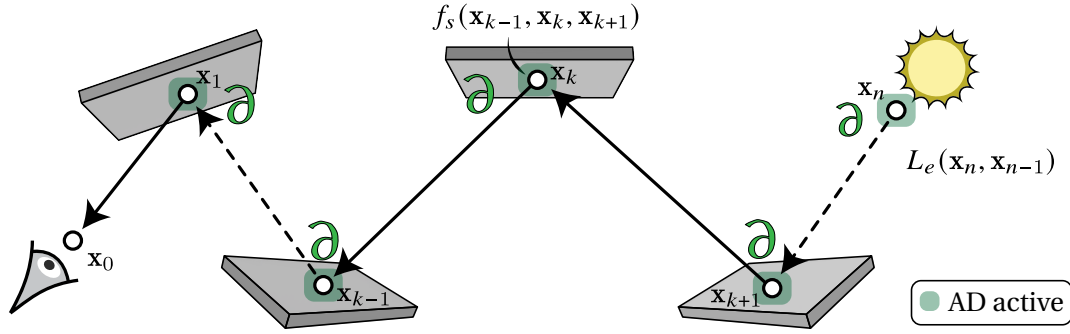


Figure 3.8: **Local AD inside a transport simulation.** Only the local source terms are differentiated with AD (e.g., BSDF computations). The global propagation of those derivatives is handled by an ordinary transport algorithm, so the path-tracing loop itself is not differentiated end to end.

does not create a new transport operator. It creates a new transported quantity with source Q_π . That is why a separate differential transport simulation is the natural viewpoint here.

A detail is worth stating explicitly. The above statement is exact in the fixed-geometry setting considered here. When geometry moves, additional derivative source terms appear, corresponding to discontinuous derivatives, but these derivative sources are still transported by the same kernel as ordinary radiance.

Local automatic differentiation This differential-radiance transport viewpoint uses automatic differentiation locally, where derivatives are created. Figure 3.8 illustrates the separation: we still rely on AD for smooth local computations such as texture lookups, BSDF evaluation, and intersection differentials. The full path-tracing loop is then treated as a transport estimator for the differential quantity rather than as one large taped program. The global propagation is handled instead by a transport algorithm tailored to the differential quantity itself.

In practice, this means that a differentiable renderer implementation often looks much more like an ordinary renderer than a taped AD program: most of the code remains undifferentiated, and only some local terms are differentiated and treated as differential sources. This greatly reduces the burden on the AD system, since differentiation no longer needs to be carried across long stochastic path-tracing loops. The local derivative fragments can be produced by a small computation graph, by a local tape, or by program-transformation/JIT machinery; after that, the ray-tracing framework only needs to transport the resulting differential sources.

3.4.2 Adjoint radiance

Just as primal rendering has a duality between radiance and importance, differential transport has a corresponding sensor-side quantity that receives emitted differential radiance.

Let pixel measurements be written as

$$I_j = \langle W_j, L_i \rangle, \quad (3.26)$$

where W_j denotes the sensor importance of pixel j . Define the image-space adjoint signal

$$\delta \mathbf{I} := \nabla_{\mathbf{I}} \mathcal{L}, \quad \delta I_j = \frac{\partial \mathcal{L}}{\partial I_j}. \quad (3.27)$$

We can fold these pixel weights into an *emitted adjoint radiance*

$$A_e(\mathbf{x}, \boldsymbol{\omega}) := \sum_j \delta I_j W_j(\mathbf{x}, \boldsymbol{\omega}). \quad (3.28)$$

This quantity should be read as a sensor-emitted field: for a pinhole camera, it behaves like a textured projector that emits sensitivity into the scene.

In an optimization, $\delta \mathbf{I}$ is the image-space signal supplied by the loss: it tells us which pixels should go up or down to reduce $\mathcal{L}$. Equation (3.28) lifts that signal from the sensor back into the scene. The field A_e therefore answers a concrete question: if a point in the scene were to send a little more radiance toward the camera along direction $\boldsymbol{\omega}$, how much would the loss change? Adjoint radiance is thus the current image mismatch rewritten in transport form. It tells the renderer where the error matters, so that when this back-propagated sensitivity meets a local differential source Q_π , their inner product contributes directly to the parameter gradient.

For a single parameter π , the desired gradient is

$$\partial_\pi \mathcal{L} = \langle A_e, \partial_\pi L_i \rangle. \quad (3.29)$$

Using $\partial_\pi L_i = GSQ_\pi$ and the usual reciprocity assumptions of light transport, Veach's operator formulation implies that the relevant transport operators are self-adjoint under a compatible inner product. Hence

$$\partial_\pi \mathcal{L} = \langle A_e, GSQ_\pi \rangle = \langle GSA_e, Q_\pi \rangle. \quad (3.30)$$

This suggests defining an *adjoint incident radiance*

$$A_i := GSA_e, \quad (3.31)$$

so that the final gradient becomes the local inner product

$$\partial_\pi \mathcal{L} = \langle A_i, Q_\pi \rangle. \quad (3.32)$$

Equation (3.32) suggests that the sensor emits adjoint radiance, the scene transports it with the same operator as ordinary light, and the gradient is accumulated wherever that adjoint field encounters a local differential source Q_π .

Equivalently, we can define outgoing and incident adjoint radiance by

$$A_i = G A_o, \quad A_o = A_e + K A_i. \quad (3.33)$$

These are the adjoint analogues of the primal transport equations.

This viewpoint gives an algorithmic advantage: instead of transporting a separate differential field for every parameter, reverse mode transports a *single* adjoint radiance field and lets the parameter dependence enter only through the local source term Q_π . That is the fundamental reason why reverse mode scales to high-dimensional parameter spaces.

3.4.3 Path replay backpropagation revisited

PRB can be interpreted as a replay-based Monte Carlo estimator of the adjoint transport problem. At a differentiable interaction point, two transported quantities meet. Ordinary incident radiance L_i arrives from the light side and determines how much primal radiance is available to be converted into a differential source. Adjoint radiance A_i arrives from the sensor side and determines how strongly a perturbation of outgoing radiance at that point affects the loss. The local parameter derivative couples these two fields through Q_π , and their inner product produces the gradient.

A useful mental picture is to freeze one interaction point $\mathbf{x}$ and replace the rest of the scene, as seen from $\mathbf{x}$, by a virtual textured emitter that emits exactly the incident radiance field $L_i(\mathbf{x}, \cdot)$. For the purpose of differentiating the local BSDF or emission at $\mathbf{x}$, nothing else about the downstream scene matters: all later bounces, geometry, and emitters have been compressed into that directional illumination. Likewise, the entire sensor side can be compressed into the adjoint field A_i , which tells us how much the loss cares about outgoing radiance in each direction. Under this viewpoint, differentiating one interaction is a local shading derivative between a virtual emitter carrying L_i and a virtual receiver carrying A_i .

This is the transport interpretation of radiative backpropagation [28]: sensors emit adjoint importance, the scene transports it like ordinary light, and parameter-dependent interactions create differential sources measured by that adjoint field.

The cost issue also becomes intuitive from this picture. A direct unbiased implementation would estimate the primal incident radiance L_i inside Q_π independently at many vertices. This repeatedly estimates downstream transport that a sampled path has already explored. Along a path of length k , this repeated re-estimation leads to the quadratic work identified by Vicini et al. [29].

PRB is based on the observation that, for a path sampled by the derivative estimator, this downstream transport does not need to be recomputed from scratch at each vertex. In a three-phase view, one simulation estimates the primal image and image-space adjoint signal. A separate forward light-transport sample supplies a light-side continuation for derivative

estimation, and the replay phase retraces that same derivative sample to expose suffix weights. For a vertex $\mathbf{x}_i$ on the derivative path, the quantity needed by reverse mode is exactly the suffix contribution S_i from Section 3.3.1: the remainder of the sampled path acts as the virtual emitter imagined above, reduced to a single Monte Carlo continuation. Replaying the same random numbers reveals the suffix one bounce at a time, while the prefix carries the corresponding sensor-side weight.

Thus, PRB is a path-sampled estimator of the same adjoint transport problem as RB, with replay replacing recursive recomputation. Section 3.3 wrote the method as a sum of local derivatives weighted by path suffixes; the present section writes the same quantity as the inner product between adjoint transport and local differential sources. The former is the arithmetic decomposition needed to implement the algorithm. The latter is the transport intuition that explains what that decomposition means.

3.5 Design space in the differential problem

Once primal and differential transport are separated, derivative estimation becomes its own Monte Carlo problem. This can be seen already for a simple image loss

$$\mathcal{L}(\pi) = \frac{1}{2} \sum_j \left(I_j(\pi) - I_j^* \right)^2,$$

where the image values I_j and the derivatives $\partial_\pi I_j$ need not be estimated with the same sampler or even the same path parameterization. Because the differential problem fundamentally differs from the primal one, estimators that are effective for primal rendering need not remain effective for the derivative. This motivates a broader design space for the differential estimator.

This design freedom manifests in two primary ways: first, in how we choose to sample the differential paths (path sampling), and second, in how we parameterize those paths to compute local derivatives (path parameterization).

3.5.1 Differential sampling strategies

The most direct way to exercise this design freedom is to use a tailored sampling density for the differential problem rather than reusing the primal one. This is made possible by the decoupled transport viewpoint, which treats the derivative as a new transport quantity with its own source term.

Formally, define the derivative integral

$$D_\pi := \partial_\pi I(\pi) = \int_X \partial_\pi f(x, \pi) \, dx. \quad (3.34)$$

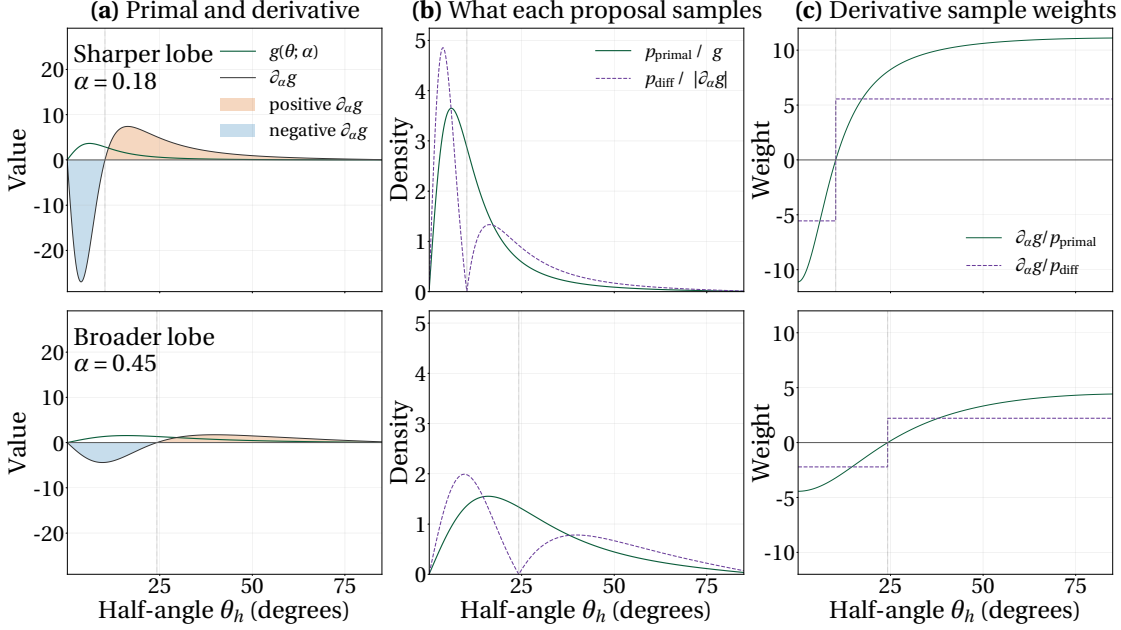


Figure 3.9: **Why the differential problem can require a different sampling density.** (a) shows a one-dimensional GGX roughness slice $g(\theta_h, \alpha)$ together with its derivative $\partial_\alpha g$ for two roughness values. (b) compares a primal sampling density $p_{\text{primal}} \propto g$ against a differential sampling density $p_{\text{diff}} \propto |\partial_\alpha g|$. (c) shows the corresponding derivative sample weights $\partial_\alpha g / p$. The primal density is matched to the primal lobe, but the differentiated integrand has a distinct signed structure concentrated in different regions, so a differential density can be substantially better matched.

A one-sample Monte Carlo estimator using a specialized differential sampling density p_{diff} is

$$\hat{D}_\pi = \frac{\partial_\pi f(X, \pi)}{p_{\text{diff}}(X)}, \quad X \sim p_{\text{diff}}. \quad (3.35)$$

If $p_{\text{diff}}(x) > 0$ wherever $\partial_\pi f(x, \pi) \neq 0$, then $\mathbb{E}[\hat{D}_\pi] = D_\pi$. The “hat” notation is important: one sample is only a noisy estimate, and practical renderers average many such estimates. When the sampling density itself depends on π , this equation should be read as a detached estimator unless the sampling map and PDF are explicitly included in an attached estimator such as Equation (3.39). The differential sampling density is chosen for the derivative problem and can provide near-optimal importance-sampling efficiency.

This freedom is especially valuable when the differentiated integrand changes shape substantially. A canonical example is the derivative of microfacet roughness: the primal BSDF sampler is designed for the primal lobe, while the roughness derivative has a qualitatively different positive/negative structure. In that case, a specialized differential sampling density can be markedly better than any reuse of the primal sampler (Figure 3.9).

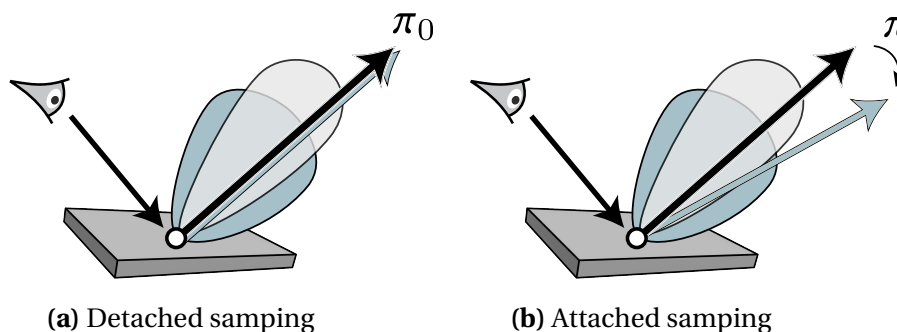


Figure 3.10: **Detached versus attached differentiation of BSDF importance sampling.** When a material parameter, such as roughness (π), increases, the underlying BSDF lobe widens. (a) In *detached* sampling, the previously sampled ray direction remains fixed; only the BSDF evaluation changes. (b) In *attached* sampling, the sampling routine itself is differentiated, causing the generated ray to shift infinitesimally to follow the changing probability distribution.

Reusing the primal sampling strategy. While custom differential sampling strategies can be highly effective, they require designing a dedicated sampler for the derivative integrand. In the more general case, it is much easier and often more practical to reuse the path sampled during the primal phase. This convenience aligns with a core principle of automatic differentiation: what is good for function values is typically good—or at least a reasonable starting point—for their derivatives [35].

When reusing the primal sampling strategy, we must still decide how to handle the sampling process—specifically, whether we treat it as attached or detached.

As an illustrative example, consider a BSDF sampling step at a glossy surface in Figure 3.10. In the primal renderer, we typically sample an outgoing direction using a distribution that tracks the shape of the BSDF lobe, which depends on parameters such as roughness. When taking the derivative with respect to roughness, we must decide whether differentiation views the sampled direction as fixed in place or as moving infinitesimally along with the changing lobe. This leads to two natural conventions:

- (1) **Detached sampling.** The sampling routine is frozen during differentiation. Under BSDF differentiation, the sampled direction stays put (Figure 3.10a). The derivative originates from the change in the contribution function evaluated at that fixed light path.
- (2) **Attached sampling.** We differentiate through the sampling routine itself. Under BSDF differentiation, the sampled direction now moves infinitesimally with the parameter as the underlying probability distribution changes (Figure 3.10b).

Neither choice is uniformly superior, as demonstrated in Figure 3.11, which compares these two conventions when differentiating the roughness of a textured rough-glass panel with a microfacet BSDF. The resulting derivative image is highly structured because changing roughness shifts and blurs both refracted background texture and specular highlights. The attached and detached estimators both use 4096 samples per pixel for derivative estimation,

and the reference derivative is computed by finite differences with 524288 samples per pixel.

Attached sampling reduces variance when sample motion follows the integrand’s smooth, parameter-sensitive parts. However, moving the sample inherently differentiates factors *not* targeted by the original sampling density. When these remaining factors vary rapidly—such as high-frequency textures seen through rough glass—the attached terms can increase variance sharply.

A minimal formalization of attached/detached sampling. Following the review and notation of Zeltner et al. [30], consider an integral

$$I(\pi) = \int_X f(x, \pi) \, dx, \quad (3.36)$$

estimated using a sampling routine

$$u \sim \mathcal{U}(U), \quad x = T(u, \pi), \quad \hat{I}(\pi) = \frac{f(T(u, \pi), \pi)}{p(T(u, \pi), \pi)}. \quad (3.37)$$

This is the standard situation in rendering, where T may represent a BSDF sampler, an emitter sampler, a microfacet normal sampler, and so on.

In a *detached* strategy, the sampling map is explicitly frozen during differentiation:

$$\partial_\pi I(\pi) = \int_U \frac{\partial_\pi f(T(u, \pi_0), \pi)}{p(T(u, \pi_0), \pi_0)} \, du, \quad (3.38)$$

where π_0 denotes a detached (non-differentiated) copy of the current parameter value, evaluated numerically at the same primal value as π . The random sample stays exactly where it was generated; only the evaluated contribution changes.

In an *attached* strategy, we differentiate the full reparameterized estimator:

$$\partial_\pi I(\pi) = \int_U \partial_\pi \left[\frac{f(T(u, \pi), \pi)}{p(T(u, \pi), \pi)} \right] \, du. \quad (3.39)$$

Now the sample itself moves with the parameter under fixed random numbers.

Applying the chain rule to Equation (3.39) yields an important new ingredient: a sample-motion term proportional to the Jacobian $\partial_\pi T(u, \pi)$. This term is the mathematical manifestation of the intuition that “the sample follows the distribution.”

A useful mental model is to write the rendering integrand as a product $f(x, \pi) = g(x, \pi) h(x, \pi)$, where the sampling density is designed to match g perfectly while h collects everything else. If p is well matched to g , attached sampling still handles that factor well, but it also introduces an extra derivative term that differentiates h purely through sample motion. When h is smooth, this motion can be beneficial. When h varies rapidly, the derivative of h becomes large, and

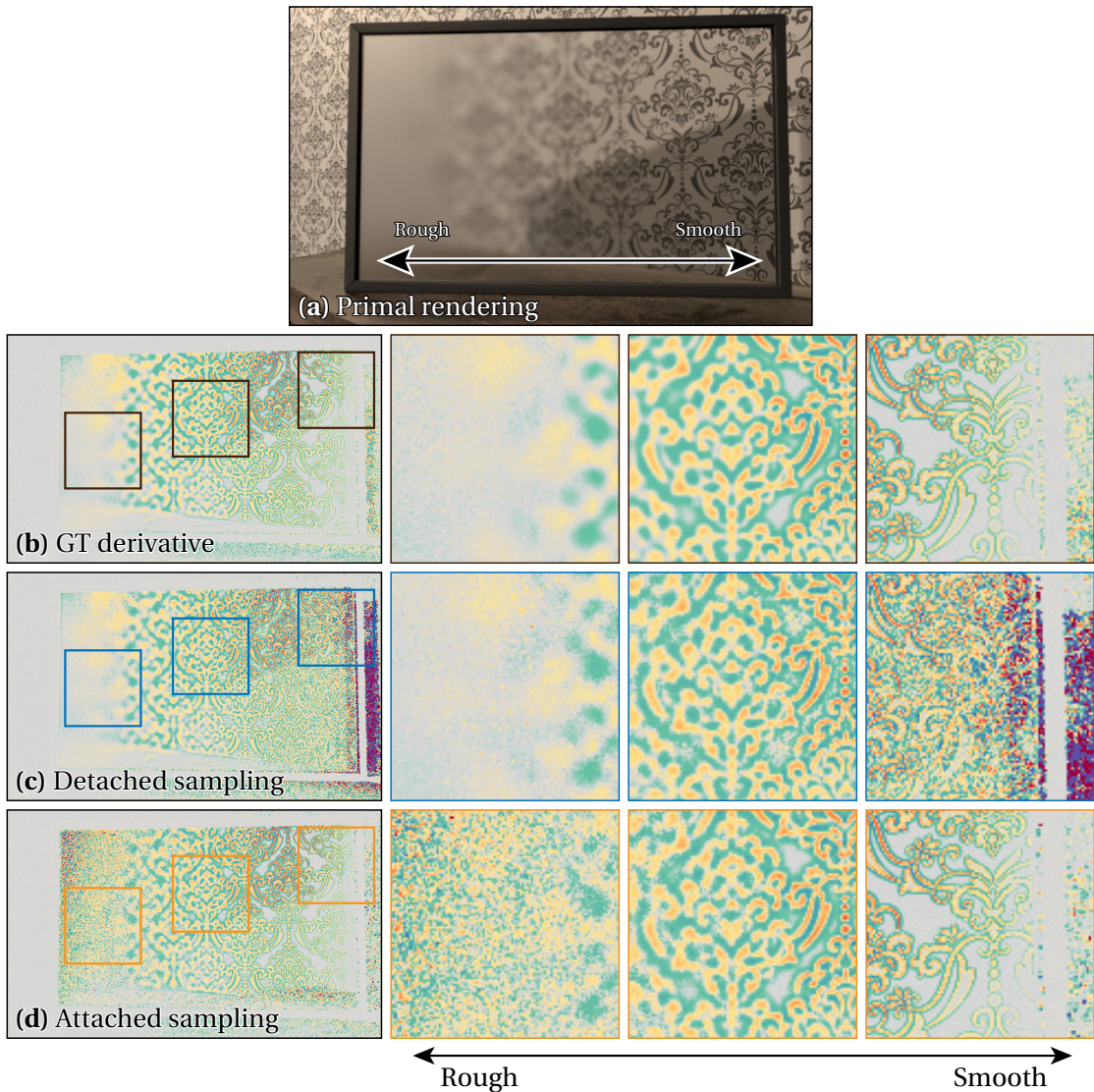


Figure 3.11: **Comparing attached and detached estimators for a roughness derivative.** (a) Primal rendering of a textured rough-glass panel. (b) A finite-difference reference for the derivative with respect to a global roughness increase. (c-d) The detached and attached estimators, both evaluated using the same light paths. They differ only in whether the BSDF sampling map is frozen or differentiated. The visible differences arise from variance and convergence behavior: neither estimator is meant to change the expected derivative image, but each requires different sample counts in different regions. Detached sampling better captures the lower-variance signal in rougher regions where high-frequency textures dominate, while attached sampling follows the smooth glossy transport more efficiently.

that extra term can dominate the variance. This is the key reason why attached sampling can be excellent in some regimes yet counterproductive in others.

The same point can be expressed in a control-variate-like form. For the attached estimator above,

$$\partial_\pi \frac{f(T(u, \pi), \pi)}{p(T(u, \pi), \pi)} = \frac{\partial_\pi f(T(u, \pi_0), \pi)}{p(T(u, \pi_0), \pi_0)} + \partial_\pi \left[\frac{f(T(u, \pi), \pi)}{p(T(u, \pi), \pi)} - \frac{f(T(u, \pi_0), \pi)}{p(T(u, \pi_0), \pi_0)} \right].$$

The added term has zero expectation under suitable smoothness and support assumptions, so it does not change the target derivative. Its variance, however, depends on the bracketed control function; when that function changes rapidly, attached sampling can be worse than the detached estimator.

3.5.2 Differential light path parameterization

The second form of design freedom arises when a light path has already been sampled, but its vertices are no longer valid under a geometry perturbation. The way these vertices are moved is known as the path parameterization. Importantly, “path parameterization” here does not refer to how the path was sampled, but rather to the local coordinate frame used to track derivatives. Whether the path came from differential sampling, BSDF sampling, next-event estimation, or path guiding is irrelevant to this discussion.

The preceding PRB section differentiated a *fixed* sampled path: the sequence of vertices $\bar{\mathbf{x}} = (\mathbf{x}_0, \dots, \mathbf{x}_k)$ was treated as fixed in space, and all derivatives came from smooth local factors evaluated along that path (BSDFs, emitters, $\dots$). When geometry moves, the vertices produced by ray tracing must move as well, even if the sequence of surfaces hit by the path remains unchanged.

Figure 3.12 separates two qualitatively different effects. In Figure 3.12(a), an infinitesimal perturbation changes the hit result itself: the sampled path jumps to a different path configuration, or even to a different subset of path space. These visibility-induced discontinuities are treated later in Section 3.6. In Figure 3.12(b), the same sequence of surfaces remains active, but the original hit points in world space are no longer valid; they must move to nearby positions to follow the perturbed geometry, and the connecting directions may move as well. This section focuses on the second continuous case, but the parameterization choices described here also affect how the visibility-induced derivatives are computed later.

For notation, we write the relevant geometry parameter as π , and we denote by π_0 the value at which the primal path was sampled. Vertices labeled $\mathbf{x}_i(\pi_0)$ therefore belong to the sampled primal path, while $\mathbf{x}_i(\pi)$ denotes the nearby position assigned to that vertex by a chosen differential path parameterization.

We compare three common choices: *recursive*, *RT*, and *UV/material-form* parameterizations. They agree on the primal contribution of the sampled path, but they assign geometry differen-

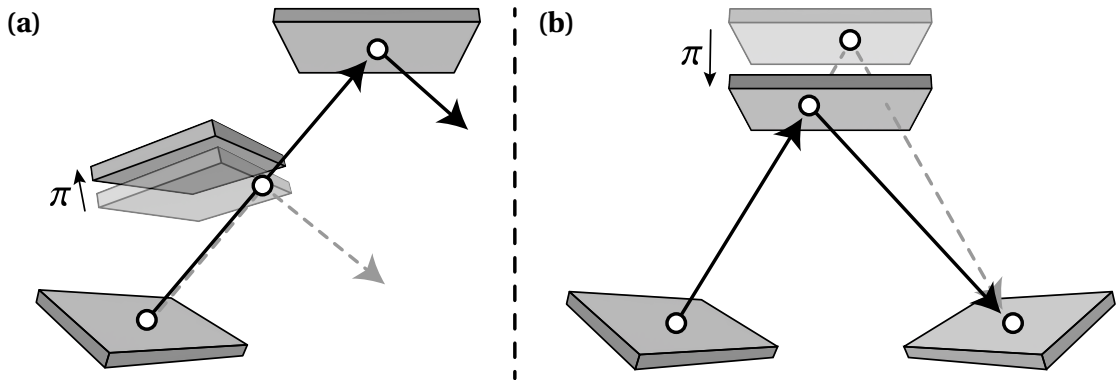


Figure 3.12: **Discontinuous versus continuous path motion under geometry changes.** (a) A perturbation can change the hit configuration itself, causing the sampled path to jump to a different path configuration; this is the visibility problem discussed later in Section 3.6. (b) Even when the same sequence of surfaces remains valid, the hit points and connecting directions move continuously. The present section studies this continuous part of the geometry derivative.

tials to intermediate vertices and directions in different ways, which can lead to very different variance behavior. This is not an exhaustive list of all possible parameterizations, but it covers the most representative ones used in practice.

Recursive parameterization. The *recursive* parameterization differentiates the actual path-construction loop, which is equivalent to applying end-to-end AD to the path tracer in the listing below. Each new ray is spawned from the already perturbed hit point, so the entire light path becomes coupled.

```
def trace_path(ray, rng, pi):
    for i in range(N):
        si = intersect(ray, scene(pi))
        vertices.append(si.p)
        wo = sample_bsdf(si, -ray.d, rng)
        ray = spawn_ray(si, wo) # Using the already moving si.p
    return vertices
```

Figure 3.13 visualizes this coupling. The gray dashed path is the primal path sampled at π_0 , while the black path is the neighboring path associated with π (in the space of all possible light paths). Once $\mathbf{x}_i$ moves to $\mathbf{x}_i(\pi)$, the ray toward the next bounce is spawned from that moved point, which in turn changes $\mathbf{x}_{i+1}(\pi)$, and so on.

Let $r_\pi(\mathbf{x}, \boldsymbol{\omega})$ denote the first scene intersection reached by ray tracing from $\mathbf{x}$ in direction $\boldsymbol{\omega}$ in the scene with parameter value π . Assuming that the hit configuration does not change, each

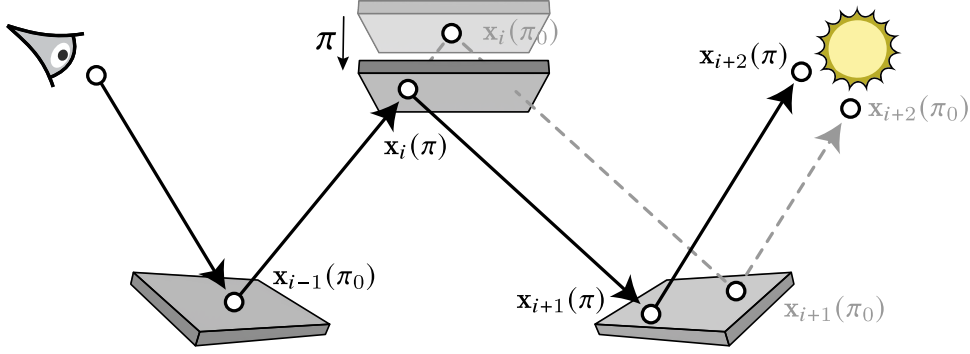


Figure 3.13: **Recursive parameterization.** The gray dashed segments denote the primal path sampled at π_0 , while the black segments denote the neighboring path associated with π . Because the ray that generates $\mathbf{x}_{i+1}(\pi)$ is spawned from the already moved vertex $\mathbf{x}_i(\pi)$, an infinitesimal perturbation propagates recursively to all downstream vertices. The figure emphasizes this long-range coupling rather than a specific formula.

vertex is defined by ray tracing from the previous one:

$$\mathbf{x}_i(\pi) = r_\pi(\mathbf{x}_{i-1}(\pi), \boldsymbol{\omega}_{i-1}(\pi)), \quad i = 1, \dots, k, \quad (3.40)$$

where $\boldsymbol{\omega}_{i-1}(\pi)$ is the outgoing direction produced at $\mathbf{x}_{i-1}(\pi)$ under the same underlying random choices. Applying the chain rule yields

$$\frac{\partial \mathbf{x}_i}{\partial \pi} = \frac{\partial r_\pi}{\partial \pi} + \frac{\partial r_\pi}{\partial \mathbf{x}_{i-1}} \frac{\partial \mathbf{x}_{i-1}}{\partial \pi} + \frac{\partial r_\pi}{\partial \boldsymbol{\omega}_{i-1}} \frac{\partial \boldsymbol{\omega}_{i-1}}{\partial \pi}, \quad (3.41)$$

which makes the coupling explicit: an infinitesimal perturbation of an early vertex propagates to all later vertices. This is the most literal meaning of “differentiate the integrator loop,” but it is also the least local one. A derivative term evaluated near bounce i can depend on the full upstream history of the path.

A local view for RT and UV. The remaining two parameterizations are best understood as *local conventions* for assigning the continuous geometry derivative to individual factors along the path. To make that precise, it is helpful to write a simple path contribution in local form. As a simple example, consider a plain BSDF random walk that terminates on an emitter, omitting MIS; the same idea carries over to richer estimators, but with heavier notation.

A path contribution with terminal emissive vertex $\mathbf{x}_k$ can be written as

$$L(\tilde{\mathbf{x}}) = L_e(\mathbf{x}_k, -\boldsymbol{\omega}_{k-1}) \prod_{i=1}^{k-1} T_i(\mathbf{x}_{i-1}, \mathbf{x}_i, \mathbf{x}_{i+1}), \quad (3.42)$$

where each T_i collects the terms at the non-emissive vertex $\mathbf{x}_i$ that depend on the incident and outgoing directions and on the local differential geometry (BSDF value, cosine factors,

Jacobians, and local PDF conventions). Here

$$\omega_{i-1} = \frac{\mathbf{x}_i - \mathbf{x}_{i-1}}{\|\mathbf{x}_i - \mathbf{x}_{i-1}\|}, \quad \omega_i = \frac{\mathbf{x}_{i+1} - \mathbf{x}_i}{\|\mathbf{x}_{i+1} - \mathbf{x}_i\|},$$

so T_i depends on the triplet $(\mathbf{x}_{i-1}, \mathbf{x}_i, \mathbf{x}_{i+1})$. The product runs only to $k-1$ because the terminal vertex $\mathbf{x}_k$ is the emitter.

Differentiating at the sampled primal path $\bar{\mathbf{x}}(\pi_0)$ gives

$$\partial_\pi L = (\partial_\pi L_e) \prod_{i=1}^{k-1} T_i + \sum_{i=1}^{k-1} (\partial_\pi T_i) L_e \prod_{\substack{j=1 \\ j \neq i}}^{k-1} T_j, \quad (3.43)$$

where all factors are evaluated at π_0 .

The message of Equation (3.43) is simply that the full derivative is assembled from a sum of local derivative terms, following the simple product rule. For two adjacent factors,

$$\partial_\pi [T_i T_{i+1}] = (\partial_\pi T_i) T_{i+1} + T_i (\partial_\pi T_{i+1}). \quad (3.44)$$

Since the shared vertex $\mathbf{x}_{i+1}$ appears in both factors, its motion does *not* have to be accounted for entirely inside $\partial_\pi T_i$. It can also appear one term later, inside $\partial_\pi T_{i+1}$, where $\mathbf{x}_{i+1}$ is itself the central vertex. This is exactly how the RT parameterization below should be read: freezing an endpoint inside the derivative of one local factor does not mean that the endpoint is globally frozen along the whole path.

Local ray-intersection parameterization. The RT parameterization keeps geometry differentiation local at the level of a single vertex neighborhood. When differentiating the local factor $T_i(\mathbf{x}_{i-1}, \mathbf{x}_i, \mathbf{x}_{i+1})$, RT freezes the neighboring vertices at their primal positions $\mathbf{x}_{i-1}(\pi_0)$ and $\mathbf{x}_{i+1}(\pi_0)$ and assigns geometry motion only to the central vertex $\mathbf{x}_i$ through ray tracing with fixed ray directions.

Figure 3.14 should therefore be read locally: the vertex $\mathbf{x}_i(\pi)$ is obtained by re-intersecting the primal incoming ray with the perturbed geometry, while the two neighboring vertices remain at their primal locations. Crucially, there is *no* recursive propagation of motion from $\mathbf{x}_{i-1}$ into $\mathbf{x}_i$ inside this local derivative.

Define the RT motion of the central vertex by

$$\mathbf{x}_i^{\text{RT}}(\pi) = r_\pi(\mathbf{x}_{i-1}(\pi_0), \omega_{i-1}(\pi_0)). \quad (3.45)$$

The corresponding RT contribution to the derivative of the i -th local factor is

$$\partial_\pi T_i|_{\text{RT}} = \left. \frac{\partial}{\partial \pi} T_i(\mathbf{x}_{i-1}(\pi_0), \mathbf{x}_i^{\text{RT}}(\pi), \mathbf{x}_{i+1}(\pi_0)) \right|_{\pi=\pi_0}. \quad (3.46)$$

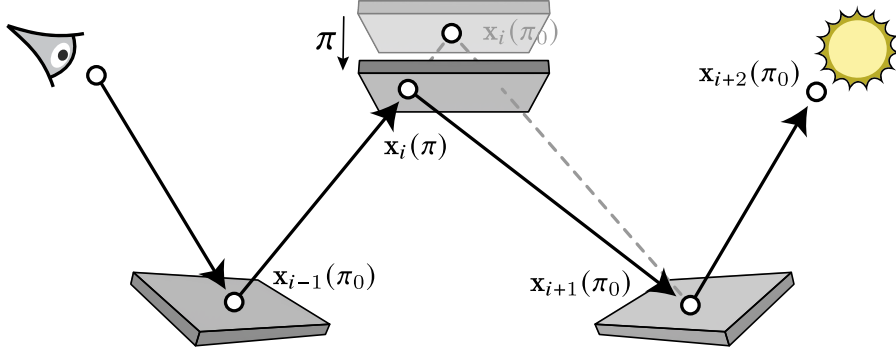


Figure 3.14: **Local ray-intersection (RT) parameterization.** When differentiating the local factor at $\mathbf{x}_i$, the neighboring vertices remain frozen at their primal positions $\mathbf{x}_{i-1}(\pi_0)$ and $\mathbf{x}_{i+1}(\pi_0)$. Only the central vertex is moved by re-intersecting the primal incoming ray with the perturbed geometry to obtain $\mathbf{x}_i(\pi)$. In particular, the motion of $\mathbf{x}_i$ does not inherit any recursive propagation from $\mathbf{x}_{i-1}$ inside this local derivative.

This is purely a bookkeeping rule. The motion of $\mathbf{x}_{i+1}$ is not ignored; it is simply *not attributed to this term*. It reappears when the neighboring factor is differentiated, namely in $\partial_\pi T_{i+1}|_{\text{RT}}$, where $\mathbf{x}_{i+1}$ becomes the central vertex. RT should therefore not be interpreted as one globally coupled perturbed path $\bar{\mathbf{x}}(\pi)$. It is a local convention for the terms in Equation (3.43).

UV parameterization (material form). The *UV* parameterization, also known as the *material form* [27], takes a different viewpoint. Instead of asking where a detached ray hits the perturbed scene, it asks where the same material point moves when the surface deforms. Here, a *material point* means a fixed point in the surface’s intrinsic parameter domain, represented, for example, by fixed barycentric coordinates on a triangle or fixed (u, v) coordinates on a smooth patch. Each intersection stores local surface coordinates (e.g., barycentric coordinates on a triangle or (u, v) coordinates on a smooth patch), and the world-space position is reconstructed by evaluating the surface map at those fixed coordinates.

Figure 3.15 illustrates this attached motion. The primal point $\mathbf{x}_i(\pi_0)$ and perturbed point $\mathbf{x}_i(\pi)$ correspond to the same material point on the surface; only the embedding of the surface in world space changes.

If the surface containing $\mathbf{x}_i$ is described by a parameterization $\Phi_{i,\pi}$, we write

$$\mathbf{x}_i^{\text{UV}}(\pi) = \Phi_{i,\pi}(u_i, v_i), \quad (3.47)$$

where (u_i, v_i) are held *fixed* during differentiation. Unlike RT, the UV view answers the question

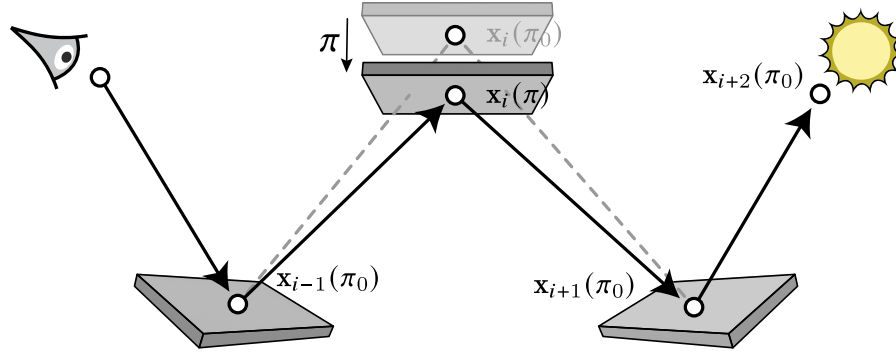


Figure 3.15: **UV/material-form parameterization.** The primal vertex $\mathbf{x}_i(\pi_0)$ and perturbed vertex $\mathbf{x}_i(\pi)$ share the same surface coordinates (u_i, v_i) (or barycentrics); only the surface embedding changes. This represents the motion of the same material point under deformation. It contrasts with RT, where $\mathbf{x}_i(\pi)$ is defined by re-intersecting a detached ray with the perturbed scene.

“where does the same material point move when the surface deforms?” rather than “where does the ray hit the perturbed scene?”

In the same local three-point form as above, the corresponding derivative of T_i is

$$\partial_\pi T_i|_{UV} = \left. \frac{\partial}{\partial \pi} T_i(\Phi_{i-1,\pi}(u_{i-1}, v_{i-1}), \Phi_{i,\pi}(u_i, v_i), \Phi_{i+1,\pi}(u_{i+1}, v_{i+1})) \right|_{\pi=\pi_0}, \quad (3.48)$$

with the understanding that each vertex uses the surface map of the surface on which it lies.

Variance comparison. Away from visibility events, recursive, RT, and UV are alternative estimators of the same continuous geometry derivative. What changes is estimator behavior: variance patterns and implementation structure.

Figure 3.16(a) is only a schematic of the layout and should not be read as an exact depiction of the experimental geometry. The actual simulation uses much larger diffuser plates so that the tested configurations stay away from visibility changes (a ray cannot miss a diffuser plate). The emitter texture is padded by a black border and linearly interpolated. The finite-difference reference in Figure 3.16(b) uses 262144 spp. RT and UV use 32768 spp. Recursive uses the same total sample count, but it is accumulated over 256 independent runs of 128 spp each to avoid exceeding the memory limit of the taped implementation. The rightmost columns visualize signed bias with a diverging colormap: gray indicates low bias, while persistent red or blue structure indicates systematic error with respect to the finite-difference reference.

In Figure 3.16, RT is the most stable overall, and UV generally has higher variance.

Figure 3.17 uses the same scene and motions but replaces the emitter texture with a much higher-frequency pattern. The gap between UV and RT narrows in some cases, because

the high-frequency emission texture does not strongly affect the UV estimator when the hit point has a fixed UV coordinate. Recursive now yields much higher variance, but can still occasionally outperform RT and UV in some settings.

No single parameterization wins universally. The best estimator depends on the scene and the motion (i.e., the differentiation task).

Theoretical comparison. Conceptually, recursive couples all downstream interactions. RT and UV are both local, but they describe different motions: RT re-intersects a detached ray, whereas UV follows the same material point on each surface. This distinction matters both for implementation and for theory. RT and UV fit naturally into the local-term/suffix-weight structure used by PRB in Section 3.3; for the recursive parameterization, a comparably local replay decomposition is not obvious because the derivative at one bounce already contains upstream motion. This parameterization choice also reappears in Section 3.6, where the normal-speed term of a moving visibility boundary depends on how path vertices are allowed to move; from that viewpoint, UV/material-form motion can lower the variance of boundary-term estimation.

Seen this way, geometry differentiation resembles rendering itself: the target quantity is fixed, but there are multiple valid Monte Carlo estimators for it. Recursive, RT, and UV differ in how they let a sampled path move under an infinitesimal perturbation, and that choice strongly affects variance, bias patterns, and implementation structure.

3.5.3 A practical implementation viewpoint

The main takeaway of the transport viewpoint is to treat gradient computation as a separate rendering problem, with its own design space and estimator choices.

In practice, debugging and implementation become much easier when the differentiable renderer is structured to separate primal rendering from derivative computation as follows:

1. **Primal rendering:** Sample paths and compute radiance in a fully detached way. Any algorithm can be used here, since derivatives are recomputed later when replaying the same paths.
2. **Local differential evaluation** (for correctness): For each ray segment, regardless of how it contributes to the primal image, always locally differentiate *all* terms in its path-space formulation with a chosen path parameterization. This prevents errors such as hidden cancellations or missing Jacobians.
3. **Adjoint transport** (for scalability): PRB is recommended whenever possible. Even when peak memory is not the limiting factor, PRB can be faster because it avoids writing and rereading large per-path traces from global memory; the relevant bottleneck is memory bandwidth rather than SIMD utilization.

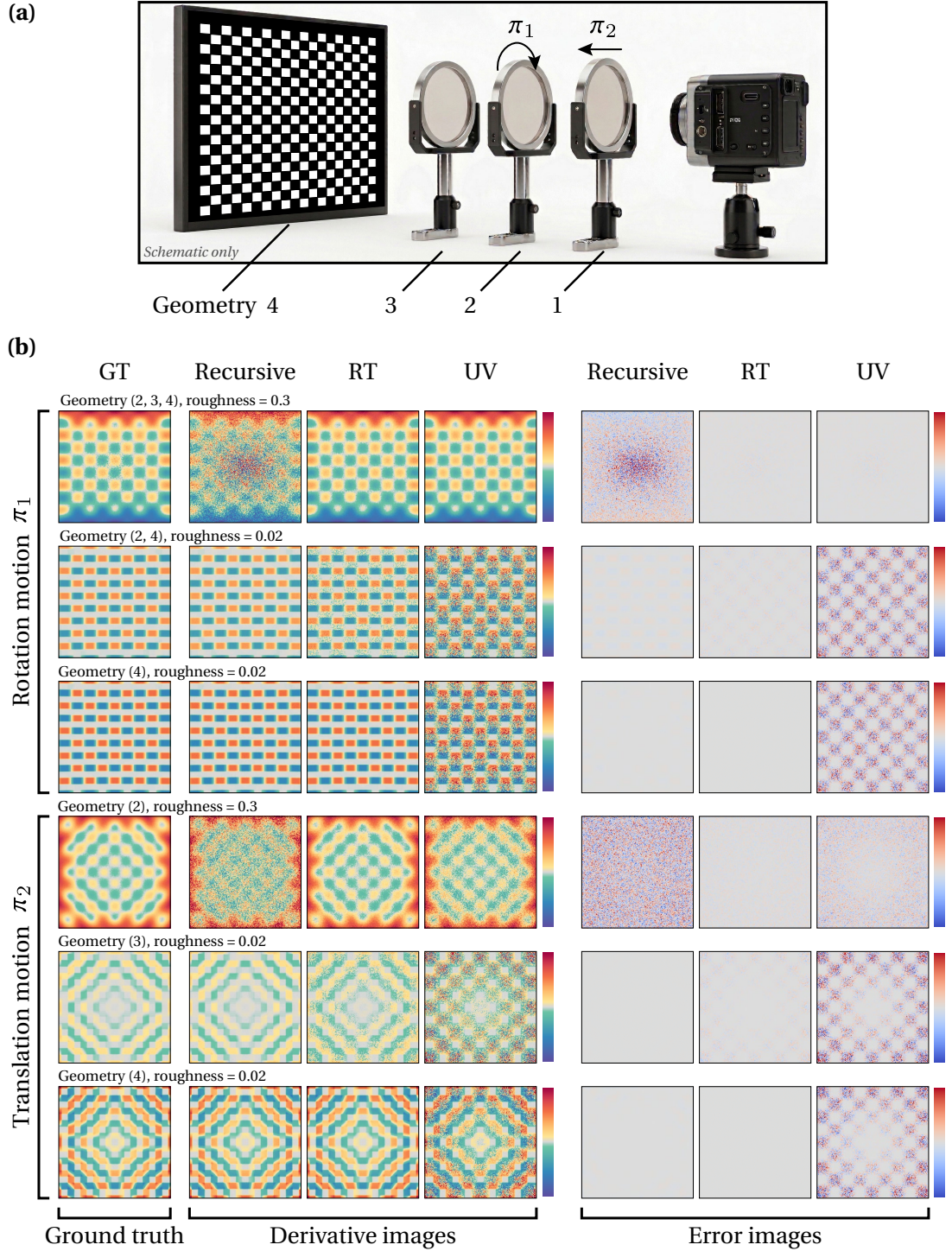


Figure 3.16: **Qualitative comparison of path parameterizations.** (a) Schematic of the optical bench layout used in the experiment: a textured area emitter (Geometry 4), three GGX diffuser plates (Geometries 1–3), and a camera. The component illustration is schematic and not drawn to exact dimensions. The arrows π_1 and π_2 indicate the tested rotation and translation motions. (b) Finite-difference reference images (GT), derivative images for recursive, RT, and UV, and the corresponding signed bias images for several choices of moving geometry and roughness.

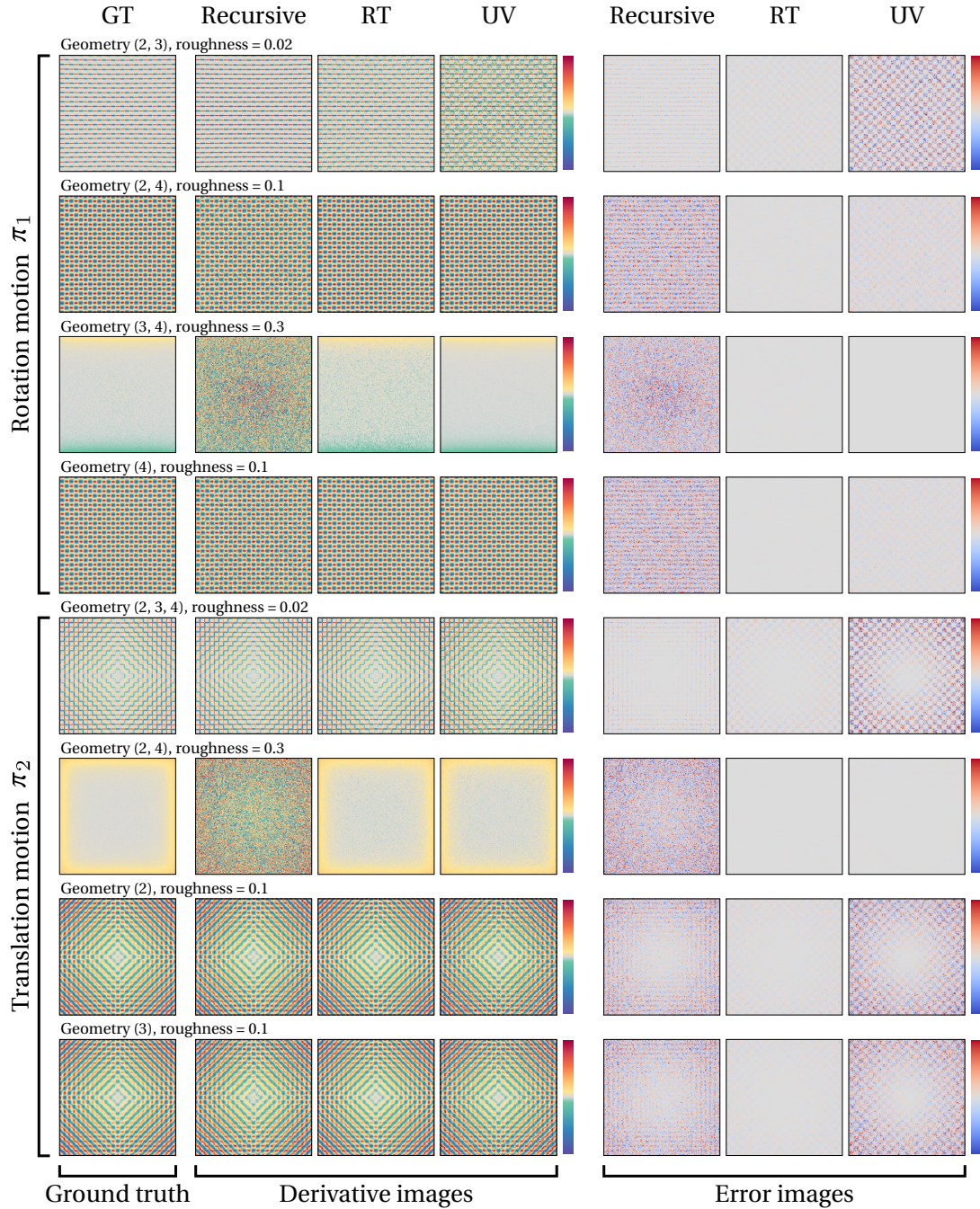


Figure 3.17: **The same comparison with a higher-frequency emitter texture.** The scene, motions, and sample counts match Figure 3.16; only the emitter pattern is changed to contain much finer spatial detail.

Within that structure, a conservative set of defaults is:

- **Path sampling:** Start from reusing primal samples with detached sampling. Attached sampling and custom differential proposals are powerful tools, but they require careful design to avoid bias and variance pitfalls, so they are best introduced after a working detached implementation is in place.
- **Path parameterization:** For the continuous geometry term (Section 3.5.2), both RT and UV are good choices. RT generally works better in the comparisons shown here, while UV can become more competitive when the differentiated signal is tied to high-frequency texture at fixed surface coordinates. Recursive parameterization couples downstream vertices and is not recommended in general, especially when using PRB.
- **Discontinuities:** When boundary derivatives are taken into account in the next section, detached sampling remains a safe default, while attached sampling typically creates more boundary terms and requires careful handling to avoid bias. RT keeps camera rays fixed under geometry changes, which allows direct use of non-smooth reconstruction filters like box filters without special handling; UV often has lower variance in estimating the boundary derivative, mostly because it avoids estimating radiance contributions of the foreground on a visibility boundary.

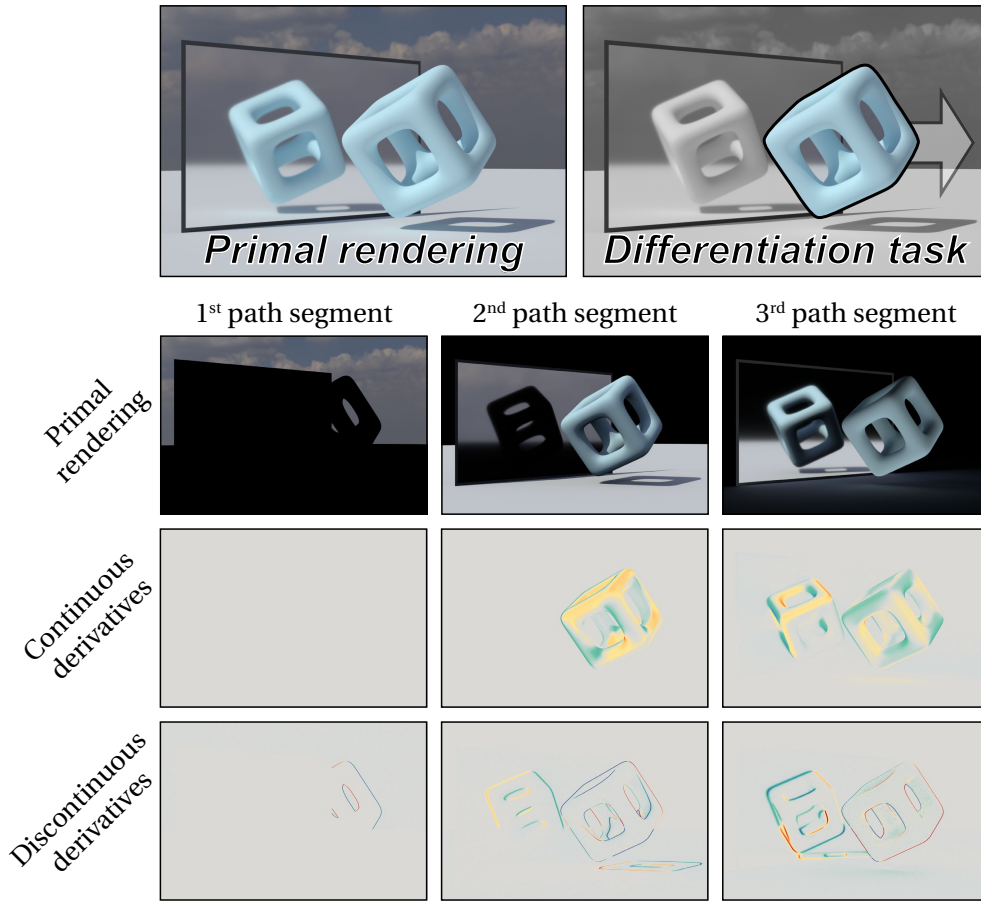


Figure 3.18: **Image and derivative decomposition.** Rendering derivatives inherit the same decomposition into visible emitters, direct illumination, and indirect transport as the primal image, but geometry differentiation introduces a further split into continuous and visibility-induced terms. The continuous part can be handled using automatic differentiation or adjoint methods [28, 29]. The visibility term is the focus of this section.

3.6 Visibility derivatives in physically based rendering

The previous sections deliberately stayed in the fixed-visibility regime. They showed how to compute derivatives of smooth transport factors efficiently (Section 3.3), and how to account for the continuous motion of path vertices when geometry changes but occlusion does not (Section 3.5.2). What remains is the genuinely discontinuous part of geometry differentiation, as we saw intuitively in Figure 3.3.

Figure 3.18 gives a useful high-level picture. For geometry differentiation, the total derivative naturally splits into a *continuous* term, which can be handled using the tools developed earlier in this chapter, and a *boundary* term caused by visibility changes.

The discussion proceeds in the order in which the problem is easiest to understand. We begin with a minimal primary-visibility example, lift the same mechanism to a reflectance integral,

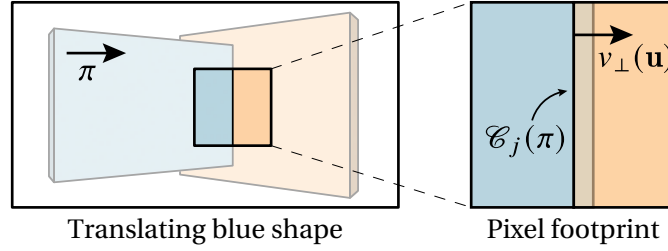


Figure 3.19: **A simple visibility discontinuity.** A pinhole camera observes two emissive objects: a blue object in front and a yellow object behind it. The parameter π translates the front object horizontally. As π changes, the projected visibility boundary sweeps across the pixel footprint, so a thin set of rays changes its first visible surface from yellow to blue. This changes the pixel value even though the pathwise derivative of radiance is zero for every fixed primary ray.

and then to recursive light transport. This leads naturally to Reynolds' transport theorem [37], which makes the missing term explicit. Once that term is identified, the remaining question is numerical: how should it be estimated in a practical renderer?

3.6.1 A simple example

The visibility problem is easiest to see in a scene with almost no other rendering structure. Figure 3.19 shows a pinhole camera viewing two emissive objects: a blue object in front and a yellow object behind it. The rear object is visible only where it is not occluded by the front one, and a scalar parameter π translates the front object horizontally. There are no BSDF interactions and no indirect bounces. Each camera ray contributes after a single segment, determined only by the first emissive object it hits.

Let $\mathbf{u}$ denote a point in the footprint $\mathcal{P}_j$ of pixel j on the sensor, and let $L(\mathbf{u}, \pi)$ be the radiance carried by the corresponding primary ray. The pixel value is

$$I_j(\pi) = \int_{\mathcal{P}_j} L(\mathbf{u}, \pi) \, d\mathbf{u}. \quad (3.49)$$

For this scene,

$$L(\mathbf{u}, \pi) = \begin{cases} L_{\text{blue}}, & \text{if the ray first hits the front object,} \\ L_{\text{yellow}}, & \text{if the ray first hits the rear object,} \\ 0, & \text{otherwise.} \end{cases} \quad (3.50)$$

As π varies, a thin strip of rays switches from yellow to blue, or vice versa, so the pixel value changes. From the viewpoint of automatic differentiation, $L(\mathbf{u}, \pi)$ is computed by a branch: either the ray first hits the foreground and returns L_{blue} , or it does not and returns L_{yellow} . For every $\mathbf{u}$, this branch decision is locally constant under an infinitesimal perturbation, so the mechanically differentiated, pathwise quantity $\partial_\pi L(\mathbf{u}, \pi)$ evaluates to zero.

Chapter 3. Differentiable PBR

This is the basic visibility paradox: the pixel value is well defined and varies with π , but its derivative cannot be recovered from the pathwise derivatives of already-existing light paths.

To make this explicit, partition the pixel footprint into visible regions $\mathcal{R}_{\text{blue}}(\pi)$ and $\mathcal{R}_{\text{yellow}}(\pi)$. Then Eq. (3.49) becomes

$$I_j(\pi) = \int_{\mathcal{R}_{\text{blue}}(\pi)} L_{\text{blue}} \, d\mathbf{u} + \int_{\mathcal{R}_{\text{yellow}}(\pi)} L_{\text{yellow}} \, d\mathbf{u}. \quad (3.51)$$

Differentiating this moving-domain integral is a two-dimensional instance of the same moving-boundary calculus formalized by Reynolds' transport theorem below. It yields a line integral rather than another two-dimensional integral:

$$\frac{\partial I_j}{\partial \pi} = \int_{\mathcal{C}_j(\pi)} \underbrace{v_{\perp}(\mathbf{u})}_{\text{normal speed}} \underbrace{(L_{\text{blue}} - L_{\text{yellow}})}_{\text{radiance jump}} \, d\ell(\mathbf{u}), \quad (3.52)$$

Here $\mathcal{C}_j(\pi)$ is the projected visibility contour within the pixel footprint and $d\ell$ is arc length along it. The factor $v_{\perp}(\mathbf{u})$ is the contour's *normal* speed induced by the scene motion under the perturbation of π ; tangential motion does not change the domain to first order. The integrand is therefore normal speed times the radiance jump $(L_{\text{blue}} - L_{\text{yellow}})$.

This toy example already contains the essential phenomenon. In physically based rendering, the moving boundary need not lie in the pixel domain: it may instead lie on the hemisphere of incident directions, or more generally in the full space of light-transport paths.

From primary visibility to illumination integrals The same mechanism appears one level deeper inside a standard reflectance integral. Consider a surface point $\mathbf{x}_a$ and, for simplicity, assume Lambertian diffuse reflection with BRDF value $\rho_d(\mathbf{x}_a) := \rho(\mathbf{x}_a)/\pi$:

$$L_o(\mathbf{x}_a, \pi) = \rho_d(\mathbf{x}_a) \int_{\mathcal{H}^2(\mathbf{x}_a)} L_i(\mathbf{x}_a, \boldsymbol{\omega}, \pi) \, d\boldsymbol{\omega}^{\perp}. \quad (3.53)$$

When π moves the occluder, the set of visible incident directions changes, and silhouette curves sweep across the hemisphere seen from $\mathbf{x}_a$. Compared to the first example, it is as if we have a pinhole camera at $\mathbf{x}_a$ looking at the hemisphere of incident directions, and the visibility boundary is now a curve on that hemisphere rather than on the sensor.

A direct-lighting configuration makes this especially concrete. As geometry moves, the portion of emitter area visible from $\mathbf{x}_a$ changes. Even an infinitesimal perturbation can shift the visibility boundary on the hemisphere and reveal a different portion of incident radiance.

If we partition the hemisphere into visible regions whose boundaries are the silhouette curves,

then differentiating Equation (3.53) gives

$$\partial_\pi L_o(\mathbf{x}_a, \pi) = \rho_d(\mathbf{x}_a) \int_{\mathcal{H}^2(\mathbf{x}_a)} \partial_\pi L_i(\mathbf{x}_a, \boldsymbol{\omega}, \pi) d\boldsymbol{\omega}^\perp + \rho_d(\mathbf{x}_a) \int_{\mathcal{B}(\mathbf{x}_a, \pi)} v_\perp(\boldsymbol{\omega}) \Delta L_i(\mathbf{x}_a, \boldsymbol{\omega}, \pi) d\ell(\boldsymbol{\omega}), \quad (3.54)$$

where $\mathcal{B}(\mathbf{x}_a, \pi)$ denotes the set of visibility boundaries on the hemisphere, ΔL_i is the jump in incident radiance across that boundary, and $v_\perp$ is the normal speed of the boundary in directional space.

3.6.2 Reynolds' transport theorem

The previous examples are instances of a general calculus fact. We now make that fact more explicit.

Suppose that a parameter-dependent domain $D(\pi)$ carries an integral

$$I(\pi) = \int_{D(\pi)} f(\mathbf{y}, \pi) d\mathbf{y}. \quad (3.55)$$

If the domain were fixed and f were smooth, we could differentiate under the integral sign. Visibility does not fit that setting: the integrand is only piecewise smooth, and the pieces are separated by boundaries that move with π .

The correct higher-dimensional generalization of Leibniz's rule is Reynolds' transport theorem [37]:

$$\frac{d}{d\pi} \int_{D(\pi)} f(\mathbf{y}, \pi) d\mathbf{y} = \int_{D(\pi)} \partial_\pi f(\mathbf{y}, \pi) d\mathbf{y} + \int_{\partial D(\pi)} f(\mathbf{y}, \pi) v_\perp(\mathbf{y}) d\sigma(\mathbf{y}), \quad (3.56)$$

where $v_\perp$ is the normal velocity of the moving boundary and $d\sigma$ is the induced boundary measure.

For piecewise-smooth rendering integrands, it is often more convenient to apply the theorem to each smooth region separately. If we decompose $D(\pi)$ into smooth regions $D_r(\pi)$ with integrands f_r , the boundary contribution is the sum of one-sided terms

$$\frac{d}{d\pi} \int_{D(\pi)} f(\mathbf{y}, \pi) d\mathbf{y} = \int_{D(\pi)} \partial_\pi f(\mathbf{y}, \pi) d\mathbf{y} + \sum_r \int_{\partial D_r(\pi)} f_r(\mathbf{y}, \pi) v_\perp(\mathbf{y}) d\sigma(\mathbf{y}), \quad (3.57)$$

where each interface is counted from both sides. For example, if a moving silhouette separates a foreground region from a background region, one side contributes the foreground radiance with the foreground normal orientation, while the other contributes the background radiance with the opposite orientation. When the two sides share the same boundary speed, these two contributions collapse to the familiar foreground-minus-background jump. In some such cases, the two one-sided terms can be further combined into a single jump term, as in the

previous examples:

$$\frac{d}{d\pi} \int_{D(\pi)} f(\mathbf{y}, \pi) d\mathbf{y} = \int_{D(\pi)} \partial_\pi f(\mathbf{y}, \pi) d\mathbf{y} + \int_{\Gamma(\pi)} v_\perp(\mathbf{y}) \Delta f(\mathbf{y}, \pi) d\sigma(\mathbf{y}), \quad (3.58)$$

where $\Gamma(\pi)$ denotes the moving discontinuity set and Δf is the jump across it. Equation (3.58) therefore assumes that both sides share the same normal boundary speed (up to orientation); if the two one-sided speeds differ, one should use Equation (3.57).

Equation (3.56) is the mathematical core of the visibility problem. Pathwise differentiation at a fixed sample $\mathbf{y}$ sees only the interior term $\partial_\pi f$. The second term is missed because it lives on a lower-dimensional set. In distributional language, the derivative of the visibility indicator contains a Dirac delta supported on visibility boundaries, which ordinary pathwise differentiation does not sample [38].

3.6.3 Recursive transport integrals

The direct-lighting example still hides the main complication of physically based rendering: in path tracing, incident radiance is itself defined *recursively*. At a surface vertex $\mathbf{x}_k$, suppressing participating media and many implementation details, we may write

$$\begin{aligned} L_o(\mathbf{x}_k, \boldsymbol{\omega}_o, \pi) &= \underbrace{L_e(\mathbf{x}_k, \boldsymbol{\omega}_o, \pi)}_{\text{emission}} \\ &+ \int_{\mathcal{H}^2(\mathbf{x}_k)} \underbrace{f_s(\mathbf{x}_k, \boldsymbol{\omega}, \boldsymbol{\omega}_o, \pi)}_{\text{scattering}} \underbrace{L_i(\mathbf{x}_k, \boldsymbol{\omega}, \pi)}_{\text{downstream transport}} d\boldsymbol{\omega}^\perp. \end{aligned} \quad (3.59)$$

Here the incident radiance field $L_i(\mathbf{x}_k, \boldsymbol{\omega}, \pi)$ is only piecewise smooth in direction: as geometry moves, the identity of the first visible surface seen along direction $\boldsymbol{\omega}$ can change discontinuously. When a geometry parameter changes, the hemisphere therefore splits into smooth directional regions $\mathcal{D}_{k,r}(\pi)$ separated by moving visibility boundaries, and the same Reynolds analysis applies:

$$\begin{aligned} \partial_\pi L_o(\mathbf{x}_k, \boldsymbol{\omega}_o, \pi) &= \partial_\pi L_e(\mathbf{x}_k, \boldsymbol{\omega}_o, \pi) \\ &+ \sum_r \int_{\mathcal{D}_{k,r}(\pi)} \partial_\pi f_{k,r}(\boldsymbol{\omega}, \pi) d\boldsymbol{\omega}^\perp \\ &+ \sum_r \int_{\partial \mathcal{D}_{k,r}(\pi)} v_\perp(\boldsymbol{\omega}) f_{k,r}(\boldsymbol{\omega}, \pi) d\ell(\boldsymbol{\omega}), \end{aligned} \quad (3.60)$$

where

$$f_{k,r}(\boldsymbol{\omega}, \pi) := f_s(\mathbf{x}_k, \boldsymbol{\omega}, \boldsymbol{\omega}_o, \pi) L_{i,r}(\mathbf{x}_k, \boldsymbol{\omega}, \pi) \quad \text{on side } r, \quad (3.61)$$

with $L_{i,r}$ denoting the restriction of incident radiance to that smooth side.

This is the recursive analogue of Equation (3.54). It suggests that the boundary term appears

in every transport integral where visibility is queried, not only at the sensor and not only in direct lighting. Furthermore, each one-sided boundary contribution is no longer a simple foreground/background color. It can contain arbitrary downstream transport: direct emission, shadows, glossy reflection, refraction, or many later bounces.

A direct consequence is that, in a full path tracer, *any* non-concave part of the geometry can create visibility boundaries at some bounce. We need to handle the global transport of visibility-induced discontinuous derivatives, so the problem cannot be solved by pre-identifying a small set of potential silhouette curves.

This is the main conceptual difference between visibility in rasterization-like settings and visibility in physically based rendering. The discontinuity is not merely observed by the camera. It is observed *through transport*.

A path-space view. The same picture becomes even cleaner in path space. Differentiating the path integral leads to a corresponding decomposition of the pixel derivative into an interior term over ordinary paths and a boundary term over a lower-dimensional set of *boundary paths* [27]. Schematically,

$$\frac{\partial I_j}{\partial \pi} = \underbrace{\int_{\hat{\Omega}} \partial_{\pi} \hat{f}_j(\tilde{\mathbf{p}}) \, d\mu(\tilde{\mathbf{p}})}_{\text{interior / continuous}} + \underbrace{\int_{\partial\hat{\Omega}(\pi)} \hat{g}_j(\tilde{\mathbf{p}}) \, v_{\perp}(\tilde{\mathbf{p}}) \, d\mu^{\partial}(\tilde{\mathbf{p}})}_{\text{boundary / visibility}}. \quad (3.62)$$

The boundary paths are the path-space analogue of the moving contours in Figure 3.19 and of the moving visibility boundaries on the incident hemisphere described above. We leave the details to the original paper, but the main takeaway is that the boundary term is always a lower-dimensional integral over paths that lie on the boundary of a discontinuity. The magnitude of the contribution is proportional to the normal speed of that boundary in path space.

3.6.4 Boundary motion

One point that the toy examples have deliberately suppressed is the precise meaning of the motion term. In Equations (3.52), (3.54), and (3.62), the quantity $v_{\perp}$ is *not*, in general, the world-space velocity of a moving object. Rather, it is the normal speed with which the discontinuity moves in the domain of integration: the pixel footprint, the incident-direction hemisphere, or path space. Geometry motion affects this speed, but only through the way it causes the corresponding visibility boundary to move in that domain.

Motion depends on path parameterization. In the primary-visibility example of Figure 3.19, the integration domain is the pixel footprint and the primary rays are fixed by the camera. Under that particular parameterization, the relevant boundary speed is simply the projected

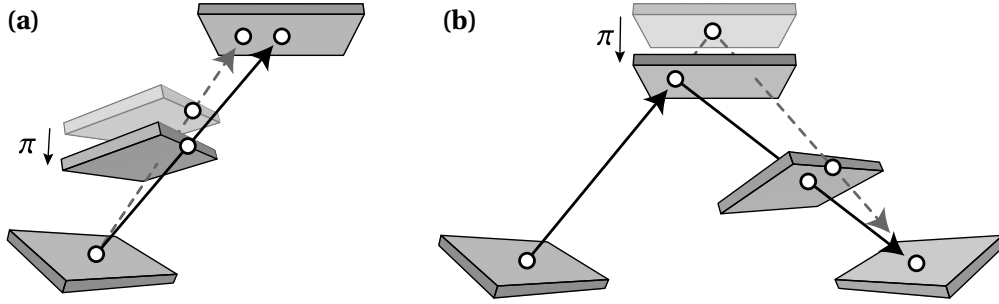


Figure 3.20: **Motion of a visibility boundary.** A visibility boundary can move because (a) the occluder itself sweeps through the path, or (b) the chosen path parameterization moves the path relative to a static occluder.

motion of the occluder in image space. This is intuitive, but it should not be mistaken for a general rule. Once we move to full path tracing, the coordinates used to represent a path become a design choice, and different path parameterizations induce different motions of the same visibility event in path space.

This matters especially for the UV/material-form parameterization of Section 3.5.2. There, a vertex on the occluder follows fixed surface coordinates rather than a detached ray. On the foreground side of a visibility event, the sampled point therefore moves with the occluding surface and does not cross the discontinuity in its own coordinates; the corresponding one-sided normal speed can go to zero even when the occluder has nonzero world-space velocity. On the background side, however, the same perturbation still changes whether the path remains visible, so the associated one-sided speed is generally nonzero. The two sides therefore need not share a common boundary speed, and the jump form in Equation (3.58) is no longer guaranteed. In such cases, one should retain the piecewise form in Equation (3.57). This asymmetry is one reason why the background of a visibility event is often the harder part of the discontinuity problem.

Motion has two sources. The toy example also suggests a second simplification: it makes boundary motion look as if it were caused only by an occluder physically sweeping across an otherwise valid ray. That is only the most direct mechanism, illustrated in Figure 3.20(a). There is also an *indirect* mechanism: the chosen path parameterization can move the ray segment itself. As shown in Figure 3.20(b), even a static occluder can create a moving visibility boundary when another perturbed interaction changes the origin or direction of the segment. For example, moving a reflector can rotate a shadow ray so that it begins to graze or intersect a previously irrelevant blocker.

This indirect boundary motion is easy to overlook because no object appears to “cut through” the segment in world space. Nevertheless, it is a genuine derivative contribution and can matter in scenes with strong self-shadowing or tightly coupled indirect transport. Computing

this relative motion again depends on the parameterization: recursive parameterizations can propagate motion through long suffixes of the path, making them hard to handle, whereas RT localizes the effect to the currently differentiated vertex neighborhood. The UV parameterization also propagates an interaction point's velocity only to neighboring vertices, but unlike RT, it does so in both directions, along the camera side and the light side, which often leads to higher variance and can create additional issues on the camera ray.

Fortunately, for the non-recursive parameterizations of practical interest, the motion factor needed by boundary estimators remains local to the tangential segment. One does not need a global description of the entire perturbed path, only the local normal speed with which the visibility event moves under the chosen path motion. For the remainder of this section, it is therefore sufficient to keep the simple picture of an occluder sweeping across the integration domain, while remembering that the actual value of $v_\perp$ is parameterization-dependent and can have another source.

3.6.5 Boundary sampling

Once the missing term has been written out as an integral that lives on a lower-dimensional domain, the most direct solution is to estimate it explicitly. The inconvenient alternative would be to keep sampling ordinary interior paths and hope to capture the discontinuity through pathwise derivatives. This fails because a random interior sample lands exactly on a visibility boundary with probability zero.

The main idea is simple. Instead of relying on ordinary path samples, which hit visibility events with probability zero, we generate samples directly on the lower-dimensional domain where the visibility derivative lives. In the simplest direct-light case, this means sampling silhouette points or tangential shadow rays. In full path tracing, it means sampling *boundary paths*.

A convenient way to write such methods is

$$\partial_\pi I_j^{\text{vis}} = \int_{\mathcal{S}_b} G_j(\mathbf{z}) \, d\mu_b(\mathbf{z}), \quad (3.63)$$

where $\mathcal{S}_b$ is a lower-dimensional boundary domain, $\mathbf{z}$ is a boundary sample, and G_j is the corresponding contribution. A Monte Carlo estimator is then simply

$$\widehat{\partial_\pi I_j^{\text{vis}}} = \frac{G_j(\mathbf{z})}{p_b(\mathbf{z})}, \quad \mathbf{z} \sim p_b. \quad (3.64)$$

All boundary-sampling methods share this structure. They differ in how the boundary sample $\mathbf{z}$ is parameterized and how the proposal p_b is chosen.

This becomes especially concrete in a local parameterization of tangential segments. For a smooth surface patch, a boundary sample can be described by a boundary point $\mathbf{x}_b$ together

with a parameter q encoding a tangent direction through that point. Ignoring technical details, the resulting local boundary contribution has the schematic form

$$\partial_\pi I_j^{\text{vis}} = \int \underbrace{L_i(\mathbf{x}_b, q)}_{\text{incident radiance}} \underbrace{W_i(\mathbf{x}_b, q)}_{\text{incident importance}} \underbrace{\kappa(\mathbf{x}_b, q)}_{\text{foreshortening}} \underbrace{(\partial_\pi \mathbf{x}_b \cdot \mathbf{n}_b)}_{\text{boundary motion}} d\mu_b(\mathbf{x}_b, q). \quad (3.65)$$

Here, L_i is the radiance transported to the tangential segment from the light side, W_i is the sensor-side importance reaching the same segment, κ is a foreshortening factor (similar to the $\cos\theta$ term in the rendering equation), and $(\partial_\pi \mathbf{x}_b \cdot \mathbf{n}_b)$ is the normal speed with which the boundary moves. This local form is useful because it turns the visibility derivative into an ordinary product of transport terms and a local motion factor.

The first physically based method in this category is the edge-sampling approach [38]. For primary visibility, the relevant boundaries are silhouette edges projected onto the image. For indirect visibility, the same idea is applied at arbitrary scene points: the algorithm samples mesh edges that are silhouettes relative to the point being shaded, and evaluates the jump in radiance across those boundaries. This was a foundational result because it showed that visibility derivatives in physically based rendering can be estimated without smoothing the forward problem.

The path-space viewpoint of Zhang et al. [27] reframes the same idea in a more transport-centric way. Instead of thinking of silhouettes relative to one scene point at a time, it treats visibility events as full boundary paths. A boundary path contains one special tangent segment and otherwise looks like an ordinary light-transport path; source-side and sensor-side subpaths can then be generated using standard path-sampling techniques. This is a natural match to physically based rendering because it places the visibility event directly in path space, where all higher-order transport effects already live.

The practical challenge is importance sampling. In complicated scenes, the integrand in Equation (3.63) is often highly peaked and sparse. Only a small fraction of boundary samples carry most of the contribution. This is exactly the difficulty emphasized by path-space methods and by the projective-sampling viewpoint: the formula is conceptually clean, but the boundary domain can be extremely hard to sample well.

Later work therefore focused on better proposals for this lower-dimensional domain. Some methods guide sampling directly in boundary-path space using specialized data structures [39]. Projective sampling, which will be the focus of the next chapter, follows a complementary intuition: ordinary primal samples already reveal where transport is important. Rather than discovering useful tangential configurations from scratch, it starts from primal ray segments and projects them onto nearby boundary samples, turning forward transport samples into proposals for the visibility term.

3.6.6 Boundary derivatives as differential transport

The boundary term also admits a very useful transport interpretation.

Section 3.4 explained that continuous derivatives can be viewed as a new transported quantity: differential radiance. It propagates like ordinary radiance, but is emitted by parameter-dependent sources such as changes in light emission or scattering.

Visibility adds one more source. Whenever a transport segment becomes “tangential” to geometry, the transport contribution can change suddenly. Equivalently, one may say that geometry emits differential radiance along tangential directions. This emitted quantity is highly singular: it is not supported on all directions, but only on those directions that form visibility transitions.

This point of view is particularly intuitive in the local formula (3.65). The terms L_i and W_i are ordinary transport quantities arriving at the tangential segment from the light side and the sensor side. The remaining factors are local: a foreshortening factor and the normal motion of the boundary. In other words, the tangential segment behaves like a tiny derivative emitter. Its strength is set by local geometry and motion, and the emitted differential radiance is weighted by foreshortened incident radiance before being transported through the scene in the usual way.

This gives a clean interpretation of several existing methods.

Edge sampling corresponds to explicitly sampling these derivative emitters. At each shading point, silhouette sampling is therefore analogous to emitter sampling, except that the “emitters” are now the tangential visibility silhouettes. Reparameterization here refers to visibility-aware warping of the integration variables: ordinary path samples are moved so that they follow nearby visibility events. From the differential-transport perspective, this approximates the effect of many highly directional derivative emitters by perturbing the ordinary path samples that happen to fall near them. Boundary sampling can then be understood as a kind of *light tracing* from these directional emitters, although the sampling domain is exceptionally large—comprising the full continuous set of tangential segments rather than a small discrete set of light sources. In practice, these methods are often implemented exactly like a light-tracing pass.

Finally, the projective-sampling viewpoint discussed in the next chapter also fits naturally into this story as an importance-sampling strategy tailored for the domain of “directional emitters.” If tangential segments are the true derivative emitters, then primal rendering already provides valuable signals about where important emitters are likely to be located. Projecting primal segments onto nearby tangential ones is thus a way of repurposing ordinary transport samples into high-quality proposals for these derivative sources.

This interpretation is helpful because it places visibility on an equal footing with the other derivative sources introduced earlier in the chapter. The visibility problem is not an exception

to derivative transport. It is another source term, except that this one is supported on a lower-dimensional set.

3.6.7 Boundary integral reparameterization

Boundary sampling attacks the visibility term where it lives: on the moving boundary itself. Reparameterization begins from a different intuition.

The core problem with naïve differentiation is that the visibility boundary moves while our Monte Carlo samples stay fixed. A natural question is therefore: what if the samples moved *with* the boundary?

This is the basic idea behind reparameterization. Instead of tracing the same ray or path sample before and after differentiation, we deform the sampling coordinates so that the sample follows the motion of nearby silhouettes. If that deformation is chosen well, the discontinuity becomes stationary in the transformed coordinates, and pathwise differentiation becomes possible again.

A simple mathematical example makes this idea precise. Consider the integral

$$I(\pi) = \int_0^1 H(x - \pi) g(x) dx = \int_\pi^1 g(x) dx, \quad (3.66)$$

where H is the Heaviside step function and g is a smooth function. The discontinuity is located at $x = \pi$, so the visible region moves as π changes.

If we differentiate Equation (3.66) directly, we obtain

$$\frac{dI}{d\pi} = -g(\pi). \quad (3.67)$$

This derivative comes entirely from the motion of the boundary. A pathwise derivative at a fixed sample x would miss it, because $H(x - \pi)$ is constant for almost every fixed x .

Now reparameterize the visible interval using

$$x = T(u, \pi) := \pi + (1 - \pi)u, \quad u \in [0, 1]. \quad (3.68)$$

Then

$$I(\pi) = (1 - \pi) \int_0^1 g(\pi + (1 - \pi)u) du. \quad (3.69)$$

The moving boundary has disappeared from the integration limits: it is now frozen at the fixed endpoint $u = 0$. Differentiating under the integral sign gives another valid expression for the same derivative. The discontinuous boundary term has not disappeared; it has been absorbed into the Jacobian and parameter dependence of the warp.

This toy example captures the essential idea in rendering. A moving shadow boundary on

the hemisphere, or more generally a moving visibility event in path space, plays the role of the discontinuity at $x = \pi$. Reparameterization attempts to construct a local map that moves samples together with that boundary, so that differentiation can proceed on a fixed reference domain.

Mathematically, the general rendering version is again just a change of variables. Consider a parameter-dependent integral

$$I(\pi) = \int_D f(\omega, \pi) d\omega, \quad (3.70)$$

where the discontinuity of f moves with π . Introduce a map $\omega = T(\mathbf{u}, \pi)$ from a reference domain U to D . Then

$$I(\pi) = \int_U f(T(\mathbf{u}, \pi), \pi) |\det J_T(\mathbf{u}, \pi)| d\mathbf{u}. \quad (3.71)$$

If T is chosen so that the discontinuity remains fixed in the $\mathbf{u}$ -domain, then the transformed integrand can be differentiated under the integral sign. Formally, for a discontinuity set $\Gamma(\pi)$ in the original domain, the desired condition is that $T^{-1}(\Gamma(\pi), \pi)$ is independent of π , at least to first order. The visibility derivative is no longer written as an explicit boundary integral; instead, it appears through the Jacobian and parameter dependence of the warp [40].

The appeal of this approach is immediate. It stays close to ordinary path tracing: we still sample an interior domain, and we do not need to build a separate boundary sampler. Moreover, the construction can be made largely independent of the underlying shape representation, since it only requires ray intersections and a way of estimating how nearby visibility changes.

This is exactly the strength of Loubet et al.'s method. The algorithm constructs local changes of variables on the fly from auxiliary rays, with the goal of estimating how the nearby boundary moves and then letting the sample follow that motion. In the language of the toy example above, the method tries to build, for each rendering sample, a local warp that keeps the relevant discontinuity as stationary as possible.

At the same time, this also explains the main weakness. The quality of the derivative estimate depends on the quality of the warp field. If the warp estimates the motion of nearby visibility events poorly, then the transformed integrand may still contain residual discontinuities or may inject unnecessary variance away from the true boundary. This is why reparameterization often spreads variance more globally than explicit boundary sampling, and why its numerical behavior can be sensitive even when the underlying idea is sound. Figure 3.21 illustrates both effects.

Bangaru et al.'s warped-area method clarifies this relationship in a particularly elegant way [41]. Starting from Reynolds' theorem, it applies the divergence theorem to convert the boundary integral into an equivalent area integral. The key object is a vector field whose normal component matches the motion of the boundary. Conceptually, this shows that boundary sampling and area sampling are not competing theories. They are two equivalent representations of the same derivative term. The main difference is numerical: one estimator samples the lower-

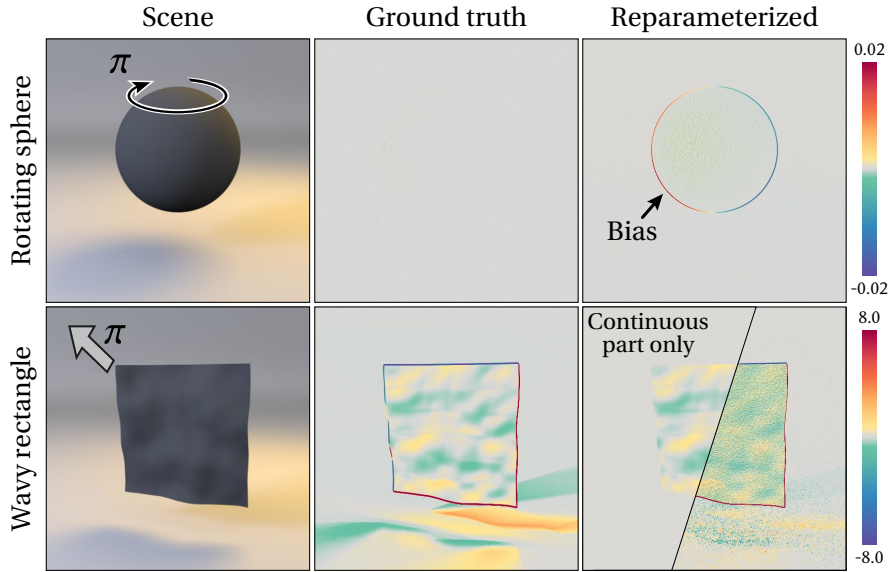


Figure 3.21: Reparameterizations [41, 40] inject variance and bias into the derivative computation. **(top)** This uniform rotating sphere should not produce a visible derivative signal. The reparameterization fails to recognize this and produces biased output. **(bottom)** The derivative of a translating wavy rectangle has interior and boundary components. A naïve Monte Carlo estimator misses the boundary derivative but produces an otherwise well-converged estimate of the interior part. The reparameterized version fixes the boundary at the cost of a global variance increase.

dimensional boundary directly, the other samples an interior domain carrying an equivalent flux.

This is why reparameterization deserves more than a brief survey mention. It changes how one thinks about the visibility problem. Instead of always treating visibility derivatives as singular terms that must be sampled on silhouettes, it shows that they can sometimes be absorbed into a carefully designed coordinate transform. That idea is both practically useful and conceptually illuminating.

3.6.8 Other approaches

The discussion above covers the two most influential strategies for handling visibility in general physically based rendering: explicit boundary estimation and reparameterization. A few additional directions are worth mentioning briefly. Figure 3.22 provides a visual summary.

Some methods exploit more restrictive analytic structure. When the forward integral admits a closed form, or when visibility can be handled analytically in a simplified rendering model, the derivative can be obtained directly without Monte Carlo boundary estimation [48]. These methods can be very effective in their intended setting, but they do not easily extend to general multiple-scattering light transport.

3.6 Visibility derivatives in physically based rendering

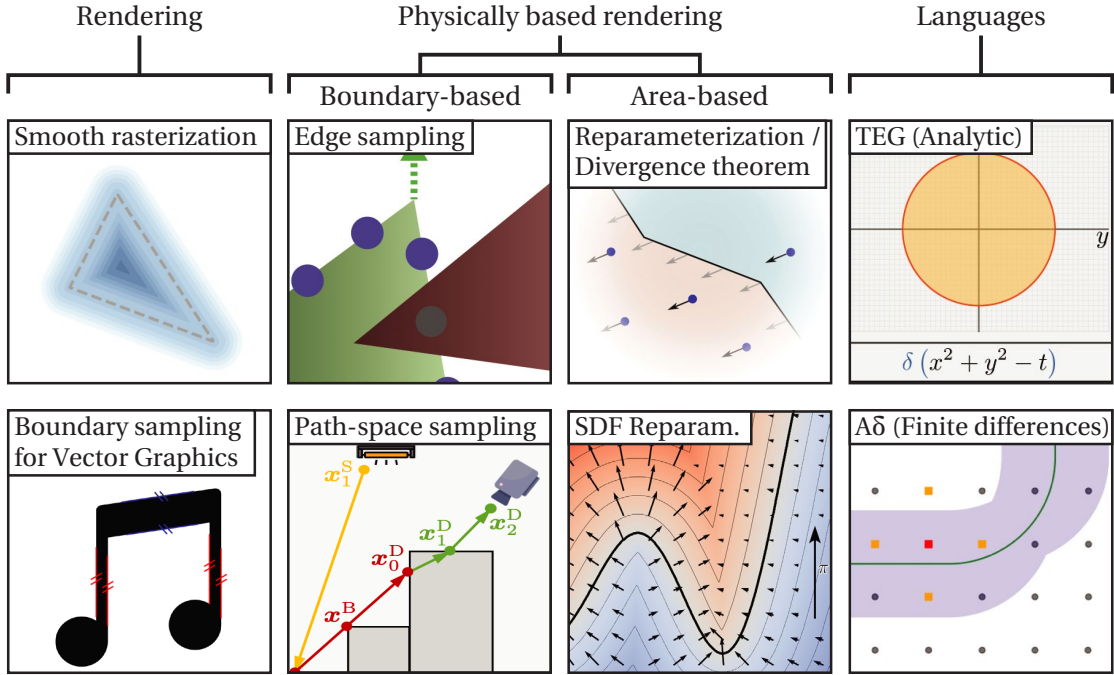


Figure 3.22: **A taxonomy of discontinuous derivative methods.** Visible discontinuities can be handled by smoothing boundaries [42] or explicitly sampling them [43]. The physically based setting features indirectly observed discontinuities. Previous boundary-based sampling methods used 6D Hough trees [38] or path space [27]. Alternatively, an area-based integral can be reparameterized to freeze discontinuities [40], which is equivalent to an application of the divergence theorem [41]. SDFs are particularly amenable to reparameterization [44, 45]. Recent differentiable programming languages differentiate discontinuous integrals using analytic methods [46] and finite differences [47].

Other work studies discontinuous integrals from the viewpoint of differentiable programming rather than rendering specifically. This has led to systematic rules and language-level abstractions for parametric discontinuities, including rendering-inspired examples and finite-difference-style distributional estimators [46, 47]. These ideas are conceptually related to visibility differentiation, although their main goal is broader than physically based rendering.

There are also methods specialized to particular geometry representations. Implicit surfaces and signed distance functions, for instance, admit reparameterizations or boundary constructions that differ from those used for triangle meshes [44, 45]. These should be understood as representation-specific refinements of the same two core ideas: sample the visibility term directly, or convert it into a smoother interior integral.

Finally, more recent work explores a more radical alternative. Instead of correcting derivatives under local surface evolution, one can change the derivative domain itself. Many-worlds derivatives extend differentiation from the current surface to a family of hypothetical surface patches in 3D space, avoiding the need to handle local visibility discontinuities in the same

form during the optimization loop [5]. This does not contradict the boundary-term picture developed in this section; it changes the optimization problem so that those boundary terms no longer have to be handled in the same local way.

3.6.9 Summary

The visibility derivative problem is best understood as a moving-boundary problem.

The simplest rendering example already contains the whole mechanism: a pixel observing two emissive objects changes because a projected visibility contour moves across the pixel footprint, even though the contribution of almost every fixed camera ray has zero pathwise derivative. The same effect reappears inside direct-light and recursive transport integrals, where the moving boundary lives on hemispheres of directions or, more generally, in path space.

Reynolds' transport theorem makes the missing term explicit: geometry derivatives consist of an interior contribution and a boundary contribution supported on visibility events. Path-space formulations reveal that this boundary term is itself a light-transport integral over boundary paths. In a local parameterization, the corresponding integrand can be understood as the product of light-side radiance, sensor-side importance, a local geometric factor, and the normal motion of a tangential visibility segment.

Existing methods then differ mainly in how they estimate or avoid this term. Boundary-sampling methods estimate it directly. Reparameterization and warped-area methods convert it into an equivalent interior integral. Newer approaches such as many-worlds change the derivative domain so that local visibility discontinuities no longer have to be handled in the same form.

This is the last major obstacle in computing correct derivatives of a physically based path tracer. The next chapter focuses on one particular way of estimating the boundary term efficiently: projective sampling.

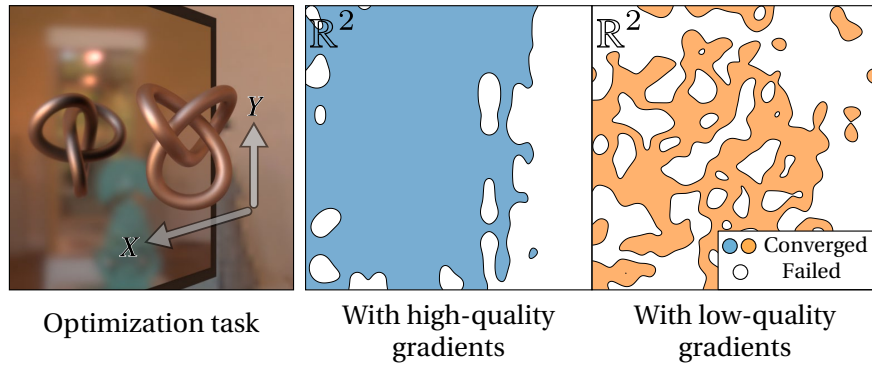


Figure 3.23: **Gradient quality and robustness.** Stochastic optimizers such as Adam can handle noisy gradients, but excessive variance can be detrimental. In this example, we infer the X/Y offset of the knot shape from a single reference view. The middle and right images depict the convergence status after initialization at the associated point in the parameter domain (white indicates a failure to converge to the known parameter value). The only difference between these experiments is the quality of the gradient.

3.7 Optimization bottlenecks beyond derivatives

Correcting visibility derivatives is necessary but not sufficient for robust geometry optimization. The derivative of the rendering process can be decomposed into continuous and discontinuous parts. Since this thesis focuses on geometry optimization, we refer to the continuous part as the *shading derivative* and the discontinuous part as the *boundary derivative*.

The shading derivative is defined everywhere on the surface and primarily acts through changes in surface normals, so it depends strongly on the lighting. By contrast, the boundary derivative is defined only on the surface’s visibility silhouette. These surface derivatives are sparse in two senses:

1. A single view observes only a limited subset of all possible silhouettes, and an update based on these derivatives is likely to introduce a kink only along that subset.
2. The derivatives are only defined on the surface, not throughout the entire 3D space. Although light transport is simulated in this larger space, only neighborhoods of the surface can use the computed gradients. Regions far from the initial guess receive gradients only after the surface extends to them.

This sparsity has two implications: optimization is slow because many iterations are required to deform the surface to the target shape, and a good initial guess is needed, especially when the target shape has complex topology or self-occlusions.

The next chapter builds on this background and focuses on projective sampling for visibility-induced derivatives. It revisits the local parameterization of boundary integrals and turns

Chapter 3. Differentiable PBR

primal rendering samples into boundary samples through projection, improving efficiency and reducing variance while preserving correctness.

The following chapter then addresses the sparsity bottleneck discussed above. Instead of evolving geometry using only local surface derivatives, it extends these derivatives into space using a many-worlds formulation based on a distribution of hypothetical surfaces.

4 Projective Sampling for Differentiable Rendering of Geometry

Chapter 3 separated geometry derivatives in physically based rendering into a smooth term and a visibility-induced boundary term. The smooth term can be handled with the differential transport machinery developed there. The boundary term is harder: it lives on a lower-dimensional domain of boundary light paths, so even unbiased estimators are difficult to sample efficiently. This chapter stays in the lower-right corner of the design space introduced earlier: explicit surfaces under physically based transport.

A natural question is whether the sampling structure of ordinary rendering can help with this derivative problem. Primal path tracing already uses carefully engineered proposals such as BSDF sampling, emitter sampling, and MIS to find important light-transport paths. Yet boundary-derivative methods traditionally treat discontinuous terms as a separate Monte Carlo problem with their own parameterizations. The central observation of this chapter is that these two tasks need not be disconnected:

“What is good for function values is good for their derivatives.”

— Griewank and Walther [35], rule #2 for automatic differentiation

In this spirit, important primal light paths are often close to the boundary paths that dominate visibility derivatives.

Projective sampling turns this observation into an estimator design principle. It starts from ordinary primal samples, projects nearby path segments onto silhouettes or other tangential configurations, and records the resulting boundary events. These projected samples are then summarized into a guiding distribution that can be sampled explicitly in a second stage. The guiding model is only approximate, but if it covers the important parts of the boundary sample space, the resulting estimator remains unbiased and substantially more efficient.

The chapter also revisits the underlying theory. It re-derives the local boundary formulation to expose a surprisingly simple expression for the boundary term. This echoes the thesis narrative by bringing visibility derivatives closer to the transport viewpoint of the previous chapter: the discontinuous term can be written in a local form that is close enough to ordinary transport to benefit from the same sampling intuition.

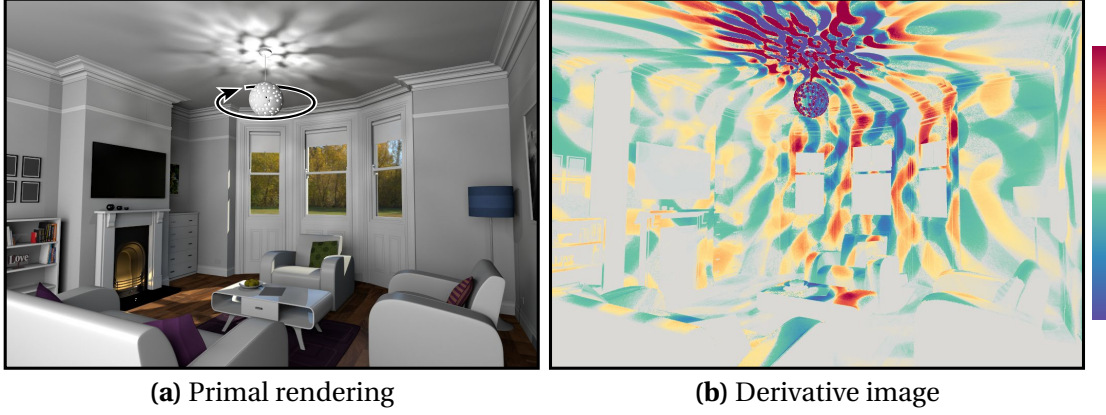


Figure 4.1: **Projective sampling.** The visibility function plays a crucial role in differentiable rendering of geometry. Parameter changes that influence visibility (e.g., a rotation of the light source in (a)) can generate a significant derivative contribution, shown in (b), that is difficult to sample. We project path segments generated by primal rendering algorithms (e.g., direct illumination sampling) onto nearby silhouettes and organize them into a uniform or adaptive guiding data structure to improve numerical integration of the challenging boundary term. Compared to prior work [27], uniform guiding reduces errors (RMSE) by an average factor of $8.1\times$, and adaptive guiding yields an additional $2.7\times$ improvement.

Unlike the later chapters, which relax either the geometry representation or the image-formation model, this chapter asks how far one can go by improving derivative estimation alone. Its answer is a modular framework that only requires a projection operator and a local parameterization for each shape class, which makes it applicable to triangle meshes, implicit surfaces, and fibers.

The remainder of the chapter follows this logic directly. It first re-derives the local boundary integral, then turns that formulation into a projection-and-guiding strategy for several shape classes, and finally evaluates the resulting estimator in both standalone derivative estimation and end-to-end inverse rendering.

4.1 A local boundary integral

Zhang et al.’s [27] path-space formulation of differentiable rendering decouples boundary effects from interior effects, enabling targeted computation of each part using specialized methods. Their equation (43) provides the starting point of our derivation.

This equation relates the change in pixel intensity I caused by visibility changes under a perturbation of a scene parameter π . Written with respect to the path segment $(\mathbf{x}_a, \mathbf{x}_c)$, it states

$$\frac{\partial I}{\partial \pi} = \int_{\mathcal{A}} \int_{\mathcal{B}(\mathbf{x}_a)} L_i(\mathbf{x}_a, \mathbf{x}_c) G(\mathbf{x}_a, \mathbf{x}_c) W_i(\mathbf{x}_c, \mathbf{x}_a) (\partial_{\pi} \mathbf{x}_c \cdot \mathbf{n}_c) dI(\mathbf{x}_c) dA(\mathbf{x}_a). \quad (4.1)$$

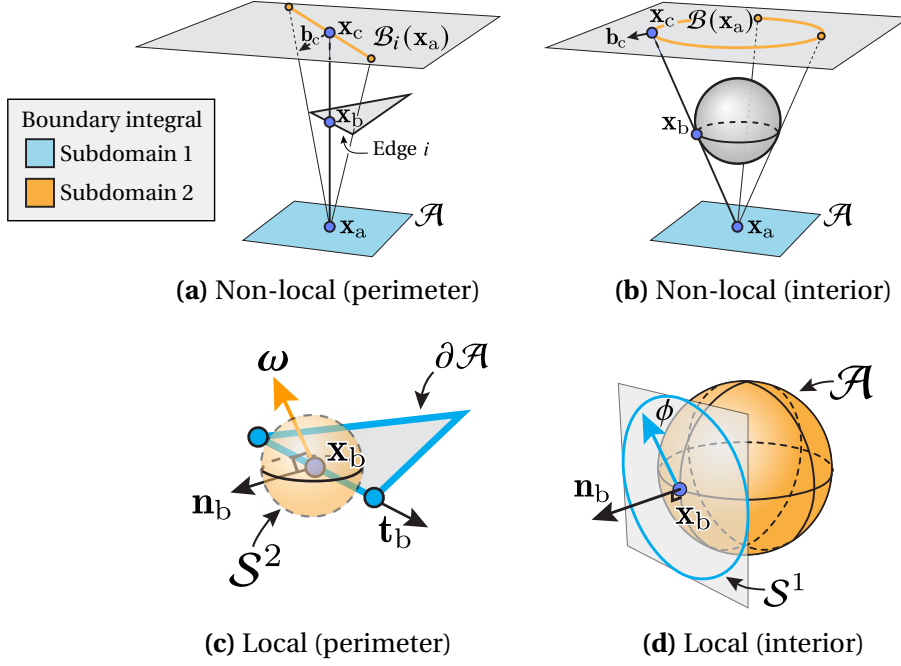


Figure 4.2: **Formulations and terms of the boundary integral.** Visibility-related derivatives arise from the perimeter (e.g., discrete edges of a triangle mesh) and the interior of shapes (e.g. the surface of an ellipsoid). Path-space methods compute an integral over tangential path segments to account for them. Decomposing the integration domain (blue and orange sets) reveals different formulations: (a) For the perimeter component, one can integrate over source points $\mathbf{x}_a \in \mathcal{A}$ and the “shadow” $\mathbf{x}_c \in \mathcal{B}_i(\mathbf{x}_a)$ cast by a discrete edge i . (b) This formulation also generalizes to the interior, but parameterizing and sampling the projected boundary $\mathcal{B}(\mathbf{x}_a)$ is difficult in general. (c) The local formulation instead evaluates a spherical integral at boundary points $\mathbf{x}_b \in \partial\mathcal{A}$ without explicit consideration of the neighboring vertices $\mathbf{x}_a$ and $\mathbf{x}_c$. (d) The interior can be handled analogously but requires a different partition into an integral over surface positions (orange) and tangential directions (blue). We propose a new local boundary integral that accounts for this component.

The integral is over segments $(\mathbf{x}_a, \mathbf{x}_c)$ that make contact with a surface boundary along the way. The domain $\mathcal{B}(\mathbf{x}_a)$ describes the “shadow” of this boundary and can be decomposed as $\mathcal{B}(\mathbf{x}_a) = \cup_i \mathcal{B}_i(\mathbf{x}_a)$ (one set $\mathcal{B}_i$ per edge) when the scene consists of discrete geometry (Figure 4.2a). The functions L_i and W_i refer to the incident radiance and importance and can be computed using existing primal estimators. The term G is the standard geometric term [49], and the inner product measures the perpendicular speed of the “shadow” at $\mathbf{x}_c$ (see Figure 4.2a).

In practice, it is simpler and far more efficient to build boundary segments “outward” starting from the tangential point $\mathbf{x}_b$, and Zhang et al. therefore also propose a *local* formulation of the perimeter contribution (Figure 4.2c).

However, their derivation [27, Appendix 1] leaves the final equation in a form that still con-

Chapter 4. Projective Sampling for Differentiable Rendering of Geometry

tains derivatives of a ray tracing operation evaluated with automatic differentiation. That representation is correct, but it obscures simplifications that are useful both conceptually and computationally.

This chapter sharpens the local viewpoint in two ways: it re-derives the perimeter formulation to expose its simpler reduced form, and it extends the same treatment to the interior of smooth shapes. Combining both sources of derivatives leads to the following complete expression, which provides the theoretical basis of the method.

$$\begin{aligned} \frac{\partial I}{\partial \pi} = & \int_{\partial \mathcal{A}} \int_{\mathcal{S}^2} L_d(\mathbf{x}_b, \boldsymbol{\omega}) W_i(\mathbf{x}_b, \boldsymbol{\omega}) \sin \phi (\partial_\pi \mathbf{x}_b \cdot \mathbf{n}_b) d\boldsymbol{\omega} dl(\mathbf{x}_b) \\ & + \int_{\mathcal{A}} \int_{\mathcal{S}^1} L_d(\mathbf{x}_b, \phi) W_i(\mathbf{x}_b, \phi) \kappa(\phi) (\partial_\pi \mathbf{x}_b \cdot \mathbf{n}_b) d\phi dA(\mathbf{x}_b). \end{aligned} \quad (4.2)$$

The derivation of this expression is somewhat technical; a detailed proof is given in Appendix A.1.

The term L_d denotes the radiance difference between foreground and background, $L_d(\mathbf{x}_b, \boldsymbol{\omega}) = L_o(\mathbf{x}_b, \boldsymbol{\omega}) - L_i(\mathbf{x}_b, -\boldsymbol{\omega})$, assuming the use of the RT path parameterization (Section 3.5.2). In the perimeter term (top integral), $\sin \phi$ refers to the angle between $\boldsymbol{\omega}$ and the boundary tangent $\mathbf{t}_b$ (Figure 4.2c). In the interior term, the angle $\phi \in \mathcal{S}^1$ parameterizes all relevant quantities over tangential directions at the surface position $\mathbf{x}_b$ (Figure 4.2d), and $\kappa(\phi)$ denotes the normal curvature. The following aspects are noteworthy:

1. Neither term involves the complex non-local domain $\mathcal{B}(\mathbf{x}_a)$.
2. In contrast to Equation 4.1, the geometric term G is absent in both integrals, which means that variance arising from this factor can be avoided in Monte Carlo methods. For instance, consider a rod with a 45-degree inclination casting a shadow from an overhead directional light onto a ground plane. Prior work placed more samples on the far end of the rod’s silhouette, which is undesirable because the contribution per unit arclength is uniform. This is not obvious when looking at Equation (4.1), and the reference implementation of Zhang et al. [27], for example, tabulates G within its guiding distribution even though it cancels in subsequent steps.
3. When the scene geometry is smooth and closed, $\partial \mathcal{A} = \emptyset$ removes the first term. Polygonal meshes do not have curved interiors ($\kappa = 0$) and hence do not require the second term.
4. The sine (perimeter) and curvature (interior) terms resemble the cosine factor in the rendering equation: the influence of an edge with tangent $\mathbf{t}_b$ in direction $\boldsymbol{\omega}$ tends to zero as $\mathbf{t}_b \rightarrow \boldsymbol{\omega}$, since the edge becomes invisible. Likewise, the curvature term accounts for foreshortening in the mapping between silhouette positions and scene positions from which the silhouette is observed.

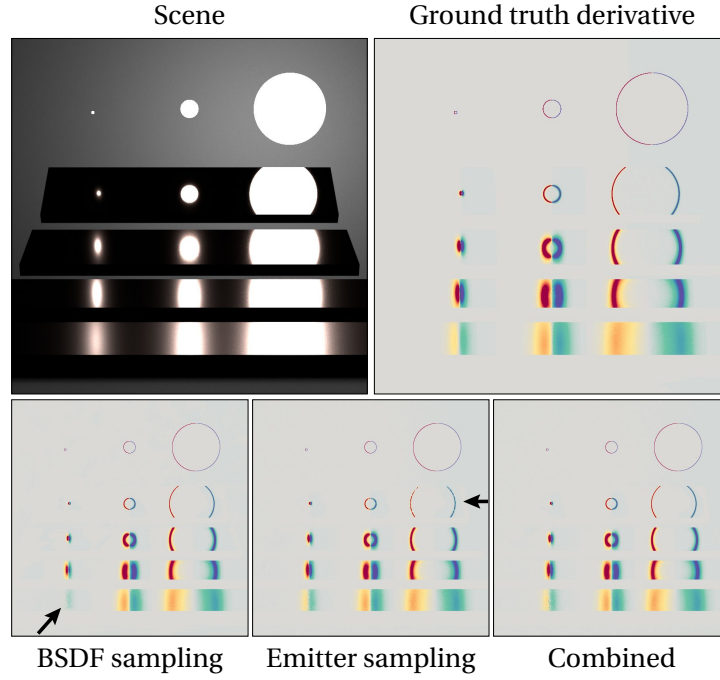


Figure 4.3: **Impact of the interior sampling distribution.** This experiment analyzes the derivative of a classic test scene by Veach with respect to a horizontal translation. Closer investigation reveals familiar failure modes: the projected BSDF sampling strategy fails to generate a sufficient number of silhouette samples to cover the rough reflection of the smallest sphere, while emitter sampling presents the opposite issue in the smooth reflection of the largest sphere. Analogous to multiple importance sampling [49] for primal rendering, it can be beneficial to project a mixture of both.

4.2 Method

With this background in place, we can now turn to the details of interior and guided sampling. We discuss the projection operation last, since it requires specialization to various geometric representations. This section also presents a number of results that examine the quality of computed gradients to motivate algorithmic design decisions. These are followed by more comprehensive end-to-end optimization results in Section 4.3.

Input sampling strategies. The method repurposes interior sampling strategies as specialized proposals for the boundary term. This raises a practical question: which interior distributions should be projected? In primal rendering, it is common to combine BSDF and emitter sampling via *multiple importance sampling* (MIS) [49]. Figure 4.3 shows that the same intuition carries over to the differential setting.

Embracing imperfection. The chapter preamble introduced a path-segment projection that maps an input segment $\mathbf{x}_a \rightarrow \mathbf{x}_b$ onto a nearby segment $\mathbf{x}_a \rightarrow \mathbf{x}'_b$ such that $\mathbf{x}'_b$ lies on a

silhouette observed by $\mathbf{x}_a$. At first glance, one might require this projection to be exact. In practice, that requirement is stricter than necessary.

Recall that the method no longer solves the boundary integral for a fixed position $\mathbf{x}_a$, because the density of projected samples would be too expensive to characterize. The projections only guide a later integration phase that accounts for all surface positions $\mathbf{x}_a$ simultaneously. It is therefore sufficient to find an approximate boundary segment $\mathbf{x}'_a \rightarrow \mathbf{x}'_b$ close to $\mathbf{x}_a \rightarrow \mathbf{x}_b$, even if the origin $\mathbf{x}_a$ shifts slightly as well. This flexibility is useful in two ways. First, the projection may fail; in that case, we snap $\mathbf{x}'_b$ to the closest local boundary (e.g. a triangle edge) and select the tangential direction closest to the original direction of $\mathbf{x}_a \rightarrow \mathbf{x}_b$. Second, when a projection requires root-finding, we terminate the iteration after a few steps instead of waiting for machine precision. The former step reduces variance, while the latter reduces the cost of constructing the guiding distribution.

4.2.1 Guiding distribution

Given a set of projected samples, the next major step is to condense this information into a representation that supports efficient sampling and density evaluation.

Grid-based guiding

The simplest option is a 3D density grid, which can be a good choice when the integrand exhibits sufficient smoothness. Zhang et al.’s [27] *path-space differentiable rendering* (PSDR) method likewise relies on a grid.

One notable advantage of grids is that the projection step can directly accumulate projected samples without having to store the samples themselves, which can enable high-quality statistics accumulation in memory-constrained environments. Conversely, a high-resolution 3D grid can be very memory-intensive and tends to become a bottleneck in complex scenes requiring a fine discretization to resolve sparse features.

Figure 4.4 presents a first set of results for grid-based guiding. Each pair of rows shows the gradient with respect to a horizontal translation in the first row, followed by error visualizations compared to a finite difference reference in the second row. We use two different color scales throughout this chapter: a standard red/gray/blue scale for the error images in the lower rows (with gray representing zero error), and a more varied color scale for the gradient images due to their large dynamic range. The $\mathcal{L}_1$ and $\mathcal{L}_2$ annotations denote the gradient MAE and RMSE values compared to the reference. The primal rendering is illustrative and uses a different viewpoint and lighting setup (Section 4.3.1 provides more details on the test scene setup).

All results in Figure 4.4 use a fixed grid with $10000 \times 100 \times 100$ voxels for (t, θ, ϕ) . The only differences are the number of accumulated samples and whether projective sampling was used. The second column, labeled “*uniform, 5x samples*”, shows the PSDR baseline using

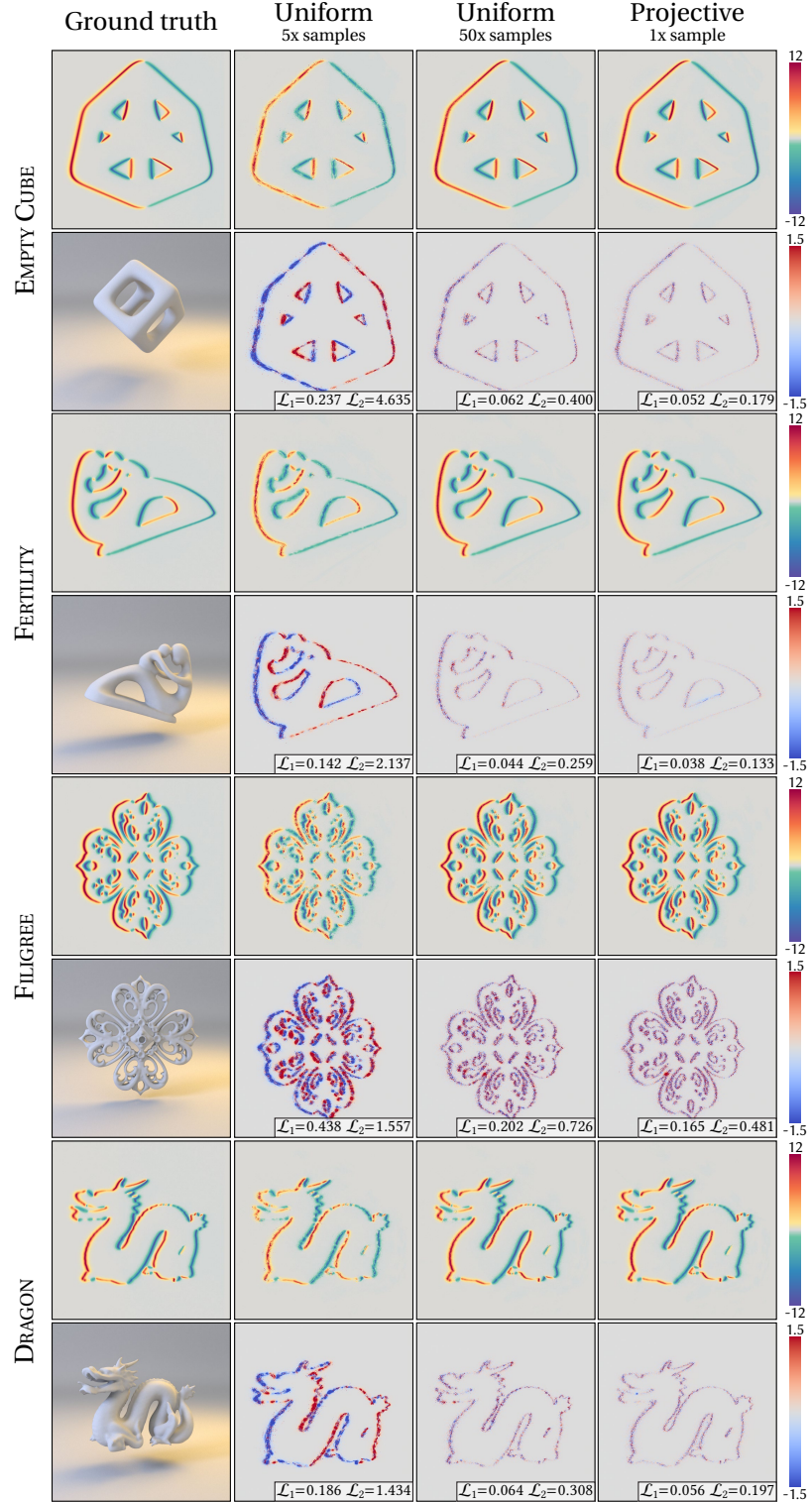


Figure 4.4: **Grid-based guiding.** Qualitative and numerical evaluation of grid-based guiding comparing uniform [27] and projective sampling on meshes of varying geometric complexity. Each row shows gradient images followed by a visualization of the associated error. See Table 4.1 for timings.

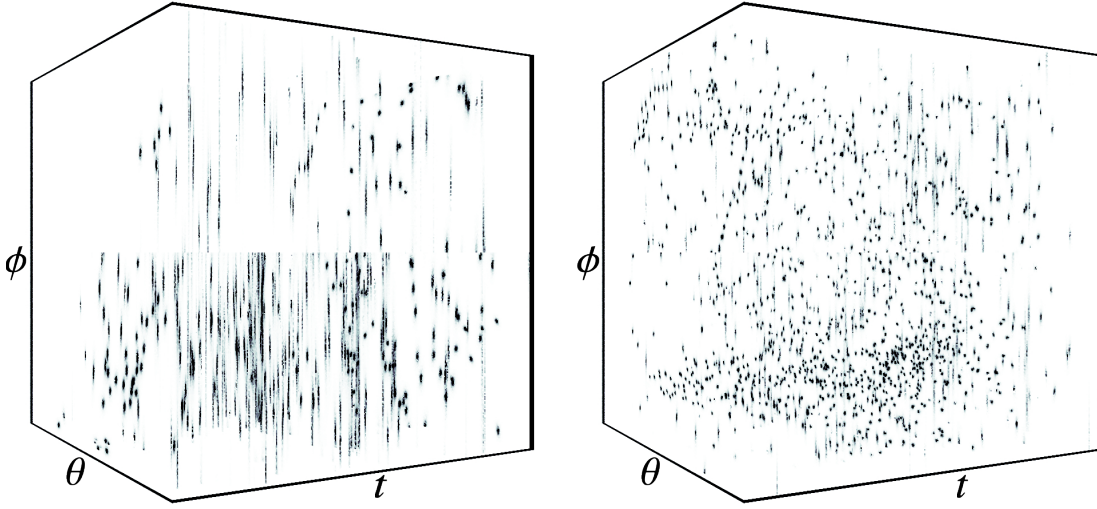


Figure 4.5: **Boundary sample space.** Visualization of the boundary integrand of the BUNNY and FILIGREE meshes from Figure 4.4 in *boundary sample space*. This 3D domain is parameterized by edge position $t \in \partial\mathcal{A}$ and a spherical direction $(\theta, \phi) \in \mathcal{S}^2$. Note the extremely sparse and high-frequency nature of this function (white corresponds to an integrated value of 0 along a ray through the 3D volume).

five samples per voxel. The last column employs our projection for triangle meshes and was generated in less time than the competing methods (see Table 4.1 for precise timings). We also use fewer samples to populate the grid given the added cost of the projection. The middle result shows the quality improvement that PSDR could obtain by using many more samples to detect fine features within voxels, though this multiplies the computation time by a factor of nearly $10\times$. These results show that projective sampling consistently outperforms PSDR by a factor of $2.5\text{--}4.5\times$ (MAE), which grows to a factor of up to $25\times$ in RMSE due to the presence of outliers.

Hierarchical guiding

Figure 4.5 visualizes the intricate structure of the integrand in *boundary sample space*. For polygonal meshes, this 3D space consists of a single parameter $t \in [0, 1]$, which parameterizes the set of mesh edges $\partial\mathcal{A}$, along with a direction (θ, ϕ) represented in spherical coordinates. Sparse features in this domain arise from complexities of the input scene, including:

1. BSDFs with narrow peaks.
2. Strongly peaked emitters, including directional sources and environment maps featuring the sun.
3. Second-order visibility effects, where silhouettes are themselves occluded by surrounding geometry.
4. Discontinuities in the edge parameterization.

The last two points become more pronounced as the geometric complexity increases. For example, although a sphere is a very simple shape, its discretization into a polygonal mesh can generate an extremely challenging integrand.

Yan et al. [39] propose two ways to mitigate these difficulties. First, they recommend rearranging the edges into longer connected chains, which reduces the number of discontinuities in the t parameter. We have not incorporated this optimization into our method, though it should be beneficial here as well.

Second, they replace the grid with a set of kd-trees (one tree per edge chain) to adapt to fine features of the 3D domain that require increased resolution. The construction of each tree is top-down and resembles that of an adaptive quadrature rule: a recursive splitting criterion subdivides the current node until the integrand becomes sufficiently smooth. The failure mode of this approach is also that of adaptive quadrature: a stopping criterion based on local evaluations will sometimes fail to detect sharp peaks and stop the subdivision prematurely.

Our hierarchical guiding uses a set of 100 octrees, each covering an equal-sized interval of the first axis to handle heterogeneity arising from the parameterization of edges, fibers, and the concatenation of multiple objects into a shared boundary sample space. The particular type of hierarchy is not a critical factor (we merely chose octrees since their construction is easy to parallelize on the GPU).

The main difference lies in *how this hierarchy is constructed*. Its construction is bottom-up, starting from a set of projected samples concentrated at the sparse features in boundary sample space. We then simply subdivide each node until reaching a maximum depth (9) or until the node contains one or fewer samples, which yields a partition matching the spatio-directional structure of the integrand. In other words, unlike grid-based guiding, where a projection was used to accumulate density into the voxels of a 3D grid, we now employ the projection to efficiently construct a high-fidelity adaptive space partition. To estimate the actual value of the boundary integrand, we additionally draw 32 uniformly distributed samples within each leaf node. This additional sampling does not require projections. In our experiments, it results in an average of $\sim 6.4 \times 10^7$ samples for a base budget of 10^8 projections, which roughly doubles the initial sample budget.

Figure 4.6 compares this scheme to the method of Yan et al. [39] (“Adaptive quadrature”) with equal time budgets (conservatively, see Table 4.2 for precise timing values) and a grid-based baseline using projections. The experiment demonstrates that the octree significantly outperforms the grid-based baseline. The method of Yan et al. [39] produces spatially non-uniform convergence with low error in some regions and significant outliers in others, leading to a $\sim 190\times$ RMSE difference in the NEPTUNE experiment.

4.2.2 Projection

We now discuss the operations needed to support a particular geometric representation:

Chapter 4. Projective Sampling for Differentiable Rendering of Geometry

Algorithm 4.1: **Projection for spheres.** This code fragment demonstrates the interface of the projection operation for a simple geometric primitive.

```
class Sphere():
    def project(self, segment: Segment):
        O = segment.a # viewpoint
        P = segment.b # point on the sphere

        N = (P - self.center) / self.radius # the normal at P
        CO = O - self.center # center to viewpoint vector
        CO_dir = (O - self.center) / norm(CO)
        proj_dir = normalize(N - dot(N, CO_dir) * CO_dir)

        # phi: the angle from C to the boundary relative to O
        sin_phi = r / norm(CO)
        B = CO_dir * sin_phi + proj_dir * sqrt(1 - sin_phi**2)

        return Segment(shape=self, a=O, b=B)
```

1. A projection that maps a ray segment $\mathbf{x}_a \rightarrow \mathbf{x}_b$ onto a nearby segment $\mathbf{x}'_a \rightarrow \mathbf{x}'_b$ tangential at $\mathbf{x}'_b$.
2. A parameterization of the perimeter (if present), and a parameterization of the interior (for curved geometry).

Spheres

This is by far the simplest case, which can be helpful for testing new implementations of projective sampling. Listing 4.1 provides pseudocode for the projection. Because spheres are smooth and closed, they only need a surface mapping (we use spherical coordinates).

Triangle meshes

We employ two different projection strategies for meshes: JUMP and WALK.

The JUMP projection is a Newton-style iteration based on a local linear approximation. We model the neighborhood of a surface position $\mathbf{p}$ using the parameterization $\tilde{\mathbf{p}}(u, v) = \mathbf{p} + u\partial_u\mathbf{p} + v\partial_v\mathbf{p}$ with an interpolated normal $\tilde{\mathbf{N}}(u, v) = \mathbf{N} + u\partial_u\mathbf{N} + v\partial_v\mathbf{N}$. Under the assumption of a fixed viewing direction, the silhouette of this approximation is a line in the parameter space, allowing us to find a solution (u', v') analytically. From the inferred silhouette point, we subsequently trace a perpendicular ray $(\tilde{\mathbf{p}}(u', v') + \epsilon\tilde{\mathbf{N}}(u', v'), -\tilde{\mathbf{N}}(u', v'))$ to find the next intersection, at which point the procedure could be repeated.

The WALK projection is a greedy search strategy. We visit the three neighboring triangles and

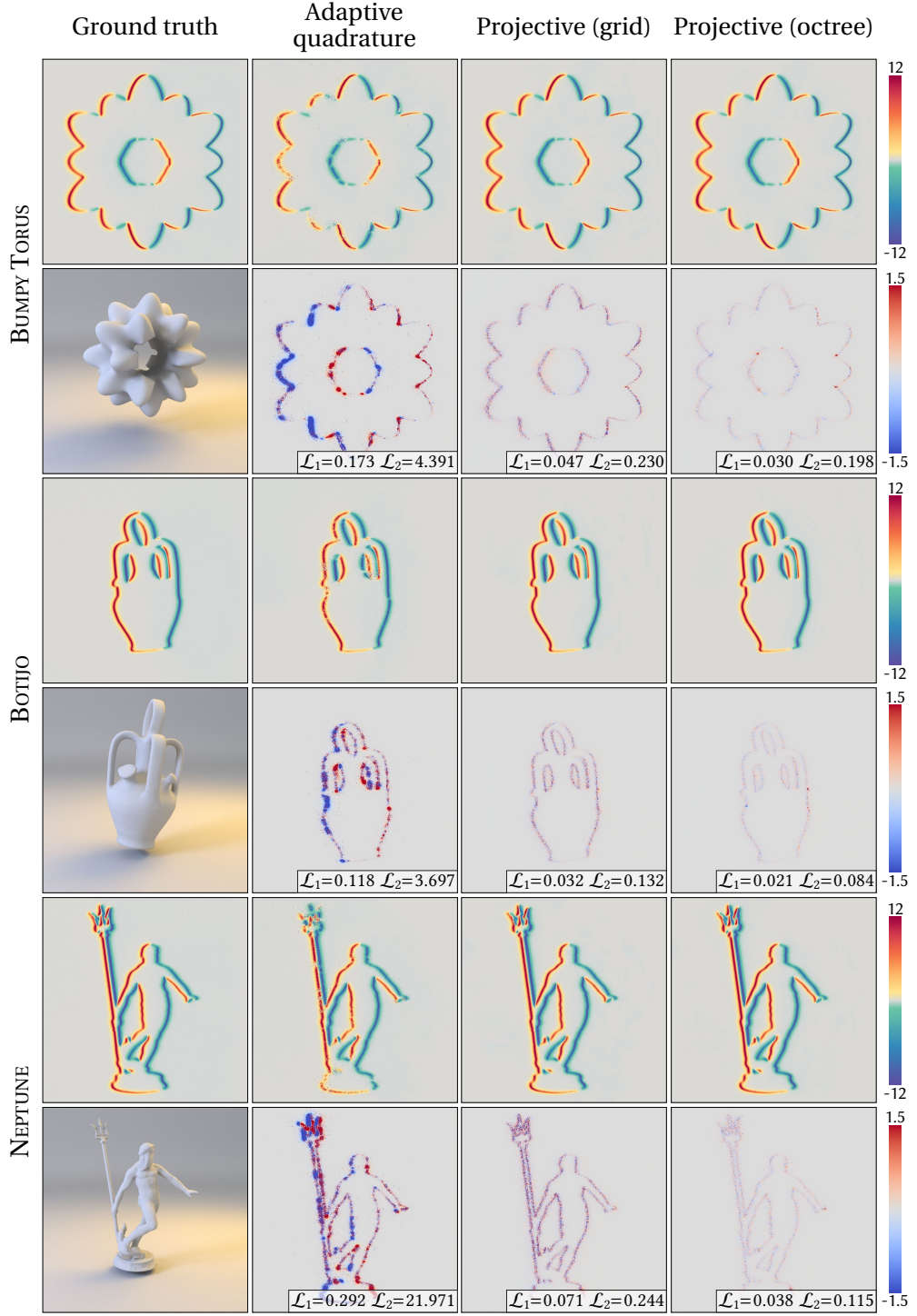


Figure 4.6: **Guiding structures.** Comparison of guiding data structures, including adaptive quadrature [39] (with edge sorting, MIS, and parameters set to use all available GPU memory), a uniform grid with projections, and an octree initialized with projected samples. For equal time budgets, the octree better captures sparse features of boundary sample space and suppresses the outliers that remain visible with the competing approaches. See Table 4.2 for timings.

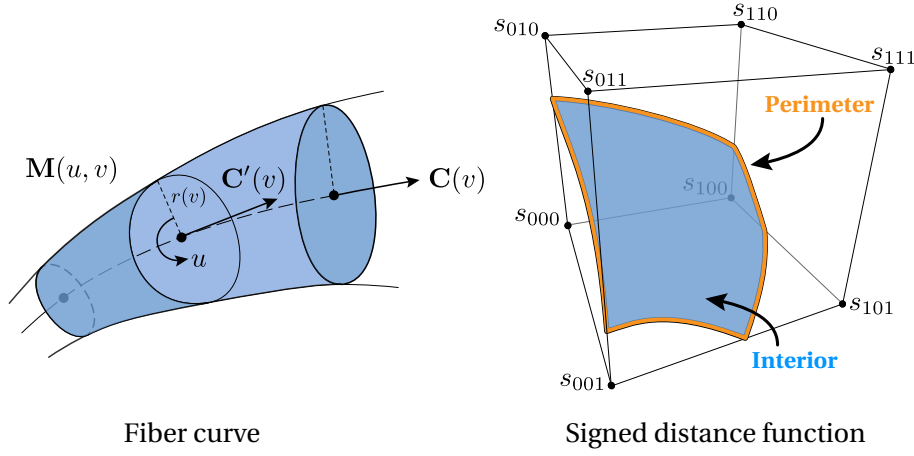


Figure 4.7: **Smooth geometry.** The local formulation of the boundary derivative (Equation 4.2) enables differentiable rendering of smooth geometry, such as cylindrical fibers based on Bézier curves (left) and implicitly defined surfaces represented using a signed distance function (right). The latter case involves derivatives arising from the curved interior and potential normal discontinuities at voxel perimeters.

compute the angles between their normals and the viewing direction. WALK aims to gradually increase this angle until reaching a triangle that is perpendicular (90°) to the viewing direction or back-facing. However, consistently walking to the neighboring triangle with the largest angle tends to attract too many projections to certain mesh edges. Instead, we randomly pick between the two neighbors with the largest angle values, using those values as discrete probabilities.

Both strategies only rely on local differential information. JUMP is more aggressive and can quickly bypass plateaus and highly tessellated regions, while the smaller steps of WALK robustly detect nearby silhouettes even on bumpy geometry, where the extrapolated local model can be deceptive. WALK does not require ray tracing, which makes individual steps much faster than JUMP.

In practice, we adopt a hybrid strategy: WALK for 30 steps and JUMP once if no silhouette is found, followed by an additional 30 WALK steps. This strategy only requires a single ray-tracing step and produces high-quality projections on all meshes considered here. If the projection of the segment $\mathbf{x}_a \rightarrow \mathbf{x}_b$ fails to find a silhouette $\mathbf{x}'_b$, we keep the tentative position $\mathbf{x}'_b$ and generate a tangential segment $\mathbf{x}'_a \rightarrow \mathbf{x}'_b$ with the direction closest to the original direction of $\mathbf{x}_a \rightarrow \mathbf{x}_b$, as outlined at the beginning of Section 4.2. More sophisticated projections could likely handle additional corner cases, but even this simple scheme is already sufficient to build effective guiding distributions.

4.2.3 Fiber curves

We model fibers using a parametric base curve $\mathbf{C}(v)$ and radius $r(v)$, which are both given by cubic B-spline interpolants (Figure 4.7). The cross-section of the surface for a fixed value of v yields a circle with center $\mathbf{C}(v)$, radius $r(v)$, and normal $\mathbf{C}'(v)$. Assigning an azimuth angle parameter u to this circle yields a C^1 -continuous surface $\mathbf{M}(u, v)$. We ignore the curve endpoints (which can, for example, be closed using spheres), in which case only the curved interior part of the local boundary integral in Equation (4.2) matters.

Projection Given a viewpoint $\mathbf{O}$ and surface position $\mathbf{P} = \mathbf{M}(u_0, v_0)$, we fix v_0 and find the value of u that yields a silhouette projection, i.e., $\langle \mathbf{M}(u, v_0) - \mathbf{O}, \mathbf{n}(u, v_0) \rangle = 0$, where $\mathbf{n}$ is the parameterized surface normal. This equation can be expanded into the form

$$A \cos^2 u + B \cos u \sin u + C \cos u + D \sin u + E = 0,$$

where A, B, C, D, E are u -independent constants that can be computed analytically. The equation has an analytic solution when the fiber radius is constant. Otherwise, we run 20 bisection iterations, which is fast and robust.

4.2.4 Implicit functions

We also evaluate the local boundary formulation on *signed distance functions* (SDFs) represented using a grid-based trilinear interpolant (Figure 4.7). This representation has two advantages:

1. **Speed.** Intersections can be computed with the help of a mesh proxy and analytic solutions within voxels, which avoids costly sphere tracing steps [50].
2. **Simplicity.** The low-order representation simplifies the mapping from $\mathcal{R}^2$ to the SDF surface.

The trilinear representation also has a clear disadvantage: geometric normals are discontinuous across voxel boundaries, which means that both interior and perimeter terms of Equation (4.2) must be considered. Note that we do not use the *signed distance* property of the SDF representation, so much of the following could in principle generalize to other types of implicitly defined surfaces.

Parameterization. To define functions parameterizing the interior and perimeter, we first make the following observations:

1. **Interior.** Given the trilinear interpolation scheme, a given pair of x, y coordinates within a voxel has at most one root along the z axis.
2. **Perimeter.** Similarly, the surface intersection with a voxel face can be uniquely parameterized by the face index and a perpendicular coordinate, which can be used to create

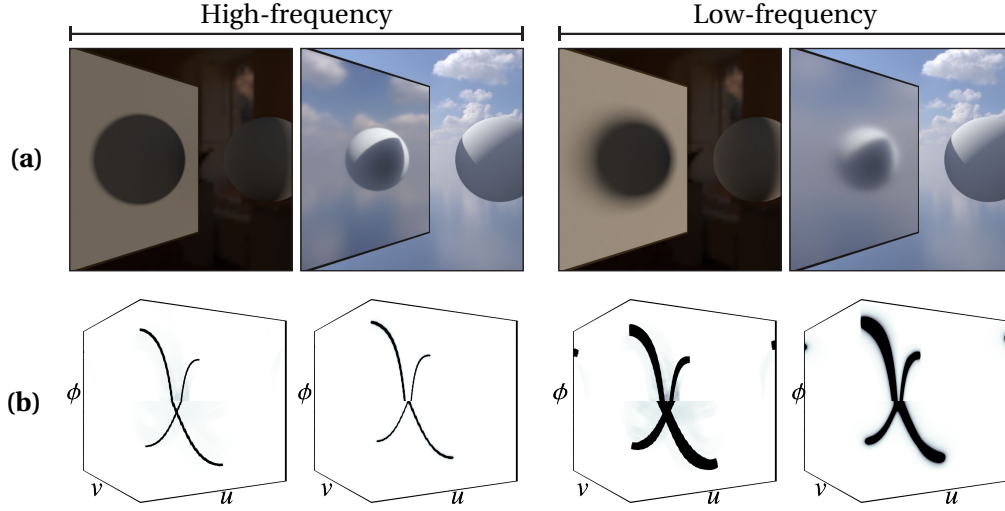


Figure 4.8: **Shadows and reflections produce a qualitatively similar boundary integral.** (a) Primal rendering of scenes with shadows or reflections. (b) The corresponding boundary sample space. This figure demonstrates their similarity for both high-frequency (hard shadows, sharp reflections) and low-frequency (soft shadows, rough reflection) cases. Consequently, most of our gradient comparisons focus on the reflection case.

a flattened 1D mapping across all possible perimeter curves, reminiscent of the mesh case. Two additional dimensions represent the direction.

We use these properties to create a globally discontinuous per-voxel mapping. This mapping depends on the choice of a dimension used to parameterize the curve/surface, which is unstable when the chosen dimension is nearly perpendicular. We assign the most numerically stable dimension to each voxel based on the SDF gradient.

A projection operation for SDFs was not implemented here, though related operations suggest that such an extension should be feasible [51]. The SDF results presented below therefore use a grid data structure with uniform initialization.

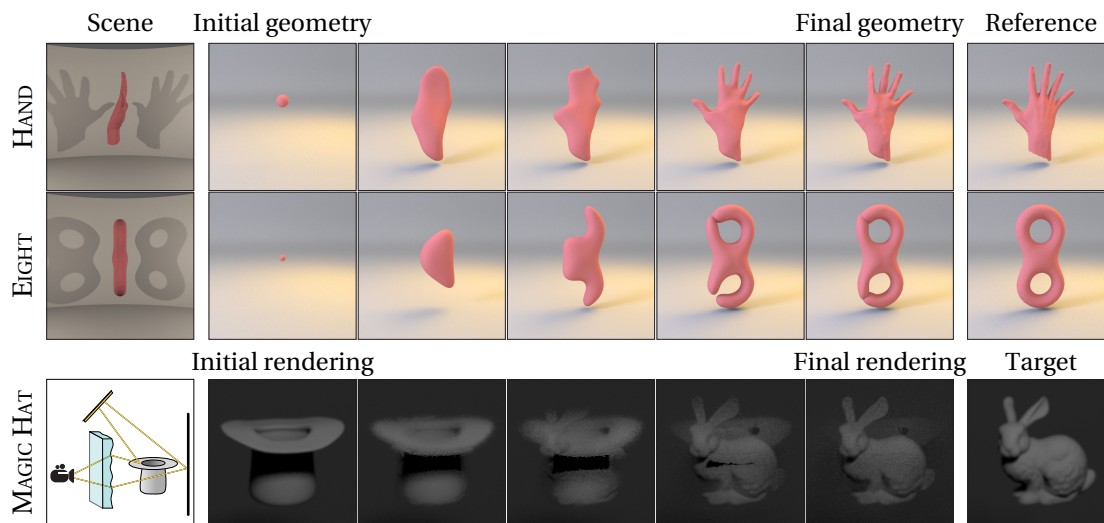


Figure 4.9: **Optimization tasks involving discontinuous derivatives.** In the first two experiments, we reconstruct the shape of an object from a single-view capture. In the last row, we optimize the position of a hat and the height field of a refractive plate with a small amount of roughness (Beckmann $\alpha = 0.01$) to form an image of a bunny.

4.3 Results

We implemented our method on top of Mitsuba 3’s [31] `cuda_ad_rgb` backend, using the underlying Dr.Jit [52] framework for forward- and reverse-mode AD and GPU kernel compilation.

4.3.1 Experimental setup

Test scene. The test scene used throughout this chapter places the shape in front of a uniform area emitter. The camera views their reflections on a conductive plane with isotropic roughness (Trowbridge and Reitz [53], $\alpha = 0.005$). This ensures that the derivative image contains only contributions from indirectly observed discontinuities, which are the main focus of this chapter.

We use sharp reflections to investigate the performance of different methods: a rougher material or a soft shadow would blur the integrand in boundary sample space, potentially concealing inaccuracies or flaws in the method. As illustrated in Figure 4.8, shadows and reflections can produce a qualitatively similar boundary sample space. Instead of a reflective plane, one could therefore equivalently use a diffuse receiver with a directionally peaked emitter.

Initialization time. Tables 4.1 and 4.2 list the time needed to initialize the guiding distributions for the previously discussed experiments in Figures 4.4 and 4.6. The runtime of the differential rendering phase is not noticeably impacted by the choice of guiding representation.

Chapter 4. Projective Sampling for Differentiable Rendering of Geometry

Table 4.1: Guiding initialization time (in seconds) for Figure 4.4.

	EmptyCube	Bunny	Fertility	Filigree	Dragon
Uniform (5x)	0.52	0.49	0.52	0.46	0.47
Uniform (50x)	4.30	4.05	4.36	3.71	3.80
Projective	0.41	0.29	0.30	0.30	0.29

Table 4.2: Guiding initialization time (in seconds) for Figure 4.6.

	BumpyTorus	Botijo	Neptune
Adaptive quadrature	1.12	1.11	1.11
Projective (grid)	0.39	0.26	0.26
Projective (octree)	0.73	0.48	0.46

Table 4.3: Guiding initialization time (in seconds) for Figure 4.10.

	Knot	Coil	Winding	Sausage	KnittingYarn
Uniform (grid, 5x)	0.75	0.68	1.92	0.98	0.59
Uniform (grid, 50x)	7.22	6.52	18.98	9.68	5.48
Projective (grid)	0.13	0.11	0.30	0.12	0.14
Projective (octree)	0.26	0.26	0.64	0.30	0.33

In these gradient comparison experiments, derivative images were rendered using 256 samples per pixel (~ 0.8 seconds of rendering time), a sample count typically higher than necessary for inverse rendering tasks. This higher sample count is used to obtain reference-quality derivative images for evaluation.

4.3.2 Derivative estimation

Fiber curves. Figure 4.10 shows the influence of the guiding distribution on differentiable rendering of curves. It highlights the effectiveness of projective sampling, which outperforms uniform sampling even when the latter uses $50\times$ more samples. The octree representation reduces variance further in complex cases such as the `KNITTING YARN` experiment, where second-order visibility effects (occlusion of silhouettes) dominate.

4.3.3 End-to-end optimization

End-to-end optimizations. Figure 4.9 demonstrates our method in several complete reconstruction tasks. In the first two experiments, the optimization must infer the shape from a single view that depicts the object and two shadows on a curved diffuse wall. We use the method of Nicolet et al. [54] to smoothly evolve the triangle mesh and periodically re-mesh the geometry to a finer resolution to add more degrees of freedom. Listing 4.2 shows the high-level pseudocode of the optimization algorithm.

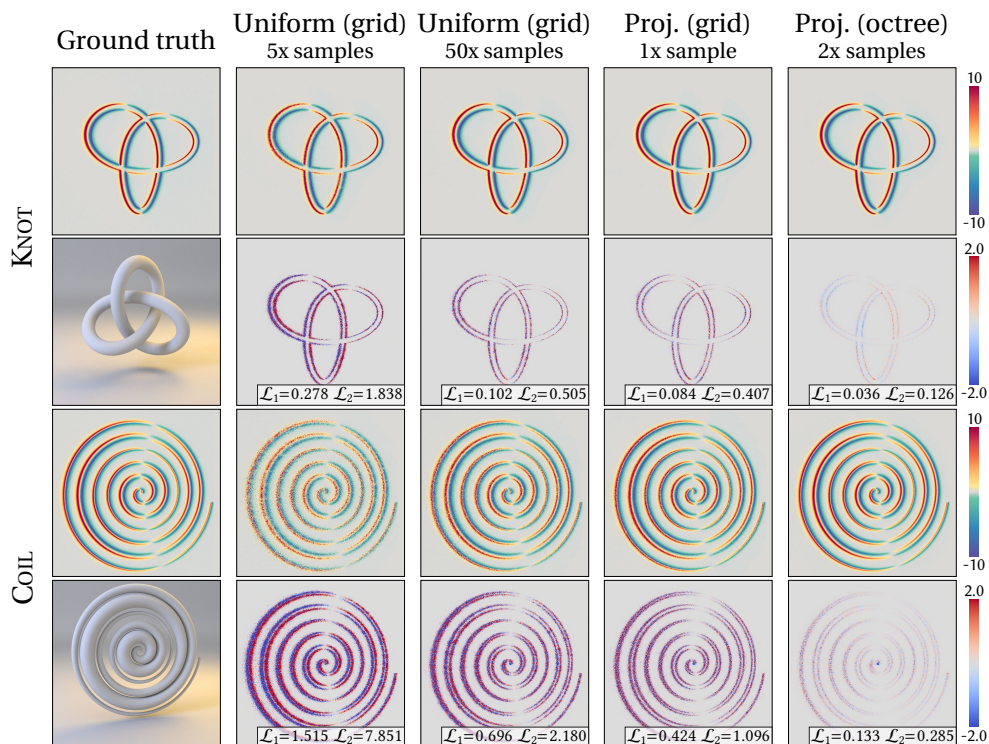


Figure 4.10: **Fiber derivatives.** Experimental validation of fiber derivatives, analogous to the previous experiments. See Table 4.3 for timings.

The final row of Figure 4.9 shows a *Magic Lens* setup as in Papas et al. [55]. The objective is to optimize the surface of a dielectric plate so that an object located behind it appears as a different object from a specific viewpoint. We place a hat behind the lens and optimize the lens surface so that the refracted appearance resembles a bunny. In addition to the lens surface, we also optimize the hat position. A sketch of this setup is shown in the leftmost column.

Influence of gradient quality. Figure 4.14 illustrates the influence of gradient quality in a challenging reconstruction task. The scene is designed so that the shadows reveal enough information to disambiguate the shape and, in principle, enable convergence using gradient descent. Because the computed boundary derivatives are strictly local, the optimization proceeds by progressively extruding the chair legs in the thin regions where the tentative geometry overlaps with the reference. This localized convergence behavior serves as an important point of contrast for the non-local methods developed in the next chapter.

We reuse the optimization strategy from Figure 4.9 and perform grid-based guiding with the same resolution in all five optimization runs. The only difference is the initialization of the grid and the resulting gradient quality. We also visualize forward gradients for a horizontal displacement to illustrate the differences in convergence.

Optimization with low-quality gradients (“*uniform, 3x samples*”) stagnates, while estimates

Chapter 4. Projective Sampling for Differentiable Rendering of Geometry

Algorithm 4.2: Pseudocode for the optimization loop and a vectorized reverse-mode implementation of the rendering function.

```
def optimize():
    for l in range(N_iter):
        loss = l2(render(scene), reference)
        loss.backward()
        optimizer.step(scene.grad)

@backward
def render(scene, grad_img):
    rays = scene.sensor.sample_rays()
    # Propagate derivative of the continuous part with PRB and return path segments
    img_cont, segments = prb_backward(grad_img)
    shape = segments.shape
    # Project path segments, returning an array of new segments and failure markers.
    proj_segments = shape.project(segments)

    # Convert to a 3D point in boundary sample space
    boundary_sample = shape.map_to_sample(proj_segments)
    # Evaluate the boundary integrand without motion
    value = eval_integrand(proj_segments)
    # Initialize the guiding structure (octree may call eval_integrand internally)
    distr = guiding_octree(boundary_sample, value)

    # Draw samples from the guiding distribution
    boundary_sample, pdf = distr.sample_pdf()
    boundary_segment = shape.map_to_segment(boundary_sample)
    # Particle-tracer-style render starting from a boundary segment
    img_boundary = render_boundary(boundary_segment, pdf)
    img_combined = img_boundary + img_cont
    img_combined.backward(grad_img)
```

with lower variance (“*uniform, 20x/50x/120x samples*”) lead to progressively better convergence when compared at the same iteration count (though this comes at a greatly increased computational cost). Our method (“*projective, 1x sample*”) achieves a good balance between gradient quality and computation time.

4.3.4 Implicit and indirect effects

Signed distance functions. Figure 4.12 validates the use of our local boundary integral formulation (Equation 4.2) for SDFs. We separate the derivative contributions from the perimeter and interior, which superimpose to produce a gradient that matches the ground truth.

Importance of indirect derivatives. Figure 4.13 shows the impact of indirectly observed discontinuities in a single-view mesh reconstruction. Such indirect effects are an inherent property of almost any realistic image. When self-shadowing is not taken into account during differentiation, the optimizer lacks the information that the shadow near the nose is caused by the eyebrow. It attempts to conceal the undesired shadow by distorting the nose to cover it. With self-shadowing derivatives computed by our method, the optimizer instead subtly adjusts the eyebrow. Other aspects (e.g., nose height) also improve, since shadows reduce ambiguities in the challenging single-view setting.

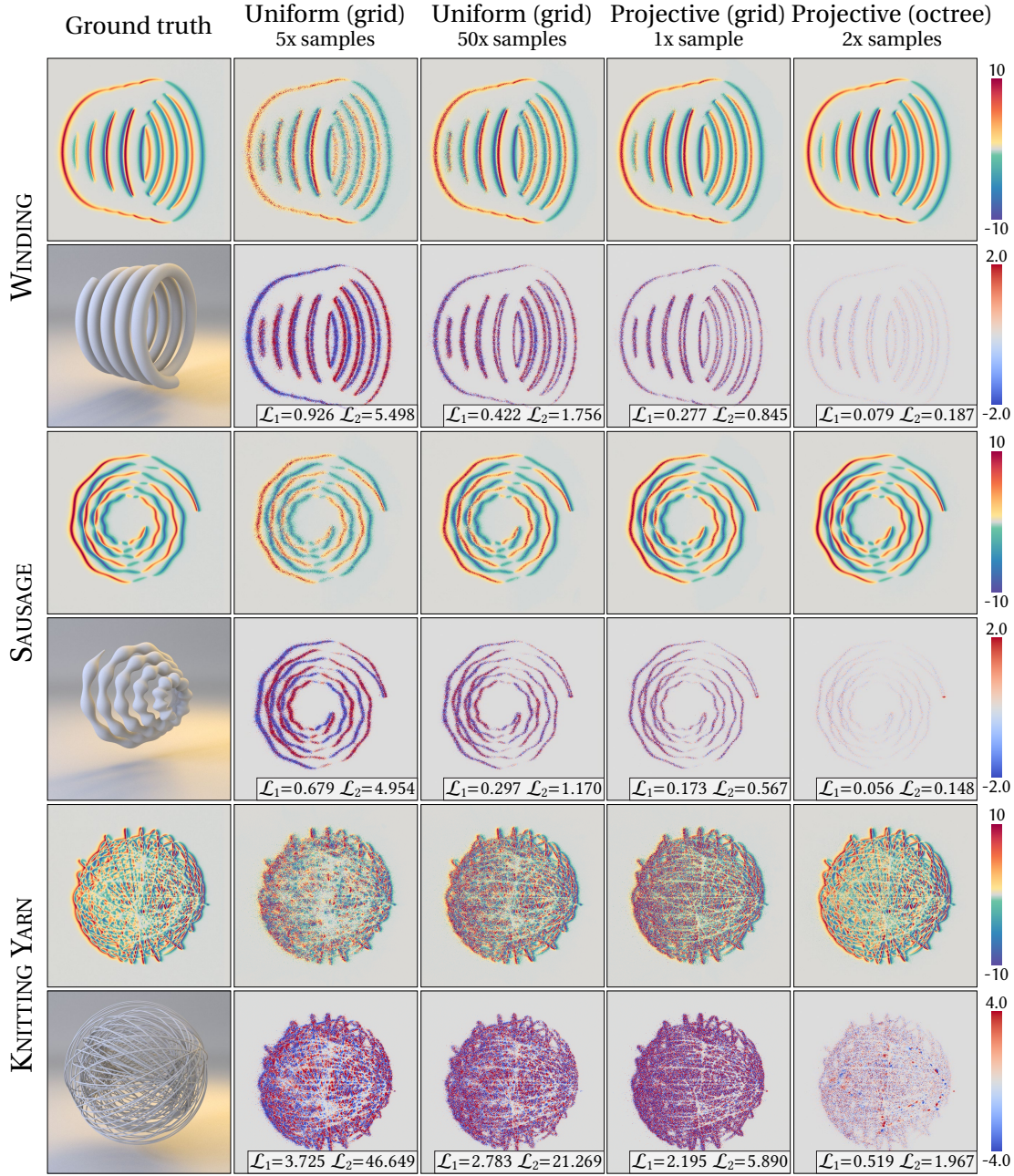


Figure 4.11: **Complex fibers.** The same experiment as in Figure 4.10, but with more intricate fiber configurations and stronger occlusion between silhouettes. These cases further emphasize the benefit of projective sampling and adaptive guiding when second-order visibility effects dominate. See Table 4.3 for timings.

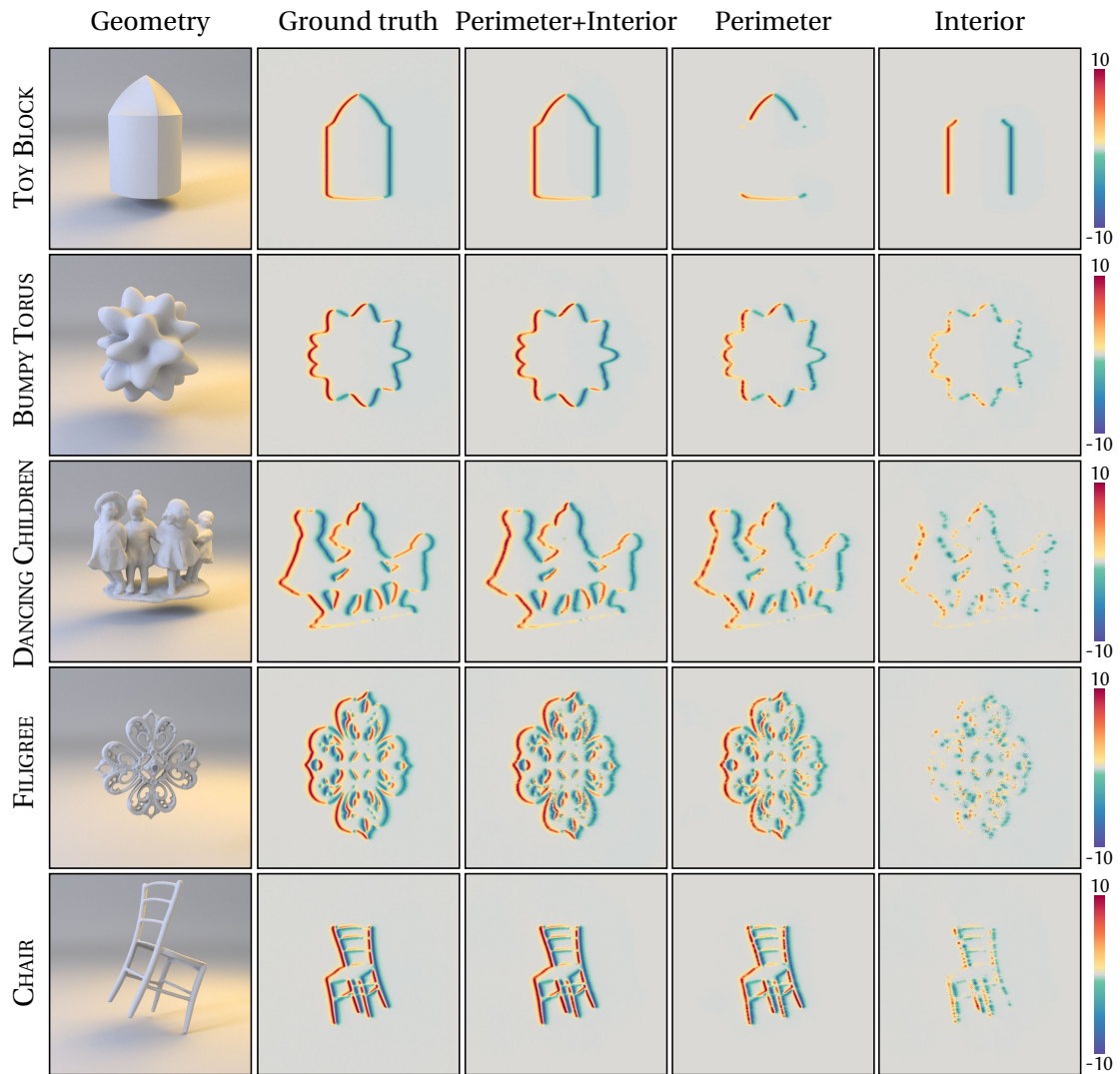


Figure 4.12: **SDF derivatives.** This visualization shows visibility-induced derivatives of signed distance functions (SDFs), separating the contributions from the perimeter and interior. These examples all use a 128^3 SDF grid with trilinear interpolation.

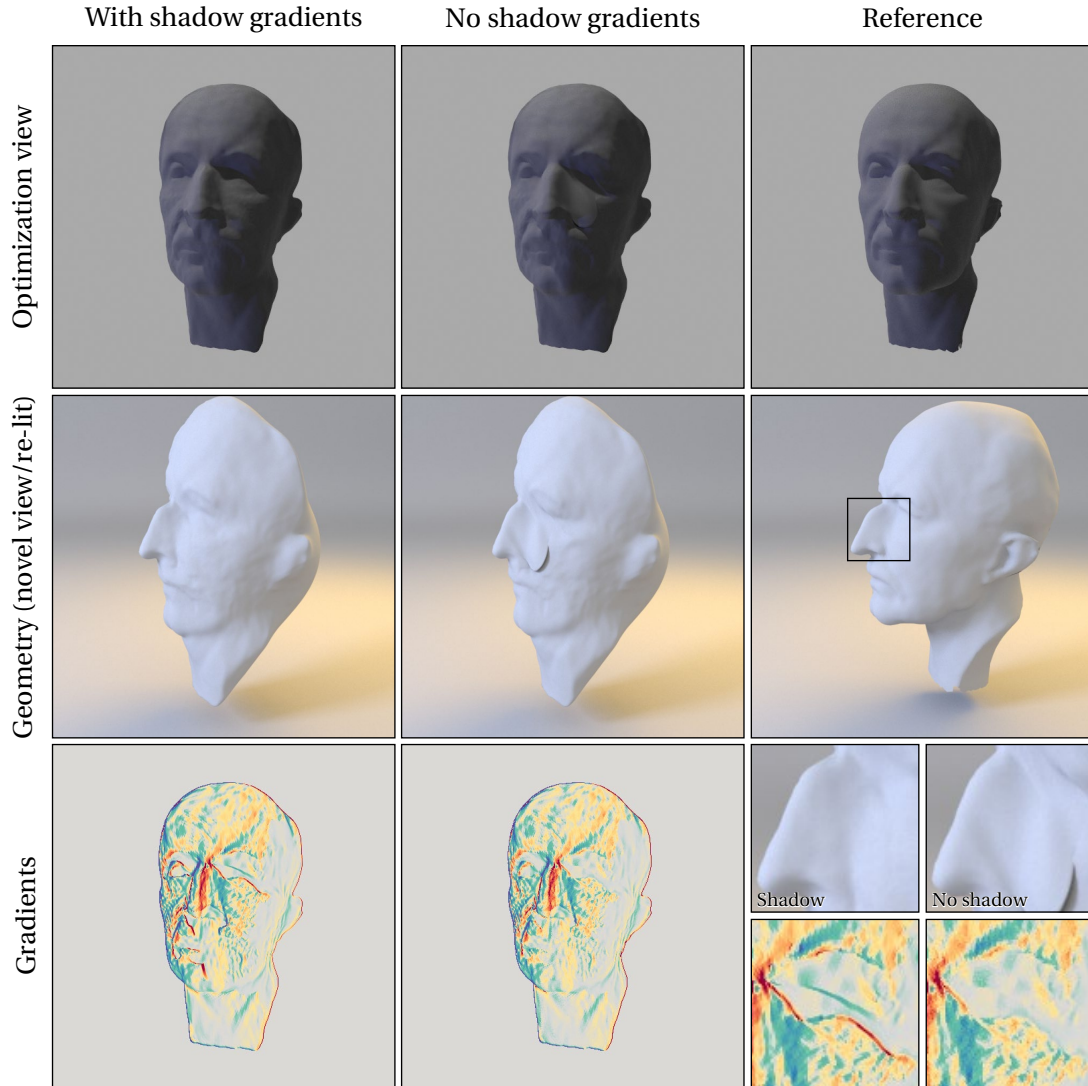


Figure 4.13: **Indirect derivatives.** Single-view reconstruction *with* and *without* derivatives due to self-shadowing. Accounting for these derivatives resolves ambiguity in this information-constrained scenario. The gradient image in the last row (with respect to a horizontal translation) highlights the missing signal needed to attribute the shadow to the correct geometric cause.

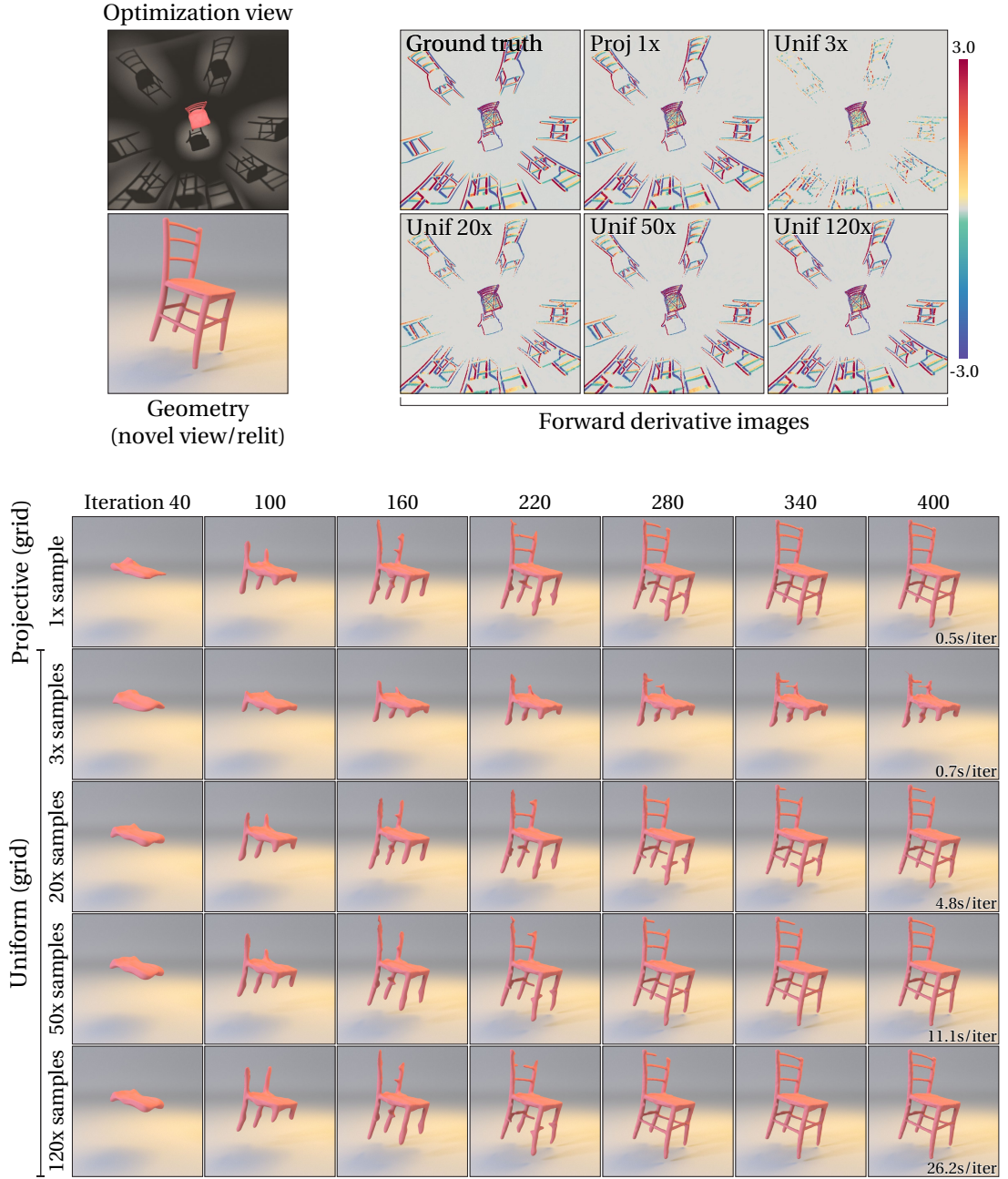


Figure 4.14: **The effect of gradient quality on geometric reconstruction.** We reconstruct a triangle mesh from a single reference view with multiple shadows on a curved floor (upper left), comparing projective and uniform sampling for different iteration counts and sample counts. Because the boundary derivatives evaluated here are strictly local, the legs of the chair are progressively extruded only where the tentative object overlaps with the reference shadows. Timing values in the rightmost column quantify discontinuity-related costs while ignoring the shared overhead of primal rendering, preconditioned gradient descent [54], and re-meshing.

4.4 Conclusion

A central discovery of path-space differentiable rendering [27] was that the interior and boundary derivatives decouple, allowing each part to be handled independently. This chapter shows, however, that estimator design should still exploit the coupling between them: boundary estimation benefits substantially from information gathered during the simulation of the interior term.

This idea leads to a modular framework of projections and parameterizations that fit into the established architecture of contemporary rendering systems. Possible constructions include hopping from triangle to triangle, jumping over larger distances, and the numerical solution of nonlinear equations. This suggests a broader design space of projection operators. One promising direction is to target linear elements specifically to better handle the difficult “blade of grass” case mentioned earlier.

The discontinuous nature of the boundary sample space remains a significant source of inefficiency. Both prior work [39] and this chapter demonstrate that edge permutations and adaptive discretizations can mitigate this problem to a certain extent, but they do not remove the underlying mismatch. The representation is intrinsically smooth, but that smoothness is lost when the representation is mapped to a parameter space. Finding smoother global parameterizations of this domain remains an interesting open problem.

Subsequent work highlights two complementary directions. Quadric-based silhouette sampling [56] improves edge sampling itself by using stronger rejection tests and traversal heuristics, making direct silhouette sampling competitive again in unidirectional settings. Fixed-step walk-on-spherical-caps visibility [57] instead strengthens warped-area reparameterization by constructing velocity fields through closest-silhouette queries on the sphere. Both developments reinforce the perspective of this chapter: the boundary integral is the right object, but the best way to reach it depends strongly on the geometry representation and the proposal mechanism.

The next chapter takes a complementary direction: instead of further refining local boundary sampling on surfaces, it extends these surface derivatives into space using many-worlds perturbations. In the local limit, that formulation is consistent with the same surface derivative theory used here, but it targets the separate bottleneck of sparse and local gradient support.

5 Many-Worlds Inverse Rendering

The previous two chapters focused on the hardest corner of the thesis design space: explicit surfaces under physically based transport. They established how to compute geometry derivatives correctly and how to sample the visibility term more efficiently. But the final section of Chapter 3 also identified a separate bottleneck: even correct surface derivatives remain sparse and local, which makes optimization slow, sensitive to initialization, and poorly suited to topological change.

This chapter addresses that bottleneck by moving along the surface–volume axis while keeping the physically based transport model. Instead of abandoning surfaces in favor of an exponential volume, it relaxes the optimization problem itself: auxiliary geometric degrees of freedom are introduced during training, but the final output is still an explicit surface. The resulting method, *many-worlds inverse rendering*, can be read in two equivalent ways: as inverse path tracing over a distribution of hypothetical surfaces, or as an extension of surface derivatives from the current surface into the surrounding space.

Given a loss function $\mathcal{L}$ and a rendering $\mathcal{R}(\pi)$ for tentative parameters π , we seek to minimize their composition

$$\pi^* = \operatorname{argmin}_{\pi \in \Pi} \mathcal{L}(\mathcal{R}(\pi)). \quad (5.1)$$

A default strategy is to evolve a scene representation (e.g., a triangle mesh) directly in Π . In physically based inverse rendering, this is brittle because the optimizer can only move the surface using information that is already present on or near that surface.

To avoid this limitation, we optimize on an extended parameter space:

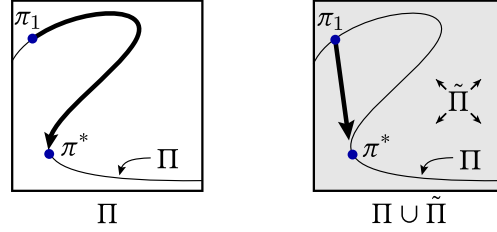
$$\pi^*, \tilde{\pi}^* = \operatorname{argmin}_{\pi \in \Pi, \tilde{\pi} \in \tilde{\Pi}} \mathcal{L}(\tilde{\mathcal{R}}(\pi, \tilde{\pi})), \quad (5.2)$$

where $\tilde{\mathcal{R}}$ is parameterized by $\tilde{\pi} \in \tilde{\Pi}$ representing features that we are *unwilling to accept* in the final solution. If the role of these auxiliary dimensions diminishes over time, we simply discard $\tilde{\pi}^*$ and keep π^* .

The parameter space extension serves two purposes: it turns a circuitous trajectory through a

Chapter 5. Many-Worlds Inverse Rendering

non-convex energy landscape into a more direct route by using extra degrees of freedom, and it turns visibility changes that are discontinuous on the surface into continuous updates in the extended domain. In this sense, the method is related to numerical continuation: optimization is first carried out in an easier extended problem, and the auxiliary dimensions are discarded once the ordinary surface representation is recovered.



Concretely, we retain a surface (denoted $\bar{S}$) and augment it with a spatial representation S to model *non-local perturbations* of that surface. This can be read in two equivalent ways:

- as inverse path tracing with a distribution of hypothetical surfaces that compete as independent explanations of the observations, and
- as an extension of the surface-derivative domain from the current surface into space.

The crucial modeling choice is that these additional hypotheses are independent, competing explanations of the observations; only one explanation should survive locally in the recovered surface. This “many-worlds” viewpoint preserves much of the exploratory behavior that makes volumetric optimization robust, while avoiding transmittance and multiple-scattering computations in the optimization loop.

There is a direct connection to the previous chapter: in the limiting case where perturbation positions coincide with $\bar{S}$, our derivatives recover standard surface derivatives [Zhang et al. 2020, Zhang et al. 2023]. In that sense, this chapter generalizes the same surface-derivative theory rather than replacing it.

Although this chapter sometimes refers to the spatial representation S as a “volume”, the method is not volume reconstruction: we do not optimize S to match images directly, and there is no transmittance or multiple-scattering transport in the optimization process.

The remainder of the chapter develops this idea by first motivating the extended parameter space more concretely, then deriving the associated derivative transport law, and finally explaining how to render and optimize the representation in practice before evaluating the resulting method.

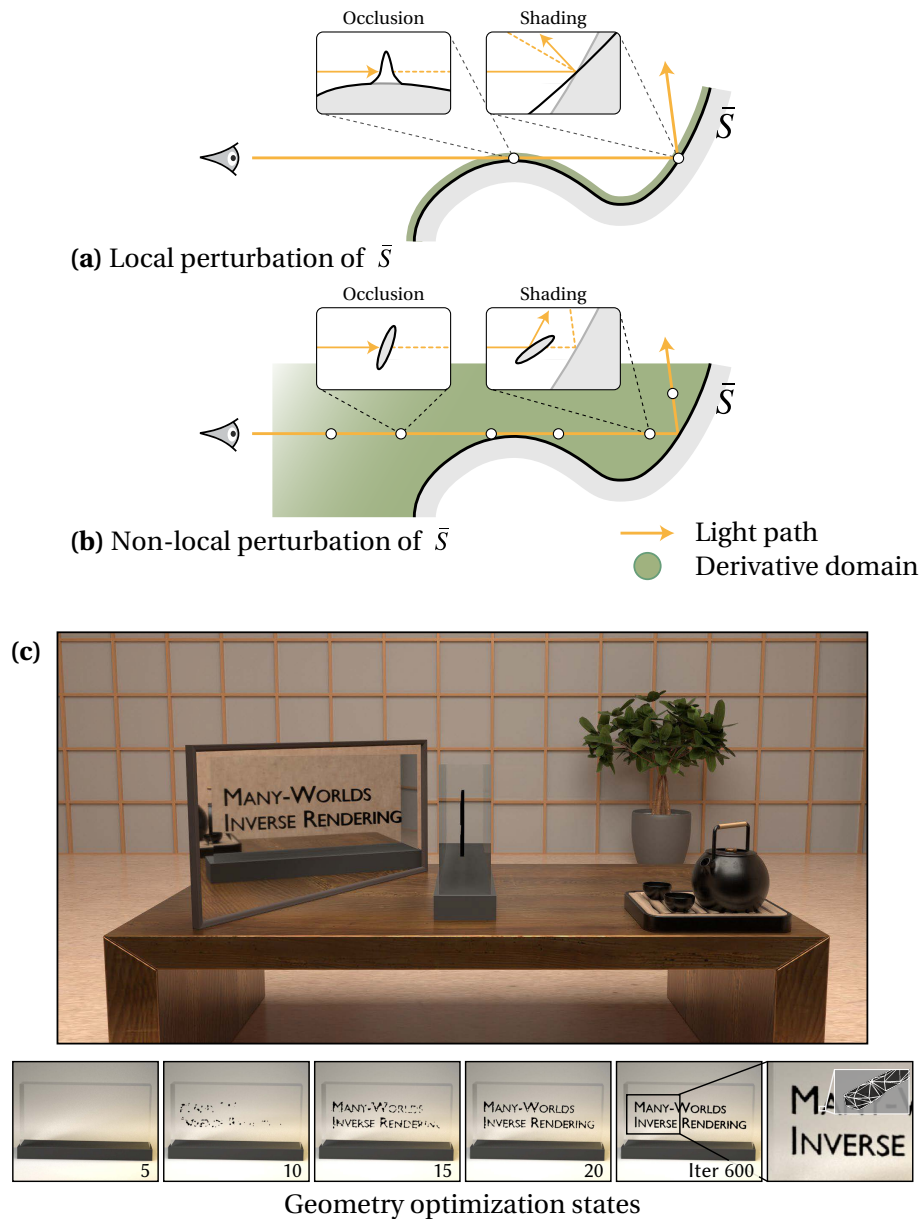


Figure 5.1: **Many-worlds overview.** (a) Prior differentiable rendering methods compute how surface deformations affect light transport through changes in occlusion and shading. The resulting gradients drive local geometric adjustments. (b) The method instead considers adding hypothetical surface patches anywhere in 3D space. We simultaneously evaluate many such patches as independent, competing explanations of the input data. This approach, termed *many-worlds derivatives*, extends gradient computation from surfaces into the surrounding space. It combines the robustness of volumes with the efficiency of surface rendering: the optimization does not require an initial mesh and can start from an empty scene, while avoiding the expense of transmittance and multiple-scattering computations. (c) An example reconstruction using many-worlds derivatives: a triangle mesh embedded in glass and observed through a mirror.

5.1 Method

5.1.1 Motivation

Extended parameter space Our algorithm optimizes a surface $\tilde{S}$ in an indirect manner via a distribution S of potential surfaces that defines an associated density field in 3D space. The surface $\tilde{S}$ is produced from this field, for example, by extracting a level set. By modifying the distribution S rather than $\tilde{S}$ directly, we enable gradient propagation throughout the entire space, not just on the surface itself. The details of how these spaces are defined are orthogonal to the main idea.

Figure 5.1b demonstrates the effect of adding a *hypothetical surface patch* (drawn from S) into a scene containing the surface $\tilde{S}$. This patch modifies how light propagates in the scene, thereby changing the surface rendering of $\tilde{S}$. By adjusting the existence probabilities and properties (e.g., normals, BRDFs) of such patches, we iteratively improve $\tilde{S}$ to better match the target image.

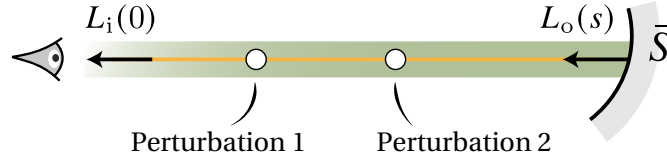
We refer to this as a *non-local perturbation* of $\tilde{S}$: optimizing such hypothetical patches propagates derivatives across the entire domain of S , rather than confining updates locally on the surface.

There is a noteworthy connection to prior work: in the limiting case where the perturbation position coincides with $\tilde{S}$ itself, our derivatives exactly match standard surface derivatives in physical light simulation [Zhang et al. 2020, Zhang et al. 2023]. This equivalence shows that the formulation here generalizes local surface evolution while preserving its geometric meaning.

The optimization converges when no further perturbation improves the match between $\tilde{S}$ and the target image. At this point, we discard S and keep $\tilde{S}$.

Within each iteration of the optimization, $\tilde{S}$ serves as a static background, providing base colors for perturbations without being optimized itself. Thus, we also refer to $\tilde{S}$ as the *background surface*.

Conflicting possibilities Consider a simple case of two non-local perturbations along a ray. We think of them as competing candidates for improving the agreement between the radiance arriving from $\tilde{S}$ and a reference. For example, suppose that $\frac{\partial \text{loss}}{\partial L_i}$ indicates that the current pixel’s radiance is too high—in this case, the same information should be propagated to both positions without weighting.



Here, it might seem natural to distribute the target update between the two positions—say, by scaling it by $\frac{1}{2}$. However, this weighting implicitly assumes the two perturbations *compound* to refine the same surface $\bar{S}$. In reality, they are *mutually exclusive* possibilities—once optimization converges, only one will contribute to the final radiance, without any kind of blending.

This principle extends to more than two perturbations: all candidate positions along the ray should receive the same target update, as if they existed in *many worlds* that do not interact. Ultimately, the ray will intersect only one of these possibilities, which becomes part of the final surface.

The above discussion provides the intuitive motivation for the method. Our next goals are therefore to quantify non-local perturbations and derive the derivative transport law.

5.1.2 Many-worlds derivative transport

Single perturbation case Consider a scene containing only the background surface $\bar{S}$, where the radiance propagating along ray $(\mathbf{y}, \mathbf{x})$ with direction $-\boldsymbol{\omega}$ remains constant:

$$L_i^{\bar{S}}(0) = L_o^{\bar{S}}(s),$$

where $L_i^{\bar{S}}(0) := L_i^{\bar{S}}(\mathbf{x}, \boldsymbol{\omega})$ (incident radiance at $\mathbf{x}$) and $L_o^{\bar{S}}(s) := L_o^{\bar{S}}(\mathbf{y}, -\boldsymbol{\omega})$ (outgoing radiance at $\mathbf{y}$) are parameterized by distance.

For a non-local perturbation candidate at distance t , we model its impact on radiative transport as:

$$L_i(0) = \alpha(t) \underbrace{L_o^{\bar{S}}(t)}_{\text{perturbed}} + [1 - \alpha(t)] \underbrace{L_o^{\bar{S}}(s)}_{\text{original}}, \quad (5.3)$$

where $\alpha(t) \in [0, 1]$ is the probability of a hypothetical surface patch existing at t . The perturbed radiance $L_o^{\bar{S}}(t)$ computes reflected radiance *as if* the patch were inserted at t , while the rest of the scene remains $\bar{S}$.

While this formulation is of little use for physically based rendering, its derivative (with respect to any parameter π) provides the means to optimize over the extended parameter space:

$$\partial_\pi L_i(0) = \underbrace{\partial_\pi \alpha(t) [L_o^{\bar{S}}(t) - L_o^{\bar{S}}(s)]}_{\text{(i) Occlusion}} + \underbrace{\alpha(t) \partial_\pi L_o^{\bar{S}}(t)}_{\text{(ii) Shading}}. \quad (5.4)$$

If $\partial_\pi L_i(0)$ is positive, we can increase the incident radiance by

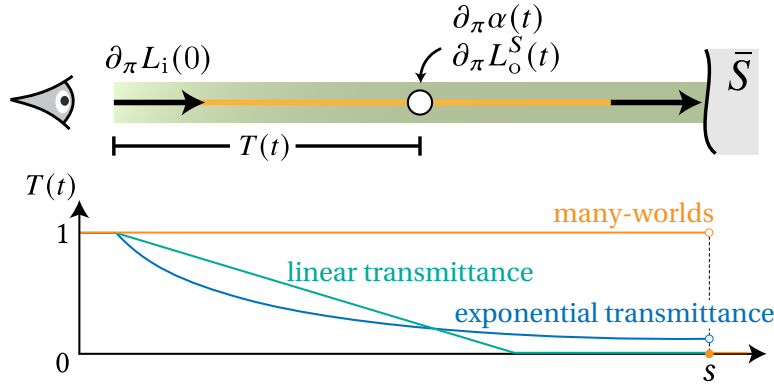


Figure 5.2: **Many-worlds derivative transport.** The propagated derivative at distance t is only weighted by the local radiance difference and the local occupancy, respectively (Equation 5.6). In contrast to exponential or non-exponential (e.g., linear [58]) volumes, the notion of *transmittance* disappears because it would model nonsensical inter-world shadowing.

1. Increasing $\alpha(t)$ if this perturbation is favorable, i.e., $L_o^S(t) > L_o^S(s)$, or lowering it otherwise.
2. Increasing the reflected radiance $L_o^S(t)$, e.g., by altering the normal or BRDF of the hypothetical surface.

Both operations locally update the distribution S at t .

Multiple perturbation case We now extend to the situation where every point along the ray can be considered a potential perturbation. As discussed in subsection 5.1.1, we aim to distribute the target update uniformly across all possibilities along one segment.

Similar to the reverse-mode derivative of an addition $c = a + b$ that simply forwards the derivative toward its operands a and b without weighting ($\frac{\partial c}{\partial a} = 1$, not $\frac{1}{2}$), we sum radiative contributions from distinct perturbations using a continuous integral to achieve an analogous propagation behavior:

$$L_i(0) = \int_0^s \left(\underbrace{\alpha(t) L_o^S(t)}_{\text{candidate at } t} + [1 - \alpha(t)] \underbrace{L_o^S(s)}_{\text{background}} \right) dt. \quad (5.5)$$

Differentiating it yields the many-worlds derivative transport law:

$$\partial_\pi L_i(0) = \int_0^s \left(\partial_\pi \alpha(t) [L_o^S(t) - L_o^S(s)] + \alpha(t) \partial_\pi L_o^S(t) \right) dt. \quad (5.6)$$

The derivative in Equation 5.6 lacks transmittance terms that would ordinarily model attenuation along the ray (Figure 5.2). This stems from the core principle that distinct worlds must not interact.

The only way in which the background surface $\bar{S}$ manifests in this equation is to provide a single baseline radiance value $L_o^S(s)$ needed to compute radiance differences. As a result, $\bar{S}$

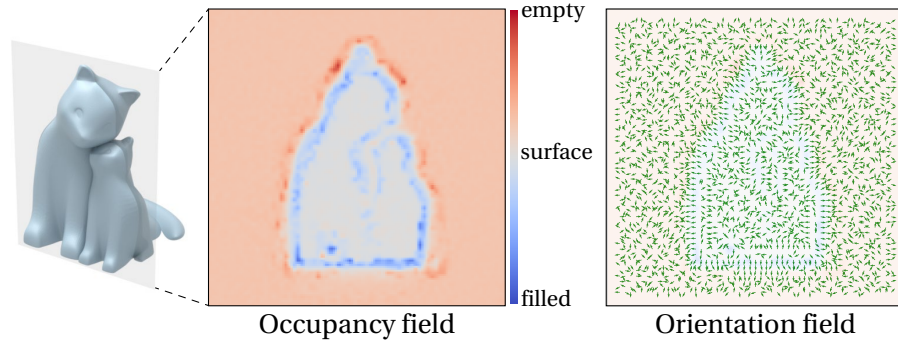


Figure 5.3: **Occupancy and orientation fields.** The images visualize the contents of an occupancy (α) and orientation (β) field after optimization. The former models the probability of a surface existing at a position, while the latter assigns normal directions.

does not directly receive gradients; yet it still evolves during optimization as a result of changes in the distribution S that generates $\tilde{S}$.

Parameterization Equation 5.6 requires derivative propagation toward two quantities in the extended parameter space: the probability of encountering a surface within the distribution, and the outgoing radiance $L_o^S(\mathbf{x}, \omega)$ determined by its properties.

Any S with differentiable realizations of these quantities is, in principle, suitable—we use an occupancy field $\alpha(\mathbf{x}) : \mathbb{R}^3 \rightarrow [0, 1]$ and an orientation field $\beta(\mathbf{x}) : \mathbb{R}^3 \rightarrow \mathcal{S}^2$ (Figure 5.3).

The orientation field $\beta(\mathbf{x})$ assigns a normal direction to the surface patch at $\mathbf{x}$. The occupancy field $\alpha(\mathbf{x})$ [59, 60] models the probability

$$\alpha(\mathbf{x}) := \Pr\{\mathbf{x} \text{ is inside of } S\}, \quad (5.7)$$

and is set to zero when the back side is encountered ($\omega \cdot \beta(\mathbf{x}) < 0$).

Anisotropy is crucial for physically based inverse rendering. Figure 5.4 demonstrates this: assuming a uniform normal distribution for surface patches produces incorrect results. This happens because light reflection in the surface-based reference scene is highly anisotropic—a property that isotropic distributions fail to capture.

This concludes our derivation via non-local surface perturbations. To reinforce these results and gain a deeper understanding:

1. the same transport law can be re-derived by analyzing standard surface derivatives and extending them into space, and
2. the formulation can also be interpreted using the language of random volumes.

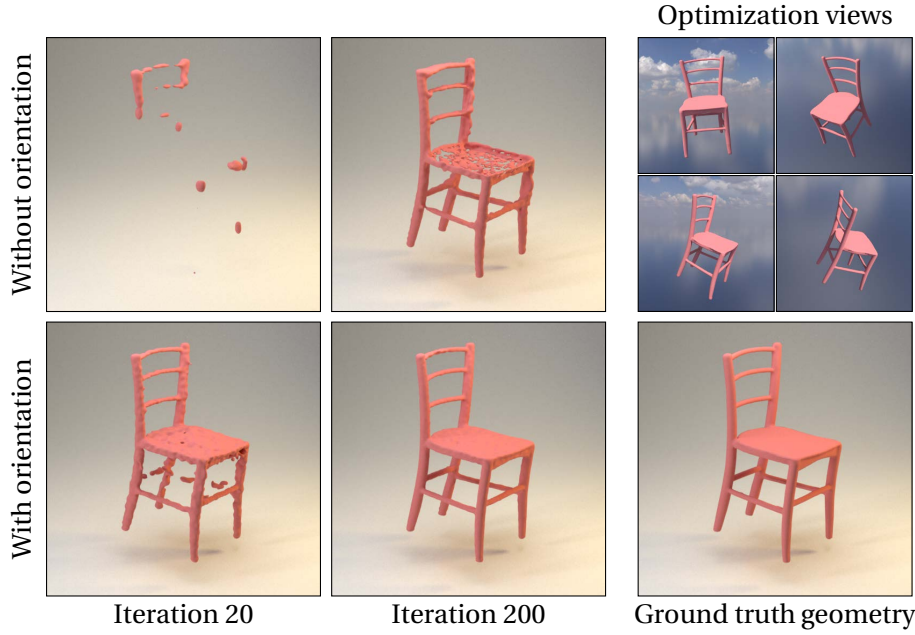


Figure 5.4: **Importance of the orientation field.** We compare reconstructions done *with* and *without* an orientation field β . The top row without β uses an isotropic normal distribution. A lack of orientation information dramatically slows down convergence and produces incorrect meshes.

5.1.3 Primal rendering

Differentiable rendering pipelines normally repeat the cycle of rendering a primal image, differentiating a loss, and then backpropagating derivatives. The previous discussion concerned only derivative propagation, so there is a question of how to generate a primal image of our extended parameter space.

A natural choice is to use only the background surface $\bar{S}$ for primal rendering. (Note that primal images serve solely to compute the adjoint radiance $\frac{\partial \text{loss}}{\partial L_i}$ —we still employ Equation 5.6 for derivative propagation.) Unfortunately, this approach fails catastrophically under the many-worlds formulation: regions beyond the surface $\bar{S}$ are *excluded* from the loss computation yet still receive gradient updates, causing optimization to become unstable, divergent, and effectively random.

We thus use a primal rendering that incorporates both the surface $\bar{S}$ and the distribution S . Recall that the many-worlds principle manifests in Equation 5.5 as a direct *summation* of non-local perturbations. This summation is not directly suitable for primal rendering, as it yields unbounded values. For the experiments in this chapter, we compute an *average* over all perturbations (different from an average over all possible surface renderings) by scaling

Equation 5.5 by a factor of $1/s$:

$$L_i(0) = \frac{1}{s} \int_0^s \left(\alpha(t) L_0^S(t) + [1 - \alpha(t)] L_0^{\bar{S}}(s) \right) dt. \quad (5.8)$$

The scaling is an empirically motivated choice rather than a theoretically unique solution. Our experiments demonstrate that this normalization is straightforward to implement and produces meaningful adjoint radiances ($\frac{\partial \text{loss}}{\partial L_i}$) that enable rapid convergence.

Relation to “radiance field loss” The next chapter [6] instantiates the same many-worlds viewpoint for radiance field reconstruction, specializing it to the simplified setting of pure emission without scattering. This enables several simplifications: (1) due to the absence of BRDF and light integration, noise-free radiance values can be retrieved from an appropriate representation (typically a neural network), (2) ray marching replaces Monte Carlo sampling, and (3) rays are traced from fixed pixel centers, so the pixel footprint integral also disappears.

As a result, there is a 1:1 mapping between surface radiance and reference pixel values, allowing individual losses to be defined for each potential surface, a possibility that is unattainable in our setting. Since radiance contributions from different perturbations are never summed, that formulation sidesteps the scaling factor in Equation 5.8. The present chapter addresses the more difficult nested integral problem inherent in physically based rendering, where such simplifications do not apply.

5.2 Discussion

Pseudocode. Algorithm 5.1 gives one possible implementation of the many-worlds framework in Mitsuba 3 [31]. Algorithm 5.1 uses a variable `mode` to distinguish between the primal rendering pass and the derivative propagation pass.

5.2.1 Relation to surface derivatives

Prior work on geometry differentiation has proposed *local derivative formulations* [Zhang et al. 2020, Zhang et al. 2023] to quantify how small perturbations of a surface affect radiative transport across the entire scene in physical light simulation.

The same transport law can also be re-derived by extending such formulations to measure how tiny changes to *any* hypothetical surface patch within S influence radiative transport. This offers a quantitative way to re-derive the many-worlds derivative transport law using established theory. We made the following observations:

1. The method *behaves correctly* near the surface: its optimization behavior matches surface differentiation algorithms without requiring explicit silhouette sampling.
2. Visibility and shading derivatives are combined into a unified expression in the extended

Algorithm 5.1: Pseudocode of the Many-Worlds primal/backward pass

```

1  mode = "Primal" # Or "Backward"
2
3  def Li(x, ω):
4      # Pick a segment to interact with a surface patch
5      kmw = ⌊kmax * rand()⌋
6      return Li_k(x, ω, kmw, 0)
7
8  def Li_k(x, ω, kmw, k):
9      if k > kmax: # Path length exceeds limit
10         return 0
11
12         # Radiance estimate from background surface
13         s = ray_intersect( $\bar{S}$ , x, ω)
14         x' = x + s * ω # Advance to surface
15         ω', wbrdf = sample_brdf(x', -ω)
16         Lbg = Le(x', -ω) + Li_k(x', ω', kmw, k + 1) * wbrdf
17         if k != kmw: # Segment is before/after the sampled segment
18             return Lbg
19
20         # Radiance estimate from sampled surface patch
21         t, wsurf = sample_surface(rand(), s)
22         if mode == "Primal":
23             weight = wsurf / s # Primal pass: not differentiated
24         else:
25             weight = wsurf # Backward pass: derivative propagation
26         x' = x + t * ω # Advance to sampled surface
27         ω', wbrdf = sample_brdf(x', -ω)
28         occupancy = α(x') # Occupancy at sampled point
29         Lfg = Le(x', -ω) + Li_k(x', ω', kmw, k + 1) * wbrdf
30         return lerp(Lbg, Lfg, occupancy) * weight

```

parameter space. This unification is impossible on the surface $\bar{S}$, because the two derivatives are defined on different domains. Unlike prior work—which requires separate algorithms to compute these two types of derivatives—the formulation here substantially reduces algorithmic complexity.

Relation to sampled-surface alternatives. The same parameterization could also be used in a more direct stochastic way: sample a surface realization from S , render it with a differentiable surface renderer, and backpropagate the loss to the parameters of S through the sampling process. This pathwise estimator over random surface realizations is conceptually simple, but it has two drawbacks for the purposes of this chapter. Each optimization step would see only a small number of realized surfaces, so the gradient support would again be sparse unless many realizations were rendered. The renderer would also still have to handle the

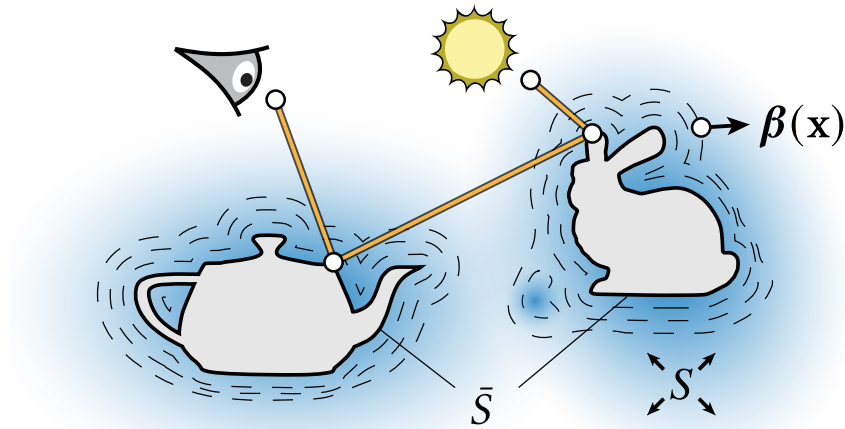


Figure 5.5: **Many-worlds representation.** The method refines a surface $\bar{S}$ by optimizing a distribution S of possible surfaces. Each hypothetical surface patch drawn from S is independently adjusted to improve the matching between $\bar{S}$ and the target image. To define the behavior of possible surfaces in space, S is parameterized by an occupancy field $\alpha(\mathbf{x})$ and an orientation field $\beta(\mathbf{x})$.

visibility discontinuities of each sampled surface. Many-worlds transport instead evaluates many hypothetical patches as competing local perturbations in one pass, avoiding a Monte Carlo outer loop over full surface realizations.

5.2.2 Relation to volume rendering

At a high level, both inverse volume rendering [61] and the many-worlds method can be interpreted as optimizing *a distribution of surfaces*. However, the two approaches differ fundamentally in how they interact with this distribution: when multiple potential surfaces exist along a ray, how do we model their interplay?

- Exponential volume rendering treats interactions as statistically independent events, leading to a memoryless Poisson process in which multiple interactions occur and are weighted by relative occlusion probabilities [62].
- The many-worlds method treats interactions as mutually exclusive events, ensuring potential surfaces along a ray do not interact via shadowing or scattering.

This distinction also clarifies the relation to Gaussian-process implicit surfaces (GPIS) and stochastic-surface renderers [21, 22, 24]. Those methods define a correlated distribution over implicit surfaces and then render statistics of that random geometry. A differentiable GPIS renderer would optimize the parameters of the stochastic surface model itself, with realizations interacting through the chosen image-formation model. Many-worlds instead uses the distribution as an optimization scaffold: hypothetical surface patches provide non-local derivative support, but the target remains the background surface $\bar{S}$, and the hypotheses

are not rendered as an interacting stochastic solid.

Consider a distribution S modeling a nearly empty scene containing a single infinite plane with an uncertain offset. Within any realization of this distribution, light traveling toward this plane scatters *exactly once*. In contrast, the volume model diffuses the ray-surface interaction into a band of microflakes that cause *arbitrarily long* scattering chains. This significantly increases computational cost and obscures the optical interpretation of these interactions: the effect of multiple scattering must later be approximated as a surface BRDF, a process that generally has no exact solution and must rely on approximations.

A traditional inverse rendering pipeline (which renders the scene to compute a loss) breaks down under the mutually exclusive assumption, because it would render each potential surface as a separate scene, thereby unrealistically requiring *every* potential surface to match the reference image. Instead, we optimize potential surfaces by refining the rendering of a background surface $\tilde{S}$, giving the algorithm a clear convergence target (Figure 5.5). Such comparative adjustments lead to the notion of *non-local surface perturbations*, and simultaneous optimization of all perturbations leads to our *many-worlds* derivative transport.



Figure 5.6: **Multi-view geometry reconstruction.** For the DEER scene, all 8 views are behind the object, and only the front side is visible in the mirror. For POLYHEDRA, NEPTUNE, and FERTILITY, the object is inside a smooth spherical or cubic glass container. The materials are known during optimization: the DRAGON has a rough gold material, POLYHEDRA and NEPTUNE are made of copper oxide, and the others are diffuse.

5.3 Results

The evaluation in this chapter focuses on end-to-end inverse-rendering behavior rather than single-parameter forward-derivative plots. Such plots are not especially informative here: the method computes an extended derivative on a higher-dimensional domain, so there is no direct one-to-one comparison with conventional surface derivatives. Moreover, the main question is not whether the local differentiation machinery is correct in isolation, but whether the extended derivative domain improves optimization from weak initialization.

All experiments in this chapter use many-worlds derivatives exclusively and *explicitly disable surface derivatives* on S , even for the albedo optimization demonstrated in Figure 5.7.

5.3.1 Core reconstruction behavior

Multi-view reconstructions. Figure 5.6 presents multi-view reconstructions of objects with known material properties in various settings using many-worlds optimization. All experiments use the Adam optimizer [64]. Disabling momentum in the early iterations helped avoid excessive changes while the occupancy field was still very far from convergence. Alternatively, a stochastic gradient descent optimizer produces comparable results. During reconstruction, each view is rendered at a resolution of 512×512 .

The last four rows of Figure 5.6 show reconstructions involving perfectly specular surfaces. No prior PBR method could handle such scenes: reparameterization-based methods would need to account for the extra distortion produced by specular interactions, while silhouette segment sampling methods would need to find directional emitters through specular chains. Both are complex additional requirements that would be difficult to solve in practice; with the many-worlds formulation, the problem simply disappears.

Albedo reconstruction. This chapter primarily focuses on geometry, but the same formulation can also be used to optimize materials or lighting. Figure 5.7 presents experiments where we jointly optimize the geometry and albedo texture of an object. We store this spatially varying albedo in an additional 3D volume that parameterizes the BSDF of the many-worlds representation.

Benefits of assumption-free geometry priors. Figure 5.9 compares many-worlds optimization to a technique that evolves an SDF using reparameterizations [63]. These experiments demonstrate that the method requires significantly fewer iterations because new surfaces can be materialized from the very first iteration.

We use 20 random optimization views, with each view rendered at 256×256 pixels. Both methods use a geometry grid resolution of 128^3 . The time required to perform one iteration of the optimization for one view is 0.25 seconds for the many-worlds method, 0.38 seconds for

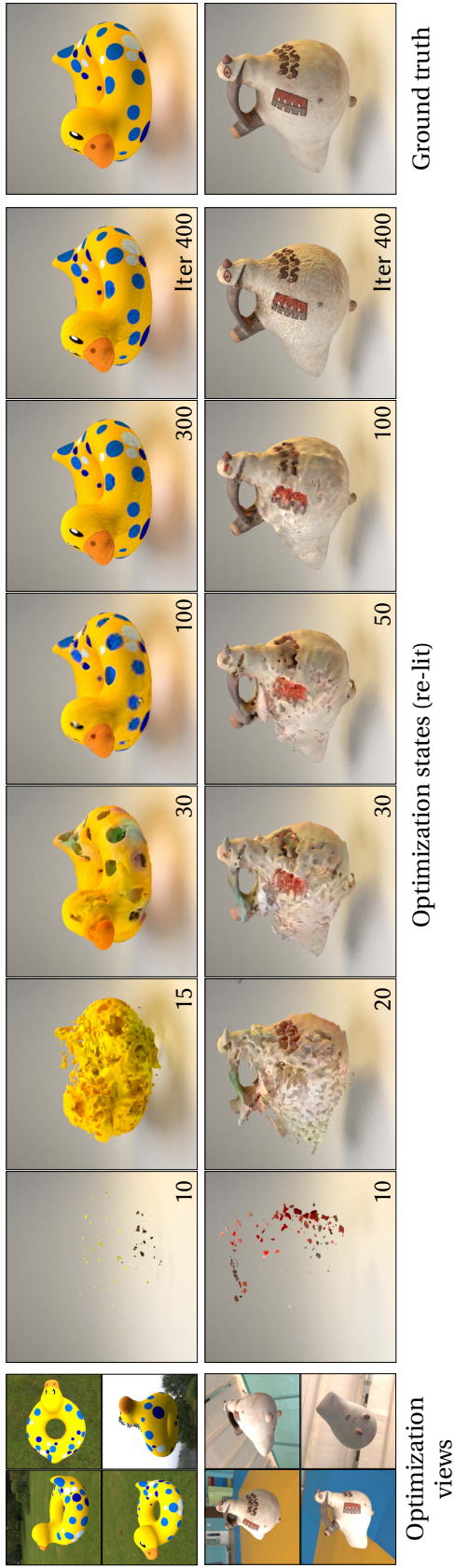


Figure 5.7: **Material optimization.** This experiment demonstrates joint optimization of geometry and an albedo texture. The method focuses on geometric optimization, but it is compatible with more general inverse rendering pipelines that also target materials and lighting.

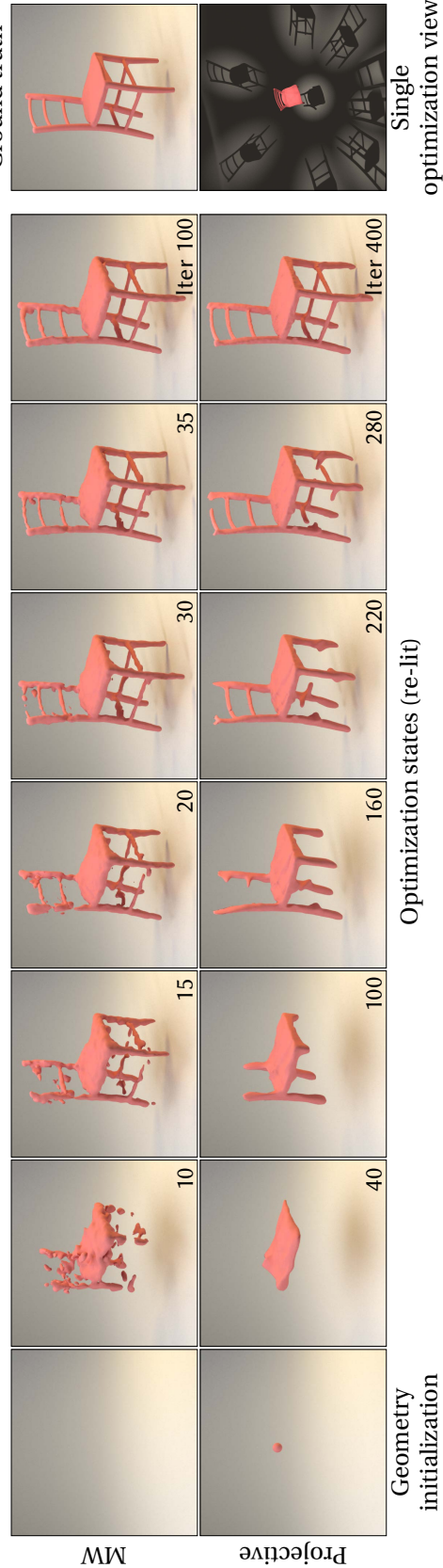


Figure 5.8: **All-position optimization.** This experiment compares the method to the local surface evolution method from chapter 4. Unlike the strictly local progressive extrusion seen earlier, the many-worlds method optimizes all positions in space simultaneously, enabling faster convergence.

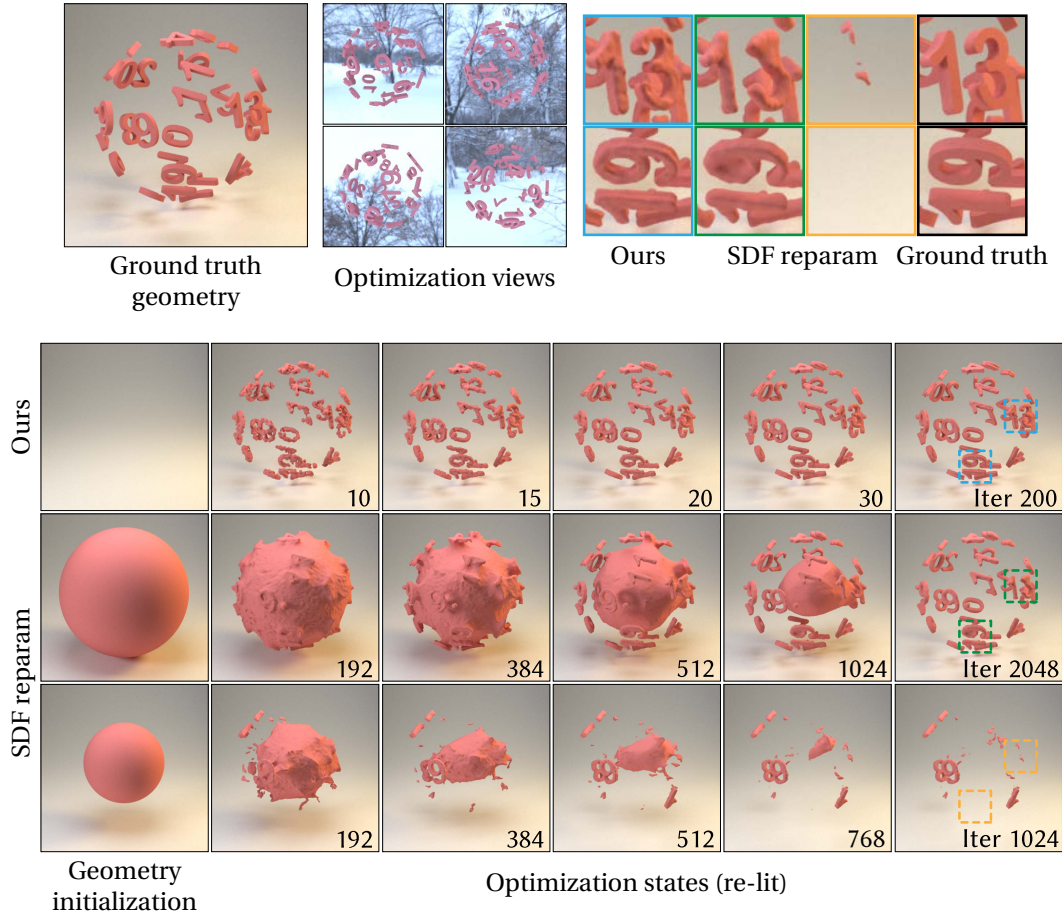


Figure 5.9: **Assumption-free optimization.** Prior surface optimization methods require an initial guess, which affects the quality of their output. The figure compares the many-worlds method to an unbiased surface derivative method (SDF reparameterization [63]) using a sphere initialization that is either *bigger* (middle) or *smaller* (bottom) than the overall size of the target. The baseline struggles to reconstruct the target from the smaller sphere, while carving it out of the bigger sphere seems more reliable. The many-worlds method (top) can optimize without such assumptions, i.e., starting from empty space.

the large sphere initialization, and 0.32 seconds for the small sphere initialization. The many-worlds method benefits from the efficiency of ray-triangle intersections, whereas the SDF representation relies on a more costly iterative sphere tracing algorithm. These measurements also show how sphere tracing slows down with smaller steps due to complicated geometry along a ray, as evidenced by the slower performance of the larger sphere initialization.

Optimizing all positions at once Figure 5.8 revisits the chair reconstruction experiment of chapter 4 to demonstrate the benefits of the extended parameter space.

There, the baseline employed preconditioned gradient descent [54] to locally deform a triangle mesh. Because the projective sampling approach evaluates strictly local boundary derivatives,

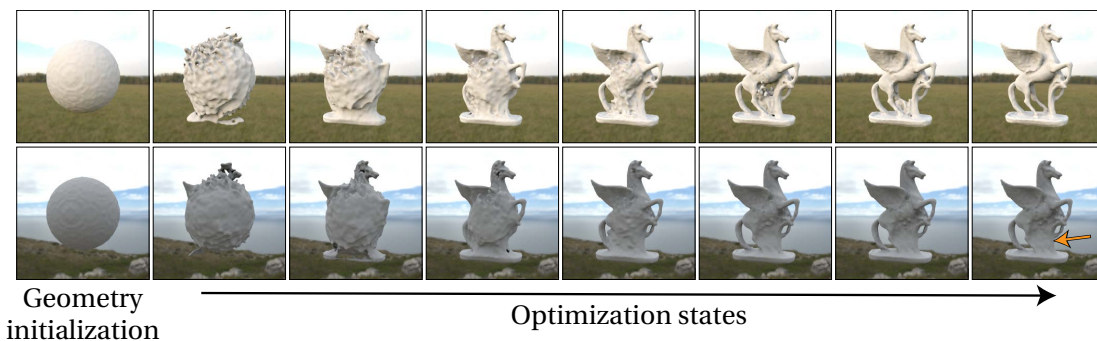


Figure 5.10: **Interior topology changes.** The method extends surface perturbations only to the exterior and does not accommodate interior topological changes — such as transforming a sphere into a donut. We demonstrate this using a sphere initialization under two lighting conditions. While shading derivatives can sometimes produce correct holes (top row), they can fail under alternative lighting conditions (bottom row).

the pixels that render the legs propagate gradients to the scene only in a thin region of overlap between the tentative object and the chair shadows in the reference image. This local gradient signal leads to the progressive extrusion of the legs seen in Figure 4.14. In contrast, because many-worlds observes and uses gradients from perturbations extending far into the empty space, it captures derivative signals outside the narrow region of initial overlap. As a result, all parts of the surface receive updates starting from the very first iteration, enabling much faster global convergence.

5.3.2 Limitations and robustness

Interior topological changes. Figure 5.10 illustrates a limitation of the method: it does not robustly handle interior topological changes. This issue arises because many-worlds derivatives extend only to the exterior of $\bar{S}$, leaving the interior unsupervised, much like traditional surface evolution methods. When attempting to create a hole, we rely on shading derivatives to bend the surface inward, which sometimes produces a hole as desired (top row). However, this type of optimization is sensitive to lighting conditions; as shown in the bottom row, different lighting can cause the optimization to stall.

To address this limitation, Mehta et al. [65] proposed explicitly testing whether creating a cone-shaped hole is beneficial, and Zhang et al. [6] suggested sampling the background surface stochastically to occasionally permit visibility through high-occupancy regions. We leave the exploration of these extensions for future work.

The sphere initialization is used solely to demonstrate this limitation; with an assumption-free initialization, the method converges correctly and more quickly in both lighting conditions.

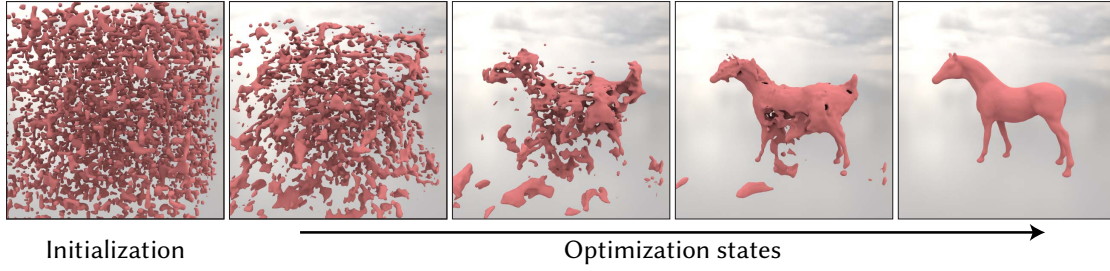


Figure 5.11: **Subtractive changes.** Many-worlds derivatives match surface derivatives when sampled close to the surface. We initialize the scene with dense geometry to demonstrate the robustness of the method against subtractive changes.

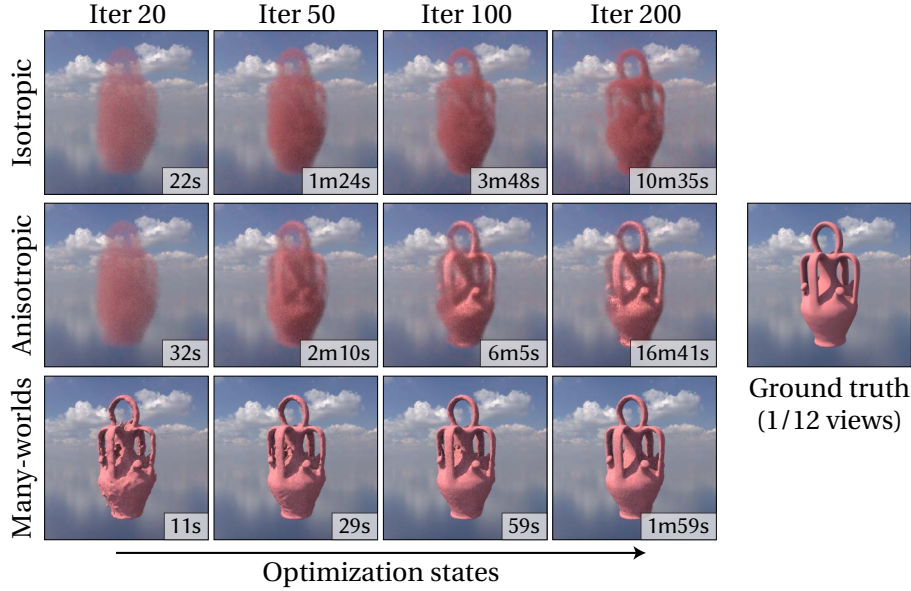


Figure 5.12: **Comparison with volume reconstructions.** The figure compares reconstruction results using exponential volume inversion and the many-worlds approach. Isotropic volumes (top) cannot replicate the surface appearance, while anisotropic micro-flakes (middle) perform somewhat better. However, simulating interactions between worlds (e.g. to model exponential attenuation) introduces substantial computational overhead in both volume reconstruction methods. Extracting the final BSDF and surface from the volume poses additional challenges. The many-worlds method (bottom) produces a high-fidelity surface reconstruction in a fraction of the time.

Subtractive changes Figure 5.11 shows the optimization states for an initialization with random geometry. We used 16 optimization views and the same setup as in Figure 5.6. Since the many-worlds derivative extends the surface derivative domain without approximations, the method naturally addresses scenarios that surface derivatives can handle—in this case, removing superfluous geometry and deforming the rest to reconstruct the desired object.

Table 5.1: We measure the average time needed to optimize one 512^2 pixel image for different scenes. The reported time covers all overheads, including the primal rendering pass, the uncorrelated derivative propagation pass, the optimizer step, and the scene update.

	Path depth	AD depth	spp	grad spp	time (s)
HEPTOROID	2	2	128	32	0.41
DRAGON	2	2	128	32	0.42
DEER	4	2	128	32	0.75
POLYHEDRA	4	2	64	32	1.10
NEPTUNE	5	3	256	32	2.56
FERTILITY	5	3	256	32	2.06

Comparison with volume reconstructions Figure 5.12 presents results and equal-iteration timings comparing the many-worlds formulation to volume-based inversion. For the latter, we set the maximum path depth of the underlying volumetric path tracer to 3 interactions, as larger values significantly degrade performance without improving visual fidelity. All experiments use the same number of samples per pixel, and each optimization iteration uses all 12 optimization views. The anisotropic volume model uses the SGGX phase function [18]. The volume models are initially as fast as ours but slow down significantly as the volume thickens, which is a consequence of iterative steps needed to resolve coupling between different parts of the exponential volume. In addition to reducing speed, this coupling leads to degraded reconstruction quality at an equal iteration count. The many-worlds approach is algorithmically simpler, produces a better result in less time, and directly outputs a mesh with materials that are ready to be relit, without the need for additional optimization to extract a surface BRDF from phase functions.

Timings. Table 5.1 lists the average computation time per gradient step and view for several scenes. We limited the maximum path depth for every scene to a reasonable value. For certain scenes, we also disabled gradient estimation for some path segments. For example, in the shape reconstructions inside a glass object, the first ray segment (from the camera to the glass interface) and the last ray segment cannot intersect the many-worlds representation.

5.4 Conclusion

The end of Chapter 3 identified a second bottleneck beyond derivative correctness: even unbiased surface derivatives remain sparse and local, which makes physically based geometry optimization brittle. This chapter addressed that bottleneck by extending surface perturbations into space while keeping the physically based transport model. Using non-local perturbations of a surface, many-worlds inverse rendering can synthesize complex geometries from an initially empty scene.

Chapter 5. Many-Worlds Inverse Rendering

The formulation developed here lays the theoretical foundation and validates it with an initial implementation. However, this implementation is still far from optimal and could benefit from various enhancements. For example, the implementation derives the local orientation from a field that also governs occupancy, which is straightforward but also introduces a difficult-to-optimize nonlinear coupling. Extending the model with a distribution of orientations would also allow it to consider multiple conflicting explanations at every point.

Reconstructing an object involves a balance between *exploration* to consider alternative explanations in S and *exploitation* to refine parameters of the current explanation $\tilde{S}$. The current implementation aggressively pursues the exploration phase using a uniform sampling strategy but lacks a mechanism to exploit that knowledge effectively. In practice, it often reconstructs a good approximation of a complex shape in as little as 20 iterations, but then requires 500 iterations for the seemingly trivial task of smoothing out small kinks.

The current implementation of the many-worlds derivative is based on a standard physically based path tracer, but previous works on differentiable rendering have shown the benefits of moving derivative computation into a separate phase using a *local formulation*. This phase starts the Monte Carlo sampling process where derivatives locally emerge (e.g., at edges of a triangle mesh in the case of visibility discontinuities). Integrating the local form of our model with occupancy-based sampling could refine the background surface with a more targeted optimization of its close neighborhood.

Within the broader thesis, this chapter is the physically based instance of the distribution-over-surfaces viewpoint. It is most useful when geometry must be discovered from weak initialization, when visibility derivatives are too sparse to guide ordinary surface evolution, or when a fast exploratory phase is more important than final local polish. Ordinary inverse surface rendering remains preferable when a good initialization is available, when the surface parameterization is fixed by the application, or when the main task is high-quality local refinement of an existing mesh. The trade-off is therefore representational: many-worlds enables efficient non-local recovery, but the current parameterization also constrains the kinds of surfaces and local refinements it can represent. The next chapter revisits the same core idea in an emissive setting, where the transport becomes simpler and the representation can be supervised more directly.

6 Radiance Surfaces: Surface Optimization with a 5D Radiance Field Loss

The previous chapter introduced the many-worlds viewpoint in the physically based setting: optimize a distribution of surface hypotheses rather than evolving a single surface directly. That formulation addressed the surface–volume trade-off identified in the thesis introduction, but it did so in the presence of visibility discontinuities, Monte Carlo noise, and recursive transport.

This chapter studies the same representation question in the simpler emissive setting, where scene points store or predict outgoing radiance directly rather than deriving it through recursive physically based light transport. It asks whether the distribution-over-surfaces viewpoint can retain much of the robustness of volumetric optimization while still targeting an explicit surface output. In that sense, this chapter is the emissive counterpart to *Many-Worlds Inverse Rendering*.

The key change is where supervision is applied. Instead of blending colors along a ray and then applying an image-space loss, the method supervises candidate surfaces directly in 5D spatio-directional space and only then aggregates those losses. Sampled points along a ray therefore act as independent surface candidates rather than contributors to a single semi-transparent volume.

This chapter therefore serves two purposes in the dissertation. First, it shows that the “distribution over surfaces” viewpoint is not tied to physically based transport; in the emissive setting, the same idea leads to a particularly simple formulation. Second, it identifies a practical midpoint in the thesis design space: much of the speed and implementation simplicity of NeRF-style methods can be retained without committing to a purely volumetric geometric model.

The chapter first derives this formulation from a surface-optimization viewpoint, then introduces the stochastic background and optional relaxation used in practice, and finally evaluates it on both view-synthesis and geometry reconstruction tasks.

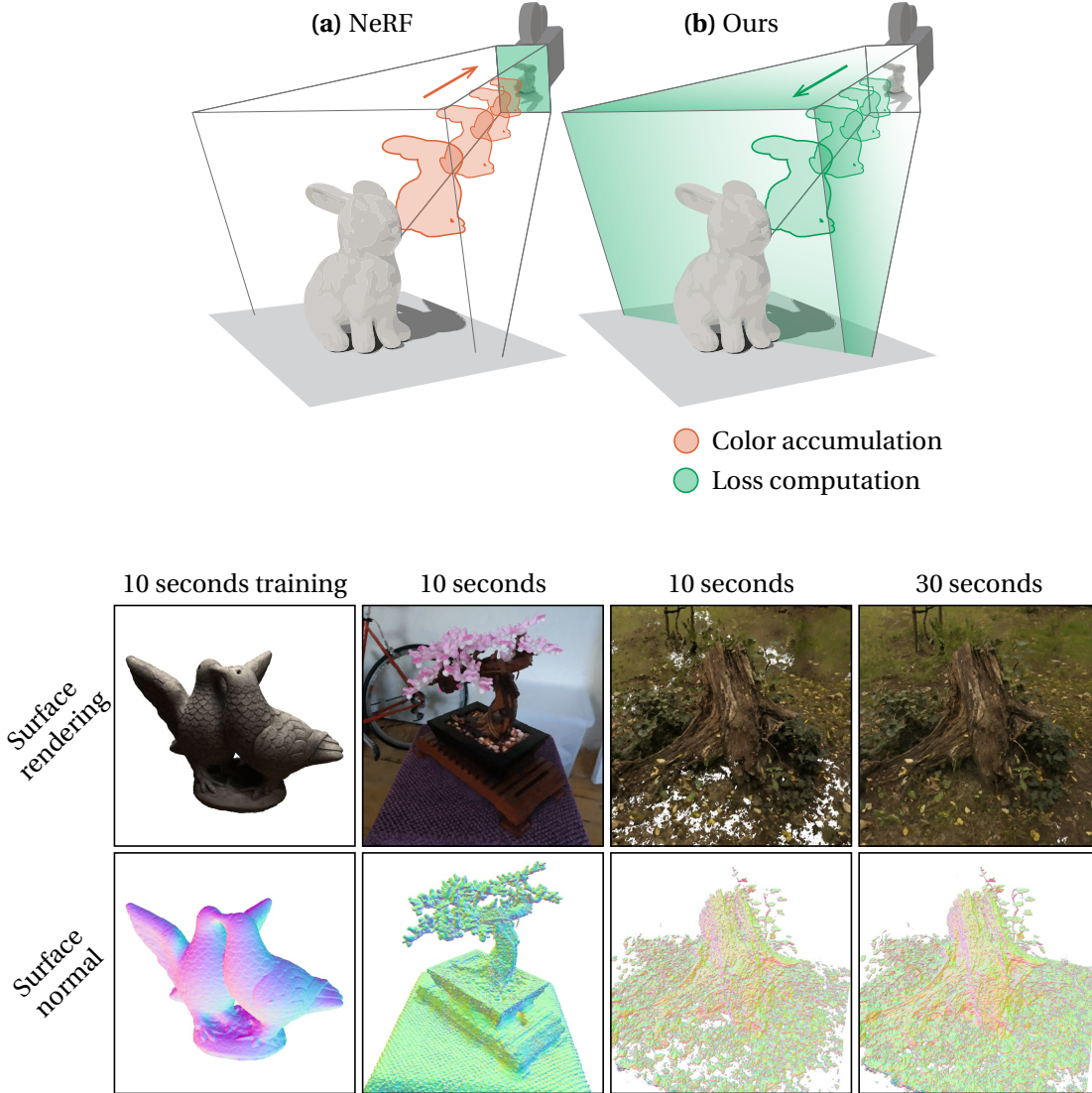


Figure 6.1: **Radiance surfaces.** The method reconstructs explicit surfaces while retaining much of the speed and robustness of NeRF-style optimization. **Top:** In contrast to volume-based methods that minimize 2D image losses, as shown in (a), we adopt a spatio-directional radiance field loss formulation, as shown in (b). At each step, the method considers a distribution of optically independent surfaces, increasing the confidence of candidates that agree with the reference imagery. **Bottom:** A meaningful surface can be extracted at any iteration during optimization.

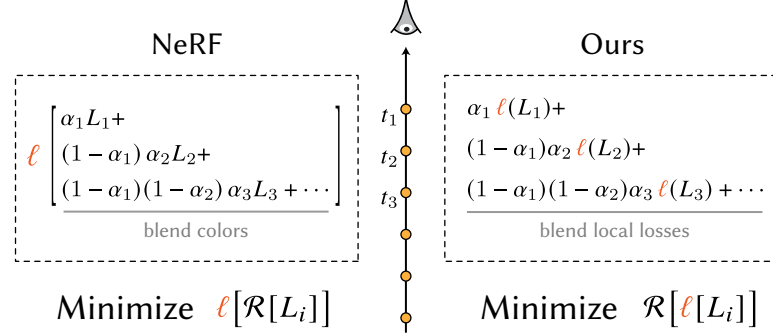


Figure 6.2: **Image-space loss versus radiance field loss.** In volumetric reconstruction, colors are blended first and the loss is applied in image space. In this chapter, we blend local losses defined on candidate surfaces, which yields a distribution over surfaces from which an explicit surface can be extracted.

6.1 Introduction

In the emissive setting considered here, the central trade-off is the same one emphasized throughout the thesis: explicit surfaces are usually the desired output, but volume-like representations are much easier to optimize. For reconstruction from photographs, surfaces remain attractive because they are easier to edit, animate, and render efficiently. Yet direct optimization of a differentially rendered surface is often brittle: the loss landscape is highly non-convex, and the resulting gradients tend to support only local geometric changes.

Volumetric methods [2, 3] avoid much of this brittleness. Their continuous representations are easier to differentiate and typically induce smoother optimization behavior. The cost is that geometry is recovered only indirectly, often requiring additional surface-promoting regularizers [66] or multi-stage extraction procedures [67].

This chapter asks whether it is possible to retain much of that optimization behavior without giving up a surface target. Following the distribution-over-surfaces viewpoint introduced earlier in the thesis, the method optimizes a *distribution over surfaces* rather than a single evolving surface. Concretely, training photographs are projected into the scene, and the method minimizes the attenuated difference between the resulting light field and the spatial-directional emission originating from the surface distribution.

The key change relative to standard volumetric training is where the loss is applied. Under the resulting *radiance field loss*, each point along a ray is treated as an independent surface candidate and optimized to match that ray’s pixel color. Points along the same ray can therefore receive independent gradients, allowing the color or density to increase at one point and decrease at another. This differs from volumetric reconstruction, where color is integrated along the ray before the loss is computed (see Figure 6.1, left). In that setting, if the integrated color is too dark or too bright, all points along the ray receive gradients with the same sign, leading to correlated adjustments.

This radiance field loss gives rise to equations that closely resemble those of volumetric reconstruction methods (see Figure 6.2). In practical terms, this means that the method is simple to integrate into existing volumetric frameworks, while still encouraging convergence to an explicit surface rather than a semi-transparent volume. While this chapter does not focus on metric comparisons, the Instant NGP [68] implementation requires modifying only a few lines of the core algorithm. It runs at roughly the same speed (in terms of PSNR vs. time) and produces surfaces whose PSNR is, on average, only 0.1 dB lower than that of the volumetric baseline.

The remainder of the chapter first derives the method from non-local surface perturbations, then develops the radiance field loss and its practical variants, and finally evaluates the resulting representation in both novel view synthesis and geometry reconstruction settings.

6.2 Method

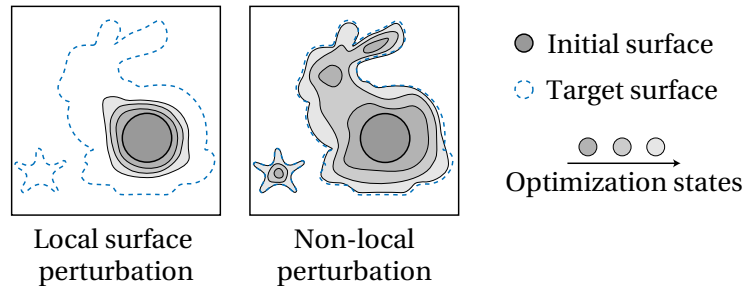
In this section, we derive our radiance field loss (Figure 6.2) by progressively transforming the optimization of a single evolving surface. While the final result resembles volumetric reconstruction, this progression demonstrates that the method’s origins are surface-based.

6.2.1 Non-local surface perturbation

Differentiating a rendering with respect to geometry reveals how small geometric perturbations affect the resulting image. However, because these derivatives are only nonzero on the surfaces themselves, they tend to cause convergence issues when used for optimization.

To overcome this limitation, consider the effect of introducing a small surface patch at some distance *above* an existing visible surface. This modification also affects the rendered image and can be interpreted as a perturbation of a more general *non-local* derivative. A similar concept was previously used by Mehta et al. [65] to *nucleate* new shapes in 2D vector graphics, and by Zhang et al. [5] in the context of physically based rendering.

Optimizing surfaces over this extended domain mitigates two key issues discussed previously. First, because updates are no longer constrained to the surface, the algorithm can achieve faster and more robust convergence within a higher-dimensional loss landscape, as illustrated below:



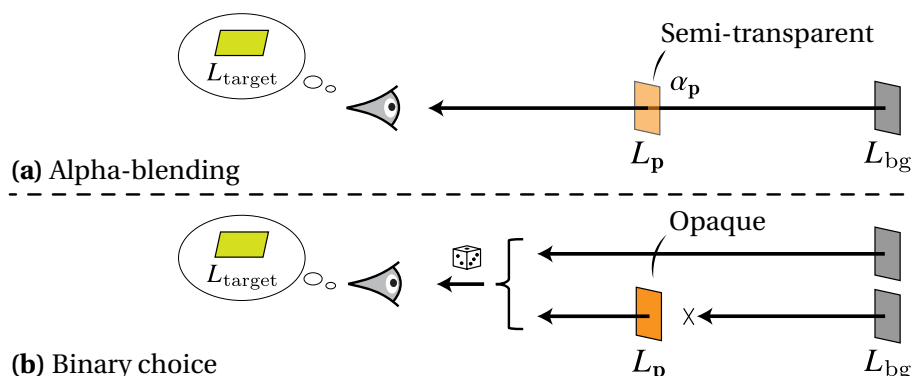


Figure 6.3: **Non-local perturbations.** We consider a single candidate surface patch (with color L_p) along the ray as a perturbation of a background surface (with color L_b). **(a)** Blending colors violates the surface assumption and leads to volumetric results. **(b)** We instead treat the perturbation as a random binary choice and optimize the associated discrete probability. The final reconstruction is non-random and will never blend contributions from multiple surfaces.

Second, the need for complex, specialized methods to estimate boundary derivatives is eliminated, which simplifies the implementation and further improves performance. Before making these abstract notions concrete, we specify the geometric representation used here.

Geometric representation Non-local perturbations require a representation that spans the entire space. To this end, we use an *occupancy field* [59, 60] that encodes the discrete probability of a position $\mathbf{x}$ being occupied:

$$\alpha(\mathbf{x}) = \Pr\{\mathbf{x} \text{ lies within an object}\} \in [0, 1].$$

After convergence, the field is expected to have occupancy values approaching 1 on the surface and 0 in the exterior. The choice of an occupancy field is somewhat arbitrary; the primary focus here is on optimizing geometry regardless of the specific details of the representation.

6.2.2 Radiance field loss

Single candidate To explain the concept of a non-local perturbation, we first focus on the case of a single candidate surface patch along a ray. Figure 6.3 depicts this setup, in which a candidate at position $\mathbf{p}$ with color L_p and occupancy α_p precedes a background with color L_b . Here, the term *background* can refer to a surface, an environment map, etc.; later sections provide a concrete definition. We discuss how this geometric configuration arises later—for now, we assume that it is given and that the color values L_p and L_b are fixed.

In this case, the optimal reconstruction is straightforward: the candidate should be created if it improves the match with respect to a specified target color L_{target} ; otherwise, it should be discarded.

The occupancy parameter $\alpha_{\mathbf{p}}$ provides the means to achieve this outcome. However, there are different ways to incorporate it. The standard volumetric approach (Figure 6.3a) interprets $\alpha_{\mathbf{p}}$ as an *opacity* for alpha-compositing, minimizing a color difference $\ell(\hat{L}, L)$ of the form

$$\ell(\alpha_{\mathbf{p}} L_{\mathbf{p}} + (1 - \alpha_{\mathbf{p}}) L_{\mathbf{b}}, L_{\text{target}}). \quad (6.1)$$

The fundamental limitation of this approach is its inability to promote binary occupancy values. When the best match is given by a blend of $L_{\mathbf{p}}$ and $L_{\mathbf{b}}$, the loss will reach zero without forming a distinct surface. A common remedy involves adding loss terms to penalize such behavior, but this lacks a principled theoretical foundation and adds complexity in the form of hyperparameters.

We instead interpret the non-local perturbation as a binary choice: the candidate surface either exists or it does not. Thus, the final color value associated with the ray is either that of the candidate $L_{\mathbf{p}}$ or the background $L_{\mathbf{b}}$ (Figure 6.3b). We quantify the quality of each possibility via ℓ and seek the occupancy value $\alpha_{\mathbf{p}} \in [0, 1]$ that minimizes the following objective:

$$\mathcal{L}(\mathbf{p}) = \alpha_{\mathbf{p}} \ell(L_{\mathbf{p}}, L_{\text{target}}) + (1 - \alpha_{\mathbf{p}}) \ell(L_{\mathbf{b}}, L_{\text{target}}). \quad (6.2)$$

By blending the losses of the two surfaces instead of their colors, this approach selects the surface that best explains the target color. For fixed colors, Equation (6.2) is linear in $\alpha_{\mathbf{p}}$: the optimum is $\alpha_{\mathbf{p}} = 1$ when the candidate loss is lower than the background loss, and $\alpha_{\mathbf{p}} = 0$ otherwise. The all-empty solution is therefore not favored by this local subproblem unless the background already explains the observation better. In the full method, colors and occupancies are learned together, so this local comparison is coupled across views and through the stochastic background construction introduced below.

The simplified example shown here assumes that the candidate color $L_{\mathbf{p}}$ is static. In practice, $L_{\mathbf{p}}$ (but not $L_{\mathbf{b}}$) is also subject to optimization, which requires multiple viewpoints to resolve ambiguity, as discussed later.

Multiple candidates We now extend the loss formulation to consider multiple candidates. This is advantageous because it allows our method to evaluate the effects of several perturbations, which in turn accelerates convergence.

The key property of the single-candidate loss formulation is that it isolates the candidate from the background surface (i.e., by considering either the candidate or the background). The generalization to multiple candidates preserves this property by treating each candidate as an independent subproblem (Figure 6.4) and minimizing the sum of their respective losses:

$$\mathcal{L}_{\text{ray}}(\mathbf{r}) = \sum_{i=1}^m \mathcal{L}(\mathbf{p}_i), \quad (6.3)$$

where $\mathcal{L}(\mathbf{p}_i)$ (following Equation 6.2) represents the loss of the i -th of m candidates sampled

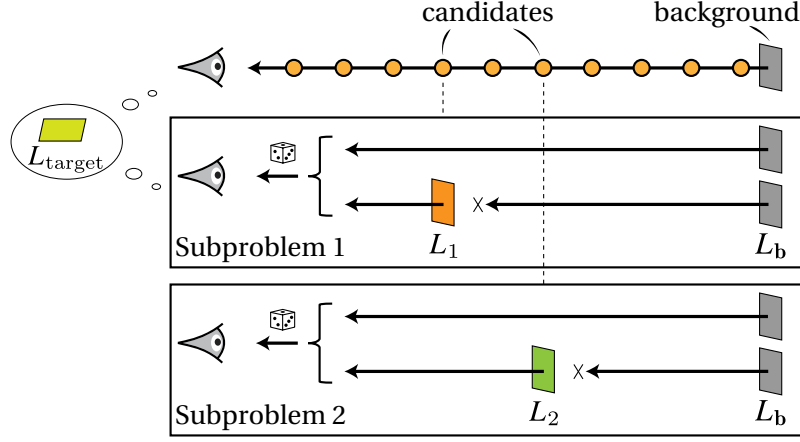


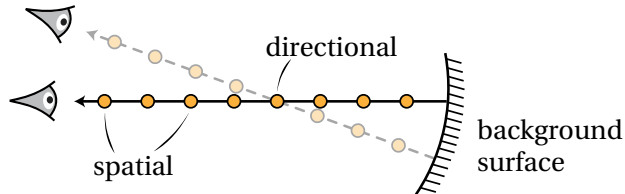
Figure 6.4: **Surface candidates as independent subproblems.** With multiple candidates along a ray, each perturbation is treated as an independent subproblem, resulting in *local* losses distributed spatially over the scene.

along the ray $\mathbf{r}$.

Spatio-directional loss Reconstruction tasks evaluate the loss (6.3) along a large set of rays $\mathbf{r}_k$ ($k = 1, \dots, n$), where n denotes the total number of pixels across all reference images. This further expands the set of independently considered candidate surfaces and leads to the combined loss

$$\mathcal{L}_{\text{total}} = \sum_{k=1}^n \mathcal{L}_{\text{ray}}(\mathbf{r}_k). \quad (6.4)$$

Whereas conventional surface optimization only propagates gradients to the surface itself, the use of $\alpha_{\mathbf{p}}$ and $L_{\mathbf{p}}$ in Equation (6.2) covers the *entire* observed 3D space. For positions viewed from multiple directions, the loss also generally varies with respect to direction:



In other words, by moving the evaluation of ℓ from image space into the scene, we have created a spatio-directional *radiance field loss*.

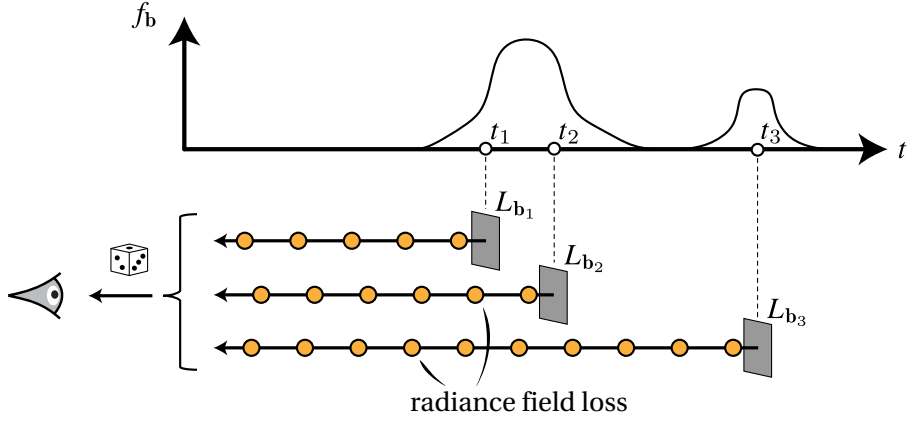


Figure 6.5: **Stochastic background.** Selecting the background surface at random from a distribution f_b enables visibility through high-occupancy regions. Each sampled background surface defines a new perturbation problem solvable with the radiance field loss. Taking an expectation over this process leads to a simple deterministic expression that we implement in practice.

6.2.3 Stochastic background surface

To complete our derivation of the loss function, we still need to define the background surface. Rather than a deterministic surface (e.g., a level set of the occupancy field), we draw the background from a per-ray distribution f_b . This enables occasional “visibility” through high-occupancy regions, allowing occluded objects to be considered as the background (Figure 6.5). Crucially, this supports complex topological changes in our optimization without having to explicitly account for them [65]; see section B.3 for additional details.

The design of the distribution f_b is flexible. One straightforward approach is to prioritize sampling in high-occupancy regions, as these areas are more likely to correspond to surfaces. During ray traversal, we stochastically decide whether to use a position as the background surface based on its occupancy value. This sequential decision process reflects the concept of free-flight distance [16] and forms the *free-flight background distribution*.

We can analytically formulate the expectation of sampling the background surface from such a free-flight distribution and derive a corresponding aggregated local loss that is analogous to classical volumetric light transport:

$$\mathcal{L}(\mathbf{p}_i) = \left(\prod_{j=1}^{i-1} (1 - \alpha_{\mathbf{p}_j}) \right) \alpha_{\mathbf{p}_i} \ell(L_{\mathbf{p}_i}), \quad (6.5)$$

which, when substituted into Equation (6.4), yields the radiance field loss (Figure 6.2). See section B.1 for the complete derivation.

6.2.4 Volume relaxation

We also consider an optional heuristic generalization of the method. It is orthogonal to the main formulation and can be enabled during training.

While a surface representation offers many advantages, the opaque surface assumption has inherent limitations in certain scenarios. For example, sub-pixel structures are challenging to model with geometry, and a single surface may fail to accurately represent the appearance of directionally varying materials. In these regions, a volumetric representation is more suitable.

Our goal is to relax our method so that it reconstructs most of the scene as surfaces (regions where low loss can be reached) and uses volumetric representations only in the remaining challenging regions. To this end, we first train with our algorithm for 20k iterations to obtain an initial surface representation. We then identify challenging regions by evaluating where local losses remain high. In subsequent training steps, we relax the surface assumption, allowing volumetric alpha blending in these regions.

After training, rather than extracting a surface, we render the scene volumetrically, with surface regions treated as fully opaque “volumes”. Compared with a volumetric scene optimized with NeRF, our method still benefits from the compact representation of surface regions. When accumulating colors along a ray, very few samples are required to saturate the transmittance, leading to faster inference and lower computational resource use during training.

Unless stated otherwise, all results in this chapter (marked as “ours”) are trained *without* volume relaxation.

6.2.5 Implementation

Our loss function can be implemented to resemble the color blending structure of standard volumetric reconstruction methods such as NeRF [2]. As such, it is straightforward to implement in existing codebases, as illustrated in the following comparison of pseudocode.

NeRF	Ours
<pre> L_target, ray = sample_ray() t_array = sample_positions(ray) color = 0 for t in reverse(t_array): alpha, L = eval_model(ray, t) # Blend colors color = (1 - alpha) * color + alpha * L loss = l(color, L_target) loss.backward() </pre>	<pre> L_target, ray = sample_ray() t_array = sample_positions(ray) loss = 0 for t in reverse(t_array): alpha, L = eval_model(ray, t) local_loss = l(L, L_target) # Blend local losses loss = (1 - alpha) * loss + alpha * local_loss loss.backward() </pre>

This resemblance also suggests that our method has an optimization landscape similar to

that of NeRF and inherits its robustness. However, while NeRF’s loss supervises all samples along the ray to collectively match the target color, our loss aims for each sample to match the target color independently or to become transparent when the background is a better match. This distinction fundamentally defines our approach as a surface reconstruction algorithm. For the same reason, the implementation-oriented expression should not be interpreted as treating empty space as a standalone explanation. Transparency is preferred only relative to a background sample that must itself explain the same ray. If the currently sampled surfaces do not explain the observations across views, subsequent samples and views continue to create local losses at other candidate locations.

6.3 Results

6.3.1 Novel view synthesis

Visual quality Despite surfaces’ inherently fewer degrees of freedom, Figure 6.10 and Figure 6.11 show that our method achieves results that are qualitatively comparable to NeRF. We also visualize the surface renderings at occupancy level sets $\{0.01, 0.1, 0.5, 0.9, 0.99\}$. Renderings of the scene optimized by our algorithm barely change, indicating a near-Heaviside step function in the occupancy field. In contrast, NeRF’s inherently volumetric representation does not yield meaningful visualizations for these level sets. Figure 6.6 highlights the reconstruction of another scene where our method with volume relaxation addresses the challenge of modeling a semi-transparent object.

Table 6.1 shows that our method achieves visual quality comparable to exponential volume reconstruction (NeRF) when trained on the MipNeRF360 dataset, using default Instant NGP hyperparameters, despite using a surface-based representation. A small PSNR gap is expected, as volume representations offer inherently more degrees of freedom that can be repurposed to model pixel-wise colors. A similar trend is observed when implementing our method in the ZipNeRF codebase, where we measured mean PSNR values of 29.73 dB for our method and 31.45 dB for NeRF on indoor scenes, and 24.06 dB and 25.24 dB on outdoor scenes, respectively.

When evaluating our relaxed variant—which switches to volumetric rendering in hard regions—the visual quality slightly exceeds the NeRF baseline. This improvement arises because our method encourages surface-like, sparse distributions, resulting in more empty space that the renderer can efficiently skip. Consequently, at equal batch size, Instant NGP automatically spawns more rays when using our method, covering more reference pixels per batch and producing a better reconstruction. When the ray count is restricted to match NeRF, the relaxed variant delivers results that are approximately equal.

These trends are consistent across other metrics as well. For instance, both our method and NeRF achieve SSIM scores of 0.89 (indoor) and 0.68 (outdoor).

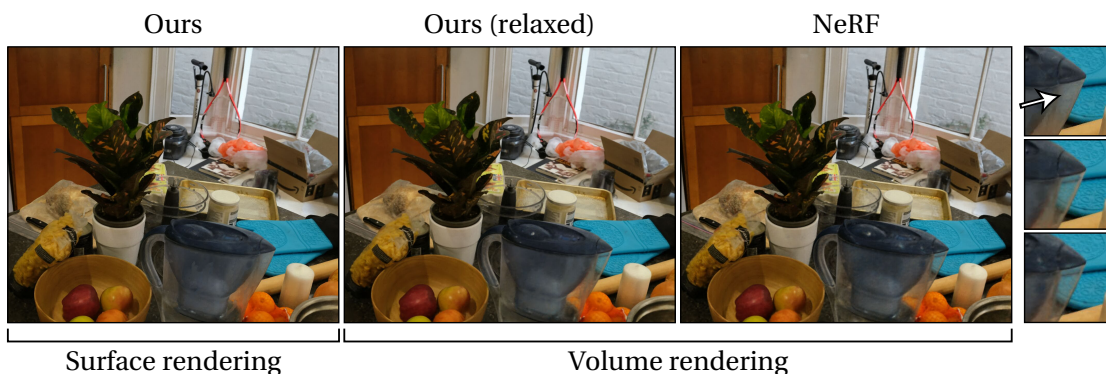


Figure 6.6: **Volumetric relaxation.** We compare reconstructions from our method, both *without* and *with* volume relaxation, against NeRF, all implemented in Instant NGP using the same hyperparameters. While our method achieves comparable visual quality using a surface-based representation, we highlight a region (white arrow) where our method fails to model a semi-transparent object due to the opaque surface assumption. The relaxed variant of our algorithm can recover the object by adopting volume rendering in such regions. Rendering the reconstructions using the same ray marching implementation leads to significant performance differences: our surface-only reconstruction is $2.6\times$ faster than NeRF. The relaxed variant benefits from the surface representation in most regions and is $1.7\times$ faster.

Table 6.1: **Visual quality comparison.** We integrate our loss into Instant NGP and train on the MipNeRF360 dataset using default hyperparameters.

	Indoor mean			Outdoor mean		
	PSNR $\uparrow$	SSIM $\uparrow$	LPIPS $\downarrow$	PSNR $\uparrow$	SSIM $\uparrow$	LPIPS $\downarrow$
Ours	29.02 dB	0.888	0.275	22.41 dB	0.679	0.563
Ours (relaxed)	29.41 dB	0.897	0.284	22.62 dB	0.690	0.626
NeRF	29.19 dB	0.893	0.303	22.47 dB	0.683	0.638

Rendering performance Our implementation builds on the Instant NGP codebase, which ray-marches fields $(\alpha_{\mathbf{p}}, L_{\mathbf{p}})$ represented using an interpolated hash grid lookup combined with a lightweight MLP. We repurpose this ray-marching code for surface rendering by returning the color of the first sample with an occupancy value exceeding 0.5. This straightforward modification results in a $2.4\times$ average speedup in frames per second (FPS) across MipNeRF360 scenes compared to the baseline. An average speedup of $2.0\times$ is achieved for the relaxed version of our method, as most of the scene remains surface-like.

An additional $2\times$ speedup can be achieved by replacing ray-marching with rasterization of a meshed isosurface. In this case, the color network is only used for mesh shading, maintaining the same visual quality as before. Rendering performance can be further boosted, e.g., by storing precomputed hash grid lookups alongside mesh vertices [69], or by projecting the MLP’s directional dependence into spherical harmonics [70].

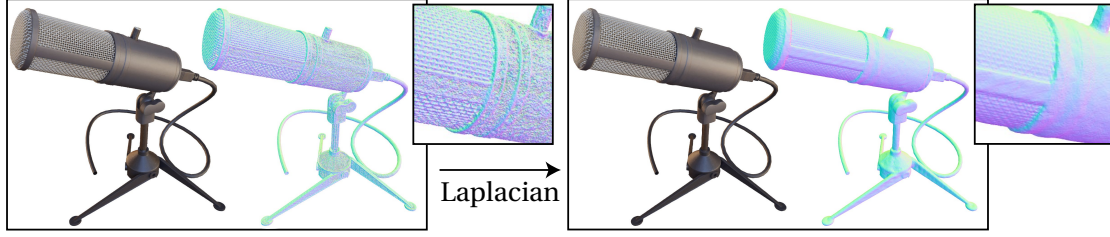


Figure 6.7: **Regularization.** For simple scenes with enough observations, the reconstructed surface closely matches the ground truth geometry without requiring additional constraints. Adding Laplacian refinement helps smooth out unnecessary small kinks, resulting in a more accurate final geometry.

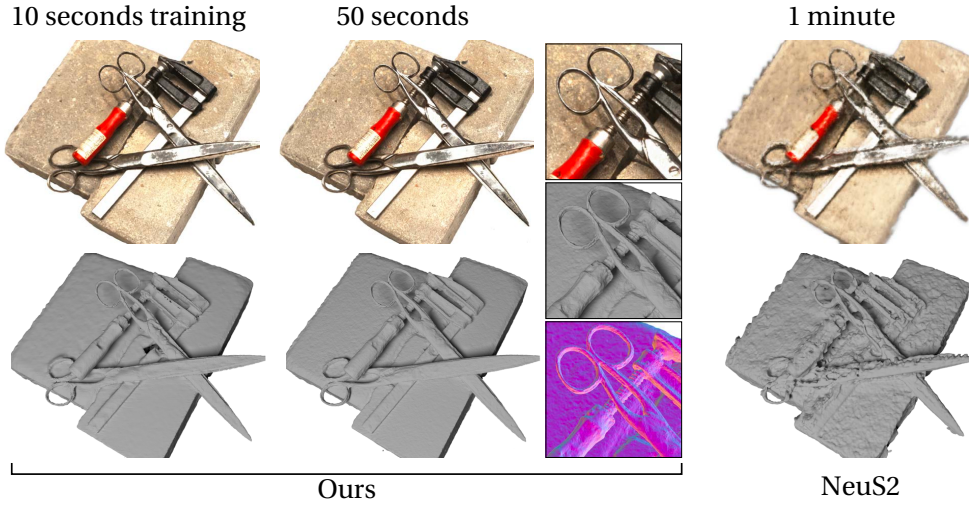


Figure 6.8: **Straightforward extraction.** Since our algorithm does not use an intermediate volume representation, efficient surface extraction is possible at any point. Given the same time budget, a fast NeuS2 baseline [71] still models the scene as a fuzzy volume, and a surface cannot be confidently extracted.

6.3.2 Geometry reconstruction

Our method is also applicable to geometry reconstruction tasks, where the goal is to obtain high-quality meshes that match the ground truth geometry. For simple multi-view input (Figure 6.7), our method produces highly detailed geometry at the speed of Instant NGP (seconds). A mesh can be extracted at any point during the optimization (Figure 6.8). However, complex real-world reconstruction tasks are often under-constrained. For example, a reflective object seen only from a narrow cone of directions does not provide sufficient information for accurate shape recovery. Even with a larger set of viewpoints, it can be challenging to disambiguate whether surface detail is due to local color variation or small-scale geometry. As a consequence, the reconstructed geometry often exhibits undesirable bump-like artifacts representing such misattributed detail. While our algorithm still excels at novel view synthesis under these conditions, the reconstructed geometry can significantly deviate from the ground

Table 6.2: Average Chamfer distance comparison on the DTU dataset with NeuS [66] and NeuS2 [71].

	Ours (1 min)	NeuS (8 hr)	NeuS2 (5 min)
CD	0.80	0.77	0.68

truth.

To mitigate this issue, we incorporate an exponentially decaying Laplacian regularizer during training. This regularizer initially enforces flat surfaces and progressively provides more degrees of freedom as its influence decays. Figure 6.12 examines the influence of the final Laplacian weight on reconstruction quality. Figure 6.13 showcases geometry reconstructions of scenes from the DTU [72] and BlendedMVS [73] datasets, all produced with a consistent Laplacian weight of 2×10^{-5} .

Table 6.2 shows that, with only minimal Laplacian regularization, our method achieves an average Chamfer distance on the DTU dataset that is just 0.12 higher than NeuS2 [71], while reducing runtime to only 1 minute thanks to our algorithmic simplicity. This chapter does not aim to compete on geometric reconstruction metrics and therefore does not incorporate other regularization extensions, which would detract from the simplicity of the presented idea. Such extensions include multi-view consistency losses [74, 75] to reduce ambiguities in regions with limited observations, as well as the TSDF algorithm [76], which helps extract smooth meshes while removing unnecessary geometry.

6.4 Discussion

6.4.1 Choice of background distribution

In subsection 6.2.3, we used the free-flight background distribution to derive a loss form dual to the NeRF loss. This choice is somewhat arbitrary, and other distributions could be used with different trade-offs. In section B.2, we discuss how alternative designs can enable new optimization strategies that are not possible in image-space methods and provide one such example.

6.4.2 Relation to many-worlds inverse rendering

This chapter builds directly on the many-worlds formulation of the previous chapter: both optimize surface distributions without collapsing them into an exponential volume. The difference is the rendering regime. Here, the objects are purely emissive, whereas the previous chapter handles differentiable shadowing and interreflection to reconstruct reflective objects in scenes with global illumination. Superficially, this emissive formulation could be mistaken for a stripped-down version of that physically based formulation.

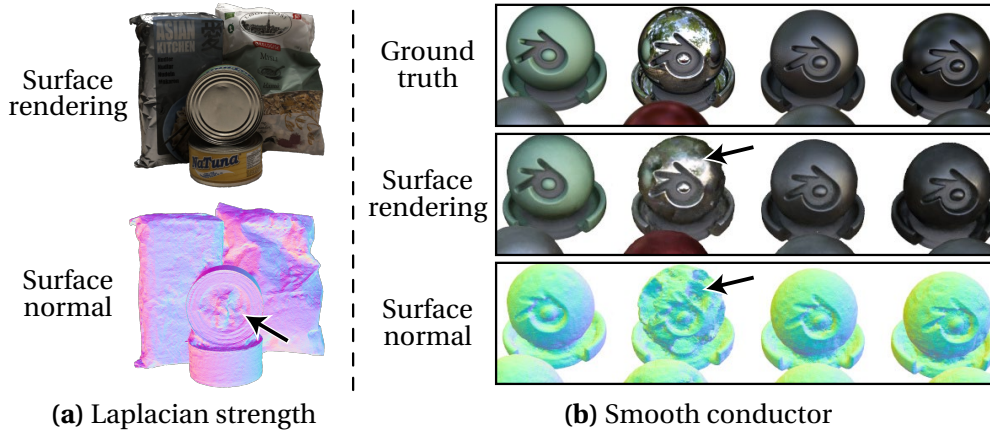


Figure 6.9: **Limitations.** (a) Our Laplacian smoothing strategy fails to reconstruct the flat can surface due to its view-dependent appearance. A larger Laplacian weight can help, but this also suppresses the geometric detail shown in Figure 6.13. (b) High-frequency color variation is more challenging to represent accurately on a surface than in a volumetric representation.

This simplification is nevertheless algorithmically consequential. In radiance surface rendering, image formation is a direct 1:1 mapping between a ray and the nearest intersected surface, whereas the physically based case requires a nested integration over materials, lighting, and geometry. Projecting training images into the scene to define a radiance-field loss depends on this 1:1 mapping and does not efficiently translate to the nested integral structure of global illumination.

Another important difference is the stochastic background distribution introduced in this chapter, which enables topological changes and substantially improves reconstruction quality. Its expectation can be evaluated cheaply enough to maintain algorithmic parity with NeRF. The associated derivations and simplifications (section B.1) are specific to radiance surfaces and do not transfer directly to the physically based formulation.

Mathematically, both chapters start from a distribution over surface hypotheses and treat candidate surfaces as mutually exclusive explanations. The connection is clearest at the level of the local objective. In the physically based chapter, a candidate patch perturbs a nested light-transport integral, and its contribution is weighted by primal and adjoint radiance. In the emissive setting, this nested integral collapses to the local comparison $\ell(L_{\mathbf{p}}, L_{\text{target}})$ at each sampled spatio-directional point. Radiance surfaces can therefore be read as the case in which the many-worlds differential source is replaced by an explicit radiance-field loss.

6.4.3 Limitations and future work

Moving the evaluation of the color loss ℓ from image space into the radiance field makes our method incompatible with loss functions that depend on image-space neighborhoods (e.g., style losses).

As shown in Figure 6.9, our lightweight Laplacian regularization fails when too few observations are available to constrain the geometry. Using alternative regularization techniques from state-of-the-art geometry reconstruction methods could help mitigate this issue. Our method also struggles to capture the appearance of conductive materials accurately; this limitation could be addressed by incorporating techniques from prior work [77].

An interesting extension would be to implement a particle-based representation, such as Gaussian splatting [3]. However, 3D Gaussians are inherently semi-transparent, which conflicts with our assumption of opacity. Future work could explore the use of opaque primitives, such as 2D disks, to replace semi-transparent particles.



Figure 6.10: **Nature of the reconstructed occupancy.** Surface renderings at varying level sets of a scene reconstructed by our method and NeRF, both implemented in Instant NGP using the same hyperparameters. **(a)** For our method, the surface renderings show minimal changes across different level set thresholds, indicating that the occupancy field has converged to a near-Heaviside step function on the surface, allowing for extraction of a surface-based representation. **(b)** NeRF reconstructs the scene volumetrically, and any surface extracted using a level set is a poor approximation of the true color.

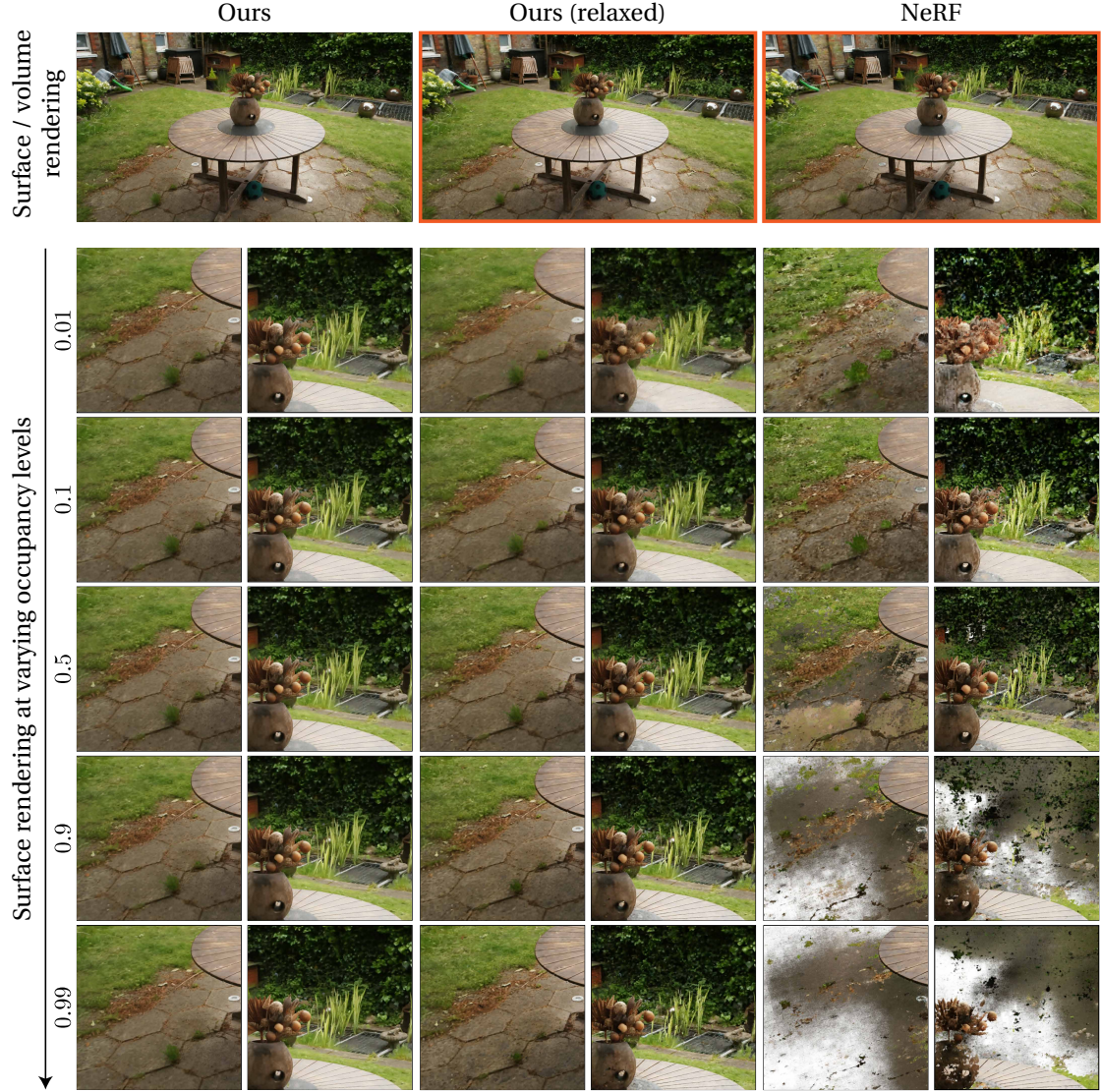


Figure 6.11: **Additional level sets.** Surface renderings at varying level sets of an outdoor scene reconstructed by our method and NeRF, using the same hyperparameters. Only the two images with orange borders are rendered volumetrically. **Left:** For our method, the surface renderings show minimal changes across different level set thresholds, indicating that the occupancy field has converged to a near-Heaviside step function on the surface. **Middle:** The relaxed variant of our algorithm uses volume representation in challenging regions, such as sub-pixel details (yellow arrow). The overall scene remains surface-like, leading to better ray marching performance than NeRF. **Right:** The NeRF reconstruction is inherently volumetric, so renderings of level sets do not produce meaningful visualizations.

6.5 Conclusion

The “many worlds” paradigm—that is, optimizing a distribution over non-interacting primitives—is still relatively new in differentiable rendering. This chapter applied that idea to emissive surface reconstruction and showed that a surface-based derivation can lead to equations that remain very close to volumetric training, except that losses rather than colors are accumulated along rays.

Together with the previous chapter, this result shows that the distribution-over-surfaces viewpoint is not specific to physically based transport. It provides a controlled way to move along the surface–volume axis while preserving an explicit surface target. At the same time, the relaxed variant highlights an unresolved modeling question that recurs throughout the thesis: when should a region of space be treated as a surface, and when should it be treated as a volume?

The next chapter explores the complementary direction in the thesis design space. Instead of relaxing geometry while keeping transport fixed, it keeps the physically based target model explicit while relaxing transport during optimization through radiance caching.

7 Radiance Caching for Differentiable Path Tracing

The previous chapter moved along the surface–volume axis while staying in the emissive regime. This chapter explores the complementary axis of the thesis design space: it keeps the target model physically based but relaxes transport during optimization by introducing a learned radiance cache.

The concrete task is material recovery under unknown illumination with known geometry. This is a deliberately narrower problem than the full inverse-rendering setting considered earlier, yet it isolates a second central bottleneck of this thesis: even when geometry is no longer the main difficulty, physically based optimization can remain unstable because recursive light transport is noisy and the effects of materials and lighting are tightly coupled.

A radiance cache can be understood as a temporary shift toward the emissive end of the spectrum. Instead of explaining every radiance value through recursive path tracing at every optimization step, the algorithm may directly store or predict part of that radiance. This can reduce variance and absorb effects that the current material estimate cannot yet explain.

The final output, however, must still consist of reusable physical materials. A naive blend of cache and path-tracing estimators is therefore not enough: it can match the training images without ensuring that either estimator is correct on its own. The method developed in this chapter resolves this issue with a learned spatial blending field and a family of separated losses that constrain both estimators individually.

The chapter first uses naive blending to expose the underlying decomposition problem, then derives a broader family of separated estimators and losses, and finally studies how the blending field should be optimized in practice.

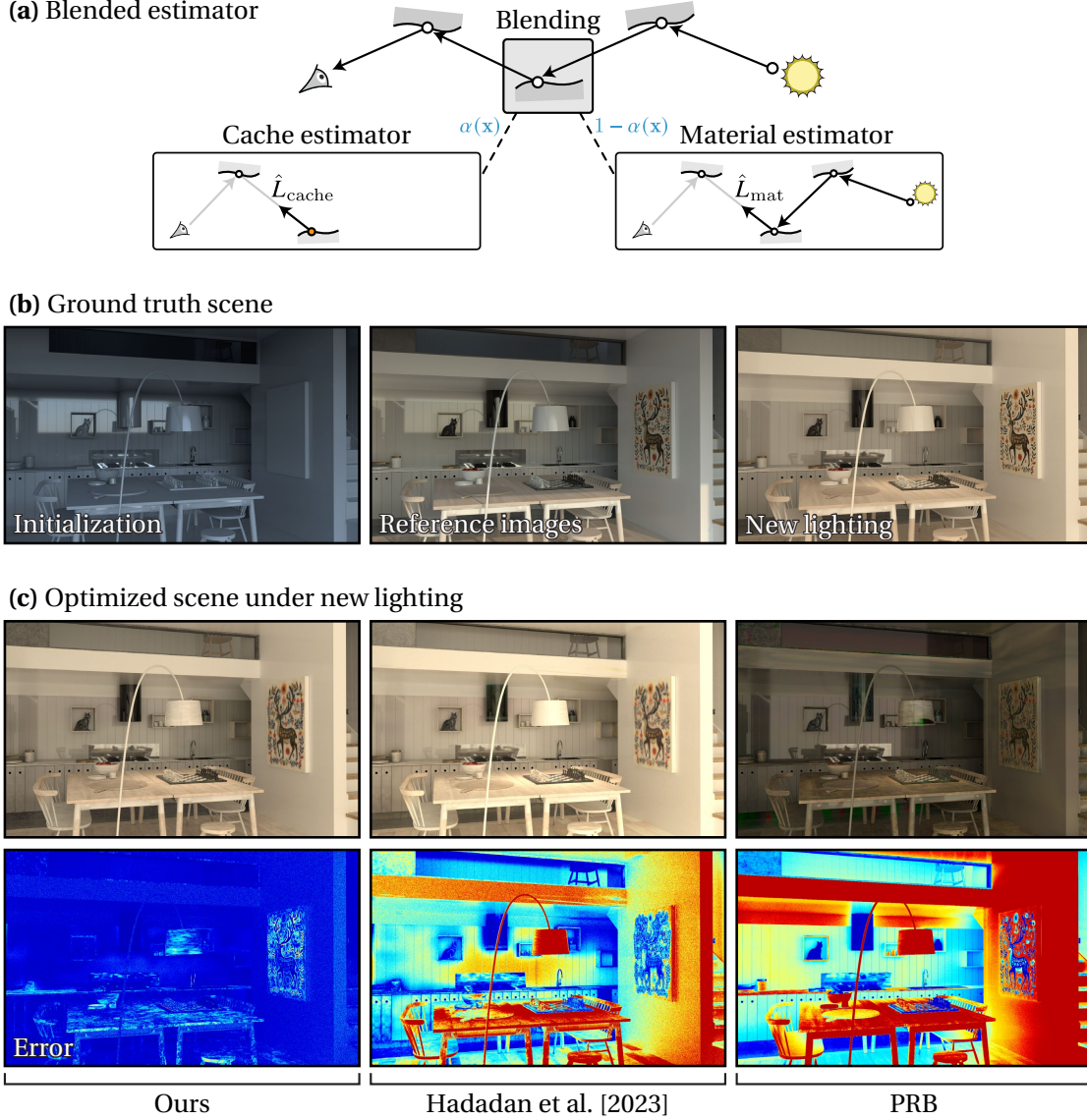


Figure 7.1: **Cache-material blending.** (a) At each surface interaction, outgoing radiance is estimated by blending a cache estimator, which queries a learned radiance cache, and a material estimator, which continues path tracing through BSDFs. A spatial blending field $\alpha(\mathbf{x})$ learns where to trust each estimator, while the training losses constrain both estimators individually so that the recovered materials remain usable for editing and relighting. (b) Starting from a rough material initialization (visualized under unknown lighting), the cache, materials, and α are optimized jointly. (c) At evaluation time, the cache and α are discarded and the recovered materials are rendered alone under an unseen relighting condition, enabling comparison with Hadadan et al. [1] and PRB [29].

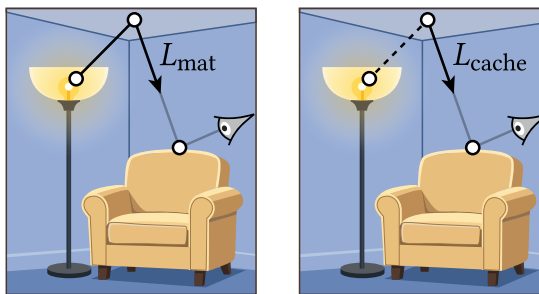
7.1 Introduction

The setting studied in this chapter is intentionally narrower than the full problem of physically based inverse rendering: given reference images and geometry (e.g., from a sensor, multi-view reconstruction, or an earlier optimization stage), the task is to recover materials under unknown illumination. This removes the geometric issues studied in earlier chapters, but it leaves a difficult transport-side optimization problem.

Three issues compound to make this problem brittle: high-variance radiance estimates from the path tracer that lead to noisy gradients, poor numerical conditioning that amplifies sensitivity to noise and initialization, and an optimization landscape with many local minima. Increasing the sample count can reduce Monte Carlo noise, but the cost of reducing that noise to acceptable levels is prohibitive and would address only one of these three issues.

Along a light path, multiple unknown materials contribute multiplicatively to the final pixel value, which makes it difficult for gradient descent to disentangle their individual roles. Unknown illumination worsens this further: the optimizer can explain the reference images by simply placing emission everywhere, which can fit the data but does not yield the desired material-lighting decomposition.

From the viewpoint of the thesis design space, a radiance cache provides a controlled relaxation toward the emissive end. It replaces high-variance Monte Carlo estimates with a learned predictor of outgoing radiance at surface points. Radiance caches are highly effective in forward rendering; for example, Müller et al. [78] demonstrated that the overhead of training and querying a neural radiance cache can be low enough for real-time path tracing. For inverse rendering, however, the central question is not only whether the cache reduces variance, but also how to use it without losing the physical meaning of the recovered materials. Despite pioneering work by Hadadan et al. [1], the use of radiance caching for physically based inversion remains underexplored. This chapter develops a general formulation that includes their method as a special case.



To build intuition, consider light from a lamp that reaches a chair indirectly after reflecting off the ceiling. A standard path tracer would estimate this radiance by tracing a ray to the ceiling and then sampling or evaluating the ceiling's BSDF to connect to the light source. We refer to this approach as the *material estimator* because it simulates the material's reflectance behavior. Alternatively, with a radiance cache that stores outgoing radiance on surfaces, we

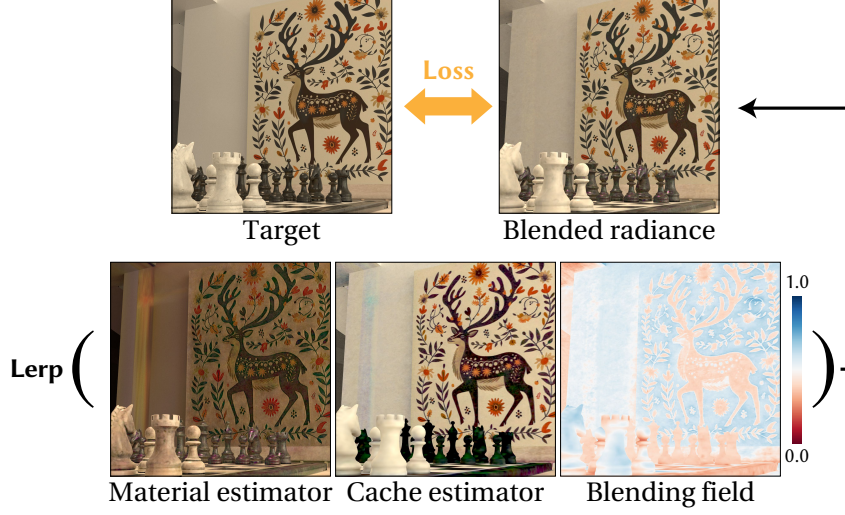


Figure 7.2: **Naive blending is ill-posed.** Optimizing only the blended radiance can match the reference image, yet the cache-only estimate (queried at camera-ray intersections) and the material-only estimate (path tracing without cache) remain incorrect. The resulting parameters are not meaningful in isolation and cannot be reliably reused for relighting or editing.

can query it directly at the ceiling intersection and terminate the path. This *cache estimator* avoids further path tracing and BSDF evaluation; with an accurate cache, the two estimators agree. In practice, they exhibit complementary failure modes: the cache estimator is fast and low-variance but may be unreliable if the cache has not converged, has insufficient capacity, or cannot represent high-frequency directional variation; the material estimator captures such effects more easily but produces noisy estimates that can impede or destabilize optimization.

A natural idea is to blend these two estimators adaptively. We introduce a spatially varying *blending field* $\alpha(\mathbf{x}) \in [0, 1]$ and define a *blended estimator*

$$\hat{L}_o(\mathbf{x}, \boldsymbol{\omega}) = (1 - \alpha(\mathbf{x})) \hat{L}_{\text{mat}}(\mathbf{x}, \boldsymbol{\omega}) + \alpha(\mathbf{x}) \hat{L}_{\text{cache}}(\mathbf{x}, \boldsymbol{\omega}), \quad (7.1)$$

where $\hat{L}_{\text{mat}}$ and $\hat{L}_{\text{cache}}$ are radiance estimates from the material and cache estimators, respectively. Intuitively, $\alpha(\mathbf{x})$ encodes how much we trust the cache at $\mathbf{x}$: when $\alpha(\mathbf{x}) \approx 1$, we terminate paths early using the cache; when $\alpha(\mathbf{x}) \approx 0$, we fall back to the material estimator to capture complex directional effects.

The blended estimator is simple to implement with a few additional lines of code, and it quickly drives the renderer toward the reference. However, as illustrated in Figure 7.2, it suffers from a fundamental decomposition problem: optimizing the blend constrains the estimators in combination but not individually. Even when the blend perfectly matches the reference, the material and cache estimators on their own may be far from the true solution. In other words, the reconstruction is only valid for the particular spatially varying blend used during training, which limits its reuse for downstream tasks such as rasterization, relighting,

or material editing.

The remainder of this chapter therefore examines optimization strategies beyond naive blending, with two goals in mind: to make both estimators accurate in isolation, and to learn a blending field that automatically favors the more reliable estimator at each location. Our experiments suggest that, compared with pure differentiable path tracing, this joint formulation is particularly beneficial in three respects:

1. **Robustness to unknown lighting:** We obtain stable reconstructions without known emitters. Light paths no longer need to reach an emitter; the cache provides reasonable incident radiance at intermediate points, preventing the optimizer from collapsing to degenerate solutions.
2. **Spatially adaptive estimator selection:** The learned blending field $\alpha(\mathbf{x})$ allows the optimizer to rely on the cache in regions requiring long indirect paths, while falling back to the material estimator near glossy surfaces where the cache is biased.
3. **Improved behavior in high-variance regimes:** In scenes with complex global illumination, the cache terminates paths earlier and reduces variance, making optimization tractable in cases where pure path tracing struggles.

The remainder of the chapter follows this progression: it first analyzes the loss-design problem behind naive blending, then develops the separated-estimator formulation and the optimization of the blending field, and finally evaluates the method under unknown and known illumination.

7.2 Related work

7.2.1 Radiance caching

Early work on *irradiance caching* [79, 80] reduced the cost of global illumination by evaluating low-frequency indirect irradiance at sparse points and interpolating it during rendering. Krivanek et al. [81] extended this idea to *radiance caching*, which stores direction-dependent outgoing radiance. A key challenge for caching methods is artifact-free interpolation from sparse samples without blur or light leaks [82, 83, 84, 85]. *Neural radiance caching* [78] replaces explicit interpolation of precomputed samples with a continuous radiance predictor trained online from noisy Monte Carlo estimates. We similarly fit a cache online but focus on its use in inverse rendering. Recently, variations of radiance caching have seen widespread use for real-time path tracing [78, 86, 87, 88, 89].

7.2.2 Differentiable path tracing

There is a rich literature on differentiable path tracing for inverse rendering [36, 25, 26, 90, 91, 92, 93, 31]. Central challenges for differentiable path tracing are visibility discontinuities [25, 40, 26, 41, 4] and the memory usage of the backward pass [28, 29]. In addition to

the usual Monte Carlo noise in incident radiance estimates, gradient estimation introduces additional derivative terms that are often poorly matched to primal sampling strategies; consequently, the variance of gradients can become arbitrarily large [30, 61]. High variance can slow convergence or cause the optimization to fail, especially with non-linear loss functions [94]. Variance reduction strategies include importance sampling of local gradient terms [30, 95], path guiding [96], reservoir sampling [97, 98], and control variates [94, 99]. The impact of gradient noise can also be mitigated using preconditioning [54, 100] or filtering [101]. We focus on reducing variance due to complex global illumination and unknown emitters using radiance caching. This is largely orthogonal to those techniques and could be combined with many of them. Closely related are *variance-aware* optimization techniques [102, 103]. Accounting for rendering variance explicitly is an interesting future direction.

7.2.3 Radiance caching in inverse rendering

Several works have explored the application of radiance caching to inverse rendering. The closest prior method is the *neural radiometric prior* [1], which we discuss in detail below. We also review the use of radiance caching with radiance field representations (e.g., NeRFs) and alternative approaches.

Neural radiometric prior. Hadadan et al. [1] augmented a differentiable path tracer with a neural radiance cache, demonstrating improvements over unbiased differentiable path tracing [29]. Their method evaluates the cache at a fixed path depth (typically at the first surface hit) and only accumulates gradients into the material at that interaction, even if the path continues further. The method in this chapter generalizes their approach by removing both limitations and introducing a more general family of consistency losses. section 7.4 provides detailed comparisons.

Radiance field methods. Following the seminal work on neural radiance fields (NeRFs) [2], there has been continued interest in developing *reliable* variants of radiance-field representations. Because neural radiance fields are expensive to evaluate, most methods in this area try to limit the number of ray queries or interactions they evaluate. Initially, Zhang et al. [104] augmented NeRF with a direct illumination term. Since NeRFs represent a scene’s radiance field, a natural extension is to leverage that field directly as a radiance cache to render indirect illumination [105, 106, 107]. Gu et al. [108] similarly compute a single indirect bounce on a 2D Gaussian representation [3, 109]. Variants of this idea are *neural incident radiance fields* [110, 111, 112, 113]. These methods represent both outgoing and incident radiance fields, allowing them to avoid recursive ray tracing when evaluating the radiance cache. Neural fields have also been used for more accurate emitter modeling in physically based inverse rendering [114]. In general, NeRF-based methods avoid multi-bounce ray tracing due to the high computational cost. For that reason, the quality of results can deteriorate if the cache is not expressive enough. Attal et al. [115] recognized this problem and proposed several strate-

gies to reduce bias, but their method also does not trace multiple light bounces. We instead use multi-bounce path tracing to disentangle materials from the cache and further reduce bias from the inherent limitations of radiance caching. This makes the classic bias-variance tradeoff explicit.

Alternative approaches. Radiance caching is not the only way to avoid recursive ray tracing in inverse rendering. Jiang et al. [116] perform an iterative radiosity solve directly on 3D Gaussian primitives. Poirier-Ginter et al. [117] path trace only specular transport and cache the diffuse component. This improves novel view synthesis but does not directly enable relighting. Finally, Hadadan and Zwicker [118] cache differential quantities at the cost of limiting reconstruction to low-dimensional parameter spaces.

7.3 Method

As previously discussed, naive optimization of the blended estimator is under-constrained because the model can perfectly match the reference by arbitrarily blending two individually incorrect estimators. In this section, we explore several loss formulations to resolve this ambiguity and to learn the blending field $\alpha(\mathbf{x})$.

Our starting point is the rendering equation, which relates outgoing and incident radiance at a surface point:

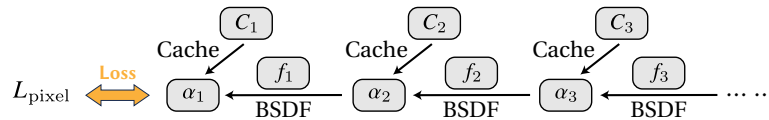
$$L_o(\mathbf{x}, \omega) = L_e(\mathbf{x}, \omega) + \int_{\mathcal{S}^2} L_i(\mathbf{x}, \omega') f(\mathbf{x}, \omega', \omega) d\omega'^\perp, \quad (7.2)$$

where L_e and L_i denote the emitted and incident radiance, and f is the BSDF. A path tracer estimates the integral by sampling a direction ω' and recursively estimating incident radiance. For now, we assume the emission is either known or zero. At each surface interaction, our blended estimator interpolates between two approaches:

$$\hat{L}_o = (1 - \alpha) \underbrace{\frac{f \cos \theta'}{p(\omega')}}_{\text{Material}} \hat{L}_i + \alpha \underbrace{C(\mathbf{x}, \omega)}_{\text{Cache}} \quad (7.3)$$

where $\alpha(\mathbf{x}) \in [0, 1]$ models our trust in the cache at position $\mathbf{x}$, C is a learned cache storing outgoing radiance, θ' is the angle between ω' and the surface normal, and $p(\omega')$ is the sampling probability.

The illustration below shows the computation graph for directly optimizing this recursive estimator. All terms receive gradients (gray boxes), but only their weighted combination is constrained, which leads to an ill-posed problem.



Throughout optimization, image formation uses the same blended estimator. The contribution of this chapter lies in how the full set of parameters (cache, materials, and α) is trained: additional losses enforce the correctness of the individual estimators instead of constraining only their weighted combination.

7.3.1 Consistency losses

We first present two intuitive consistency loss formulations to address this issue and then show that both are inherently flawed.

Depending on the parameter being optimized, we can enforce consistency in either direction: by training the cache to match the material estimator (subsection 7.3.1), or vice versa (Equation 7.3.1). Both losses follow directly from the rendering equation but require care in choosing which terms to differentiate and how to sample them to obtain unbiased gradients.

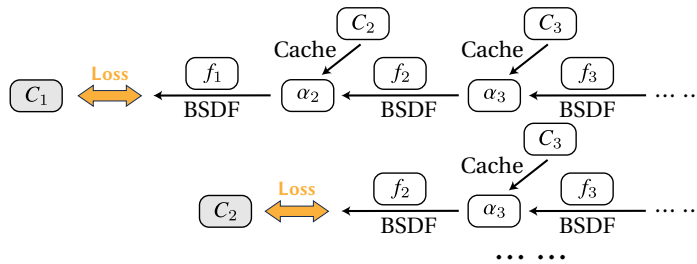
We assume here that the consistency losses are applied in the context of minimizing an image loss based on the blended estimator while also optimizing the blending field. If both estimators are consistent, blending becomes primarily a bias-variance tradeoff: α selects whichever estimator is more reliable at each location.

Cache consistency loss

The first loss formulation trains the cache C to match a one-sample estimate of the rendering equation:

$$\mathcal{L}_{\text{cache}}(\mathbf{x}, \omega) = \left\| C(\mathbf{x}, \omega) - \int_{\mathcal{S}^2} f \cos \theta' L_i(\mathbf{x}, \omega') d\omega' \right\|_{\text{rel}}^2. \quad (7.4)$$

The loss is parameterized by $\mathbf{x}$ and ω . In practice, it would be used by accumulating gradients with respect to $\mathcal{L}_{\text{cache}}$ whenever the path tracer encounters a vertex $\mathbf{x}$ from direction ω and then samples a scattered direction ω' , which also provides the direction for a single-sample estimate of the integral in Equation 7.4. This focuses the consistency optimization on the spatio-directional subset of ray space that actually contributes to the rendered image. Imposing this loss at different path depths leads to the following graph structure, with gray terms receiving gradients and other terms held fixed.



This is the loss used by Müller et al. [78] to train a neural radiance cache in forward rendering. The target is a one-sample Monte Carlo estimate, which is noisy; following their approach, we use a relative L^2 loss so that the cache learns the expected value rather than overfitting to

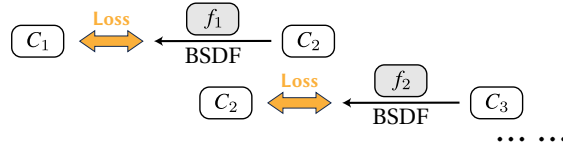
individual samples [119]. In practice, we substitute the blended estimate for $\hat{L}_i$, which reduces variance and reuses values already computed along the path.

Material consistency loss

Conversely, we can train the BSDF to match the cache. If we use C to evaluate both incident and outgoing radiance, the rendering equation provides a training signal for f :

$$\mathcal{L}_{\text{mat}}(\mathbf{x}, \omega) = \left\| C(\mathbf{x}, \omega) - \int_{\mathcal{S}^2} \boxed{f(\mathbf{x}, \omega', \omega)} \cos \theta' C(\mathbf{x}', -\omega') d\omega' \right\|_{\text{rel}}^2, \quad (7.5)$$

where $\mathbf{x}'$ is the surface point visible from $\mathbf{x}$ in direction ω' . Here, gradients only propagate through the BSDF (gray box), while the other terms are held fixed. As before, this loss can be imposed at different path depths:



Unfortunately, the simple approach to constructing a one-sample derivative estimator that worked in Section 7.3.1 fails here and produces biased gradients. The issue arises from the combination of the nonlinear L_2 loss and the fact that we are now propagating derivatives *into* the integral. Appendix C.1 provides more detail and presents a decorrelated two-sample estimator that resolves the issue.

Limitations

The consistency losses suffer from multiple fundamental limitations:

- **Directional mismatch.** The cache consistency loss propagates information from the emitters toward the camera and is therefore well-suited to forward rendering, where known emitter and material parameters are used to infer the observed image. In inverse rendering, the supervision comes from reference imagery at the camera. The loss therefore propagates information against the direction of supervision, making it inherently less suitable.
- **Garbage in, garbage out.** Early in optimization, both the materials and the cache are unreliable (e.g., randomly initialized). Training with a consistency loss then simply transfers unreliable estimates from one representation to the other. This is particularly problematic for material consistency: the cache has limited capacity and cannot represent the radiance function with high accuracy. High-frequency spatio-directional variation may be severely distorted or lost entirely, preventing the cache from ever becoming an authoritative training signal for materials.
- **Unknown emission.** Both losses assume non-emissive surfaces, yet emission must clearly be present for any radiance to exist. Generalizing the formulations to include an emission

term does not help: reflected and emitted radiance cannot be separated without additional constraints, so the consistency relationship no longer defines a meaningful target for either estimator.

These limitations motivate an alternative approach. We modify the role of the cache so that it represents total outgoing radiance (i.e., including both emission and scattering) instead of trying to separate the individual contributions. On top of this representation, we introduce a unified family of losses that simultaneously constrain both estimators with respect to the reference images. This drives gradients from the camera into the scene, aligning the natural flow of information with the inverse rendering problem.

7.3.2 Separated estimators and losses

The cache, material, and blended estimators all aim to recover outgoing radiance L_o . At any surface interaction along a path, we can therefore choose either to terminate the random walk via a cache lookup or perform a recursive estimate. Each strategy yields a distinct rendering and hence a distinct image-space loss. The key observation is that satisfying *all such losses simultaneously* constrains the estimators to be individually correct. This construction is illustrated in Figure 7.3: instead of supervising only the recursively blended prediction, we define losses that force paths to terminate at different depths.

Concretely, terminating at depth k while using the blended estimator at earlier vertices gives:

$$\hat{L}_o^{(k)}(\mathbf{x}, \omega) = \begin{cases} (1 - \alpha) \frac{f \cos \theta_i}{p} \hat{L}_o^{(k-1)}(\mathbf{x}, \omega') + \alpha C(\mathbf{x}, \omega), & k > 1, \\ C(\mathbf{x}, \omega), & k = 1. \end{cases} \quad (7.6)$$

where some variable dependencies are omitted for notational simplicity. Prior to vertex k , the expression depends on the cache, BSDF, and blending field. At vertex k , only the cache contributes, so it alone must represent the correct outgoing radiance. All three parameter sets (cache, BSDF, blending field) receive gradients during optimization.

Implementation A naive optimization approach would simultaneously compute and optimize renderings for all $k \in \{1, \dots, k_{\max}\}$, but this is too memory-intensive. Instead, we can sample a different termination depth k each iteration, keeping memory constant. The distribution over k determines the relative weight of each *separated loss*.

Termination strategies One concern is that simple uniform sampling may terminate at vertices with a low α value, propagating errors from an unreliable cache to upstream BSDFs.

In practice, optimizers like Adam are fairly robust to occasional bad gradients. Nonetheless, an α -aware strategy that extends the path beyond the sampled depth until $\prod(1 - \alpha)$ falls below

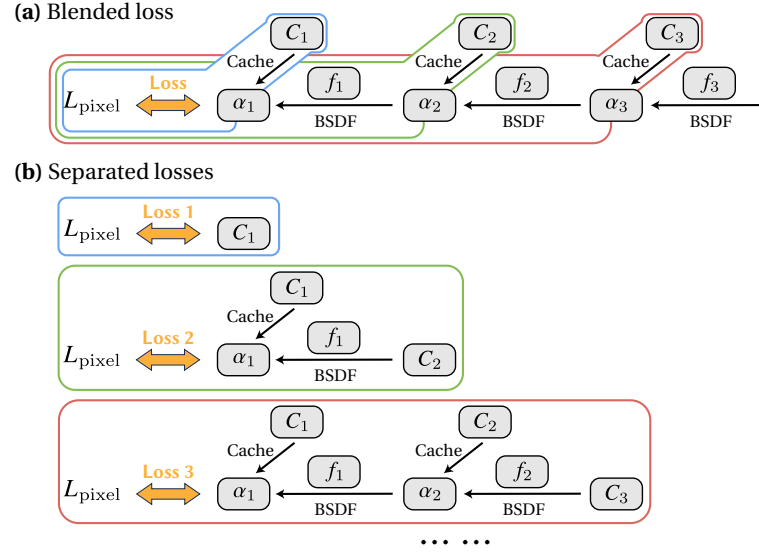


Figure 7.3: **Separated losses.** (a) A single blended loss supervises only the recursively mixed estimator, which can leave the cache and material factors under-constrained in isolation. (b) We instead define pixel losses that force the path to terminate at a chosen bounce k , using the cache value C_k as the endpoint. Each such loss backpropagates to C_k and to the BSDF factors before it, removing the “only-the-blend-is-supervised” ambiguity.

a threshold avoids low- α endpoints and improves stability by a small margin (~ 0.4 dB PSNR across scenes). We detail options in Appendix C.3.

Algorithm 7.1 shows a pseudocode implementation of the forward pass. The reverse-mode derivative of this algorithm can be evaluated using path replay backpropagation [29].

7.3.3 Optimizing the blending field

The blending field $\alpha(\mathbf{x})$ locally controls how much we rely on the cache versus the material estimator. Consider setting $\alpha = 0$ everywhere: paths would then use only the material estimator and terminate using the cache when they reach a maximum depth. This works but sacrifices two benefits: blending at earlier vertices reduces variance, and a family of losses probes both estimators in different ways, yielding a better-posed inverse problem. Section 7.4 analyzes these benefits in more detail.

We instead optimize α jointly with the cache and scene parameters, treating it as a learned *confidence field*. Since the true outgoing radiance is unknown during optimization, either estimator may be more accurate at any given point and training stage. Early on, the materials may be far from correct while the cache already captures much of the signal. Learning α lets the system favor the more reliable estimator at each location and adapt as both evolve.

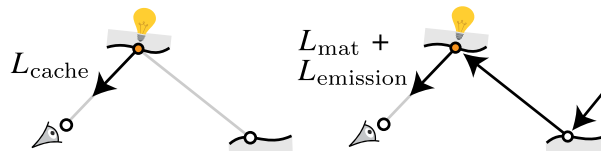
Chapter 7. Radiance Caching for Differentiable Path Tracing

Algorithm 7.1: Pseudocode of the separated radiance estimator (Equation 7.6) using a blending field-aware stopping criterion (section C.3) with threshold τ . The structure of this algorithm resembles standard (differentiable) path tracing and only requires minimal changes in existing systems.

```
1 def render_pixel(pixel_uv,  $\tau$ ):
2      $\mathbf{x}, \omega$  = generate_camera_ray(pixel_uv)
3     return  $L_i(\mathbf{x}, \omega, \tau)$ 
4
5 def  $L_i(\mathbf{x}, \omega, \tau)$ :
6      $L = 0$ ;  $\beta = 1$ ;  $\alpha_{\text{prod}} = 1$ 
7     for _ in range(MAX_DEPTH):
8          $\mathbf{x} = \text{ray\_intersect}(\mathbf{x}, \omega)$ 
9          $\alpha, C = \text{eval\_cache\_alpha}(\mathbf{x})$ 
10        # Path termination check (separated estimator)
11        if  $\alpha_{\text{prod}} * (1 - \alpha) < \tau$ :
12            return  $L + \beta * C$ 
13        # Accumulate cache contribution and continue path
14         $L += \beta * \alpha * C$ 
15         $\omega, \beta_{\text{bsdf}} = \text{sample\_bsdf}(\mathbf{x}, \omega)$ 
16         $\beta *= (1 - \alpha) * \beta_{\text{bsdf}}$ ;  $\alpha_{\text{prod}} *= (1 - \alpha)$ 
17    return  $L$ 
```

Bias-driven optimization We optimize α so that it favors the estimator with lower bias, which requires decorrelating the loss and gradient evaluations following standard practice in differentiable rendering [36, 90]. This provides a fallback when either estimator struggles, improving robustness. When the material estimator is unreliable in a region (e.g., due to high variance, poor initialization, or difficult transport), α naturally increases, and the system becomes more reliant on the cache. Conversely, where the cache is unreliable (e.g., on the mirror in Figure 7.4), α decreases and the system falls back to path tracing. This adaptive behavior prevents the failure of either estimator from destabilizing the optimization.

Handling unknown emission A unique challenge in inverse rendering is that emitter locations are often unknown. In this formulation, the cache stores full outgoing radiance including emission, while the material estimator computes only reflected radiance. On emissive surfaces, setting $\alpha < 1$ would blend these inconsistent quantities.



Fortunately, the optimization naturally handles this: as shown in Figure 7.4a and other results, α converges to 1 on emissive surfaces because this is the only configuration that can correctly model emission. Note that $\alpha = 1$ is not a definitive indicator of emission—it can also occur

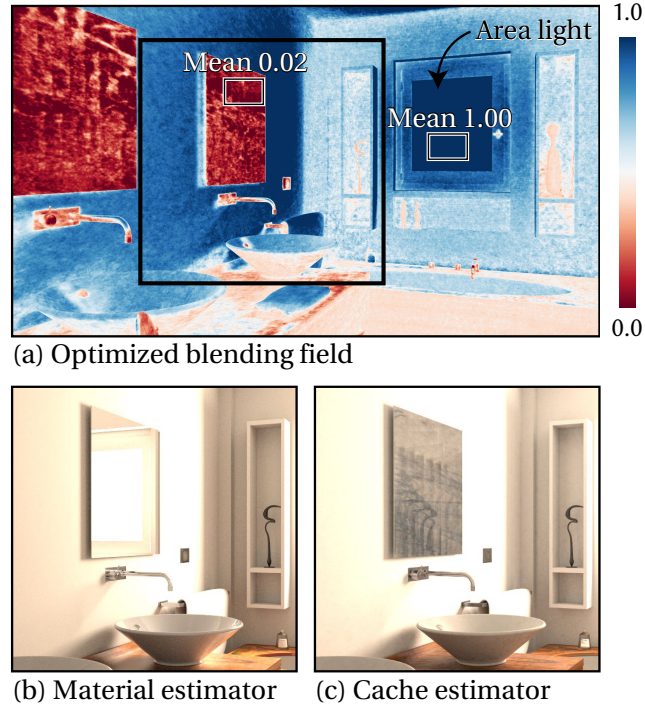


Figure 7.4: **Adaptive estimator selection via a learned blending field.** (a) Optimized blending field $\alpha(\mathbf{x})$. Blue indicates greater reliance on the cache estimator. (b) Rendering using the optimized materials without the cache. (c) Rendering using only the cache at camera-ray intersection points. $\alpha(\mathbf{x})$ increases reliance on the cache where indirect transport is high-variance and reduces reliance where the cache is inaccurate (e.g., specular effects).

where material optimization struggles.

Variance-aware optimization An alternative is to optimize α for both bias and variance by *not* decorrelating paths, which naturally favors the estimator with lower variance. This follows the principle of variance-aware inverse rendering [103], which shows that lower variance can lead to better optimization. While using correlated paths for material parameters yields biased gradients [90, 120], this is acceptable for α because both estimators target the same quantity. We include a comparison in section 7.4 and leave further exploration for future work.

Alternative formulations For completeness, we note that α can also be optimized via blended local losses computed at each bounce, without additional overhead. This idea has been used in Zhang et al. [6] for optimizing blended losses over a surface distribution. We discuss this alternative in Appendix C.2.

7.3.4 Connection to Hadadan et al. [1]

Hadadan et al. [1] can be seen as a special case of the framework above, tailored to single-bounce material optimization. Key differences are *where* cache supervision is applied, *how* gradients are routed, and *whether* the estimator is allowed to adapt across space and path depth.

Mapping to our formulation. Hadadan et al. combine three ingredients: (i) a *primary-hit material loss* in which materials at the first surface interaction are optimized using cached incident radiance from the next bounce; (ii) a *primary-hit cache loss* supervised directly by reference images; and (iii) a *cache-consistency* objective as in neural radiance caching [78] that propagates radiance information from emitters toward the camera.

In our terminology, (i) corresponds to a separated loss with $k = 2$ that updates materials using cached continuation beyond the first hit (with gradients detached from the cache and no blending), (ii) corresponds to a separated loss with $k = 1$ that directly supervises cache values at camera-ray intersections, and (iii) corresponds to a cache-consistency term that is effective for forward rendering but can become unreliable for inverse rendering, especially when emission is unknown.

Generalization in this chapter. The approach here removes the single-bounce restriction by optimizing a *family* of separated objectives and introducing a learned blending field $\alpha(\mathbf{x})$ that adaptively selects between cache and material estimators across the scene. This makes the optimization more robust: when estimating incident radiance, cache values are only used where reliable, providing more accurate derivatives for materials along the entire light path.

Hadadan* baseline. In the experiments below, we also report a variant, Hadadan*, which disables the cache-consistency loss. This isolates the limitations of consistency terms in the unknown-lighting setting; we discuss the resulting behavior in subsection 7.4.1.

7.4 Results

The evaluation proceeds in three stages. First, we present results in a hard setting: room-scale material recovery under unknown lighting. Second, we analyze the effects of key design choices. Third, we present experiments under known lighting to show the benefits of the method even in applications where lighting is available.

Evaluation methods The method labeled “Ours” uses separated losses to train the cache, materials, and the blending field $\alpha(\mathbf{x})$, which is optimized for low bias (i.e., with decorrelated paths). At evaluation time, we discard both the cache and the blending field and render images

with path tracing using only the optimized material parameters. This evaluates whether the recovered materials are physically plausible and reusable, rather than whether a training-time blend can match the input images. For all tables, we compute metrics by rendering unseen test views under a scene-specific adjusted relighting condition.

We do not rely on texture-space error as a primary metric because we optimize *all* materials jointly, including many regions that are occluded (e.g., the floor under furniture) or never observed, making texture-space evaluation unreliable.

Metric	Method	Bedroom	Bathroom2	Kitchen	Living-rm	Living-rm-2	Living-rm-3	Staircase
PSNR $\uparrow$	Ours	24.85	23.43	26.96	22.64	27.56	29.57	20.71
	PRB	11.46	4.66	10.09	15.46	10.54	6.15	10.65
	Hadadan [2023]	16.47	14.82	23.92	18.50	20.33	30.42	12.15
	Hadadan [2023]*	18.26	15.24	24.21	18.89	19.91	31.28	17.33
SSIM $\uparrow$	Ours	0.5225	0.8032	0.6724	0.8068	0.8331	0.8769	0.8350
	PRB	0.4445	0.3357	0.4028	0.7101	0.5724	0.4464	0.3808
	Hadadan [2023]	0.4702	0.7182	0.6573	0.7474	0.8073	0.8821	0.6438
	Hadadan [2023]*	0.4904	0.7190	0.6588	0.7519	0.8049	0.8805	0.7401
LPIPS $\downarrow$	Ours	0.2190	0.1254	0.2154	0.1257	0.1187	0.0721	0.1184
	PRB	0.3113	0.4112	0.2053	0.1927	0.2030	0.2638	0.3098
	Hadadan [2023]	0.2698	0.1712	0.2284	0.1530	0.1411	0.0689	0.1910
	Hadadan [2023]*	0.2543	0.1581	0.2279	0.1521	0.1427	0.0738	0.1771

Table 7.1: **Unknown-lighting reconstruction quality.** We assume known geometry but unknown lighting and materials. We evaluate by rendering test views using *material-only* path tracing (discarding the cache and α) under a new lighting condition. The Hadadan* variant [1] disables its cache-consistency loss. Best results per scene/metric are highlighted.

7.4.1 Unknown lighting material optimization

When both lighting and materials are unknown, differentiable path tracing struggles to disentangle emission from reflectance. Gradient variance compounds this difficulty: every surface could be an emitter and must be sampled as such, leading to low-quality gradients that exacerbate an already degenerate problem.

Quantitative comparison. Table 7.1 reports reconstruction quality using material-only renderings of test views. Our method substantially outperforms PRB and improves over Hadadan [1]. PRB fails in this setting because it tries to explain most of the appearance with optimized emission, leading to poor material estimates that are not useful on their own.

We also report results for Hadadan*, which disables the cache-consistency loss. This variant can improve numerical scores in some scenes because the cache-consistency loss propagates radiance from emitters toward the camera, leading to incorrect decompositions when the

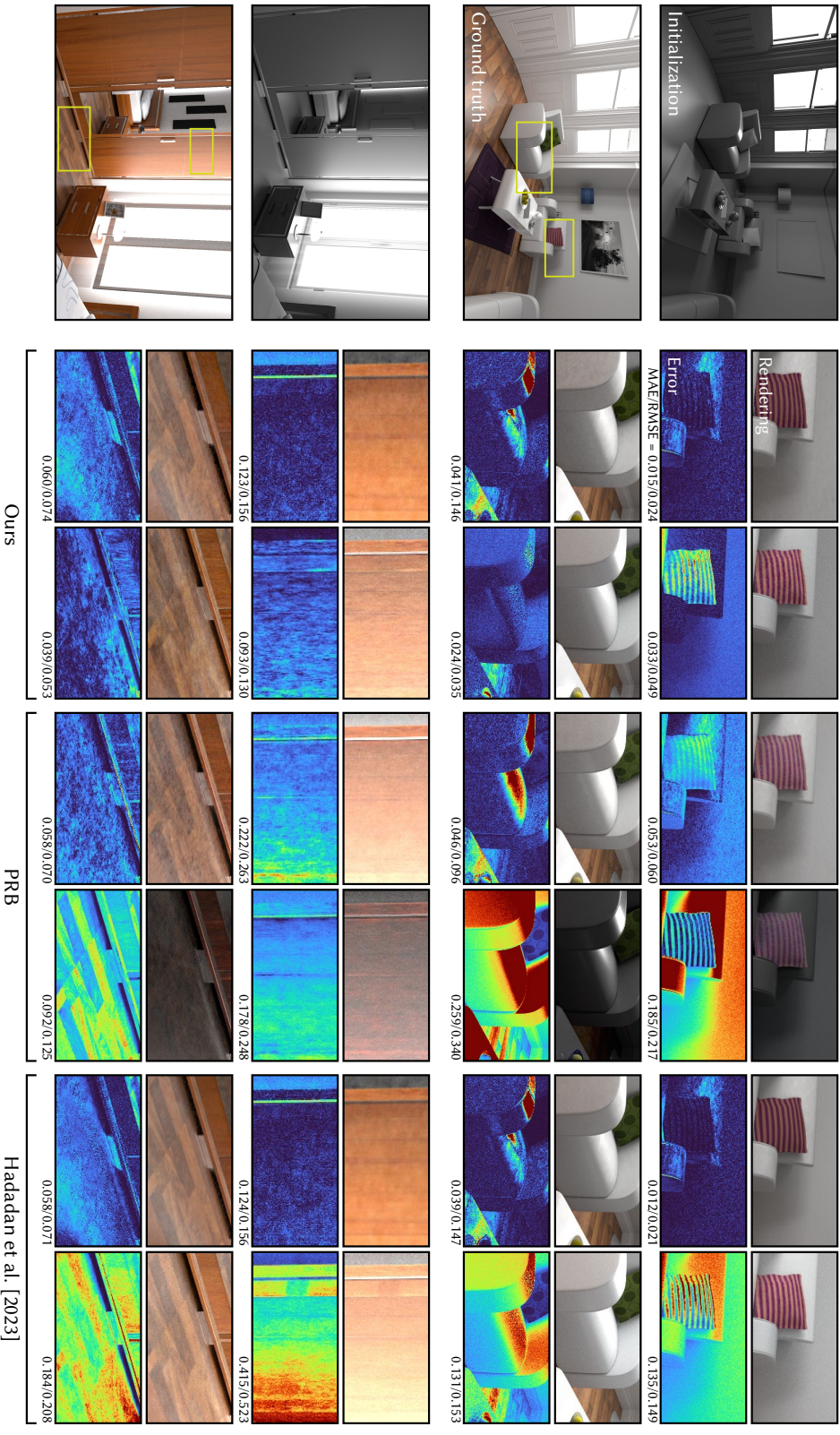


Figure 7.5: **Material reconstruction from unknown lighting.** Visual comparison corresponding to Table 7.1. For our method and Haddadan et al. [1], we show the cache rendering at camera-ray intersections and a *material-only* rendering illuminated by the ground-truth lighting (unknown during optimization) to verify the decomposition. PRB optimizes both materials and emission; we show its optimized-light rendering (used during training) and its material-only rendering under ground-truth lighting.

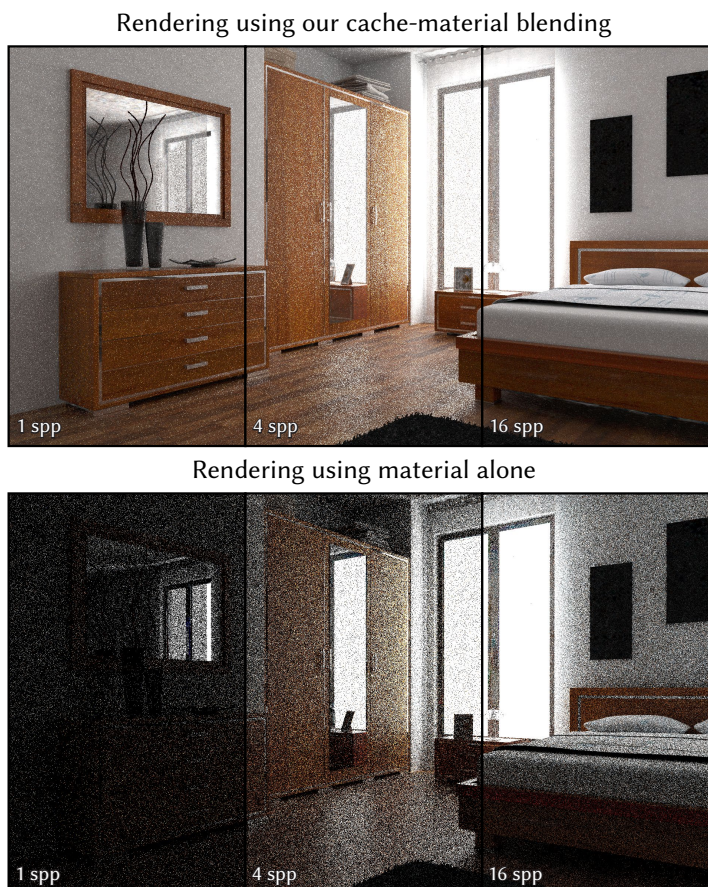


Figure 7.6: **Blending reduces rendering variance.** We render the same optimized scene with a low sample count using (left) our blended estimator and (right) the material estimator (without cache/blending field). Blending substantially reduces variance for the same sample count, which translates into cleaner optimization gradients.

emission is unknown.

Qualitative comparison. Figure 7.5 visualizes representative reconstructions. For our method and Hadadan, we show the cache rendering at camera-ray intersections and the *material-only* rendering illuminated by the ground-truth lighting. PRB is shown both with its optimized emission and with ground-truth lighting. Consistent with the quantitative results, our recovered materials remain plausible, while PRB frequently produces materials that only explain the training images when paired with its optimized emission.

Blending reduces rendering variance. Figure 7.6 visualizes two renderings of the same optimized scene, one made with the pure material estimator (no cache, no α), and one made using the blended estimator, which produces a markedly cleaner image. These reductions in primal variance directly translate into lower-variance gradients that stabilize the optimization

Chapter 7. Radiance Caching for Differentiable Path Tracing

Scene	PSNR $\uparrow$			SSIM $\uparrow$			LPIPS $\downarrow$		
	Ours	$\alpha = 0.5$	$\alpha = 0$	Ours	$\alpha = 0.5$	$\alpha = 0$	Ours	$\alpha = 0.5$	$\alpha = 0$
Bedroom	24.85	23.12	12.07	0.5225	0.5158	0.3955	0.2190	0.2183	0.3653
Bathroom2	23.43	16.54	11.73	0.8032	0.8132	0.7609	0.1254	0.1518	0.2766
Kitchen	26.96	23.58	10.62	0.6724	0.6355	0.4519	0.2154	0.2473	0.4127
Living-room	22.64	18.54	9.76	0.8068	0.7399	0.5095	0.1257	0.1563	0.3283
Living-room-2	27.56	18.75	8.47	0.8331	0.7808	0.6138	0.1187	0.1554	0.3368
Living-room-3	29.57	28.74	14.61	0.8769	0.8750	0.7771	0.0721	0.0728	0.1939
Staircase	20.71	16.52	6.98	0.8350	0.7569	0.4229	0.1184	0.1291	0.4826

Table 7.2: **Effect of optimizing the blending field.** An ablation under unknown lighting using the same evaluation as Table 7.1 (material-only relighting). We compare learning $\alpha(\mathbf{x})$ (ours) to fixing $\alpha(\mathbf{x}) = 0.5$ and disabling blending altogether ($\alpha(\mathbf{x}) = 0$). Learning $\alpha(\mathbf{x})$ provides consistent gains.

and ultimately produce higher-quality reflectance parameters.

Simple scenes. One notable exception is the scene Living-room-3 in Table 7.1, where Hadadan* can achieve higher PSNR. This scene is dominated by direct illumination and can be sufficiently explained by single-bounce optimization, so longer paths provide little benefit and can introduce additional Monte Carlo noise.

7.4.2 Design choices

We next analyze how each key component contributes to performance in the unknown-lighting setting.

Importance of the blending field $\alpha(\mathbf{x})$

Table 7.2 compares the full method against two alternatives: one with a constant blending field (in particular, $\alpha(\mathbf{x}) = 0.5$), and another that disables blending altogether ($\alpha(\mathbf{x}) = 0$). Learning a heterogeneous $\alpha(\mathbf{x})$ yields the best performance across scenes. A fixed blend helps but is consistently worse than an adaptive field, while disabling blending degrades performance severely.

Figure 7.4 illustrates why: the learned $\alpha(\mathbf{x})$ encodes a spatial map of estimator reliability, favoring the cache where material estimation struggles and falling back to Monte Carlo sampling where the cache bias is large. This adaptive routing is crucial under unknown lighting, where the optimizer must balance variance reduction against the risk of biased cache predictions.

Scene	Metric	Max light path depth				
		2	3	4	5	6
Bedroom	PSNR $\uparrow$	20.48	23.67	24.71	24.85	24.85
	SSIM $\uparrow$	0.5036	0.5169	0.5213	0.5222	0.5225
	LPIPS $\downarrow$	0.2403	0.2233	0.2195	0.2191	0.2190
Bathroom2	PSNR $\uparrow$	14.03	16.82	24.57	23.81	23.45
	SSIM $\uparrow$	0.8017	0.8091	0.8089	0.8050	0.8033
	LPIPS $\downarrow$	0.2043	0.1666	0.1188	0.1232	0.1253
Kitchen	PSNR $\uparrow$	25.45	27.25	27.07	26.97	26.97
	SSIM $\uparrow$	0.6506	0.6689	0.6714	0.6719	0.6724
	LPIPS $\downarrow$	0.2362	0.2184	0.2164	0.2160	0.2154
Living-rm	PSNR $\uparrow$	22.46	22.60	22.59	22.59	22.64
	SSIM $\uparrow$	0.8025	0.8061	0.8060	0.8058	0.8067
	LPIPS $\downarrow$	0.1270	0.1271	0.1262	0.1261	0.1257
Living-rm-2	PSNR $\uparrow$	21.44	27.38	27.44	27.51	27.56
	SSIM $\uparrow$	0.8124	0.8330	0.8330	0.8331	0.8331
	LPIPS $\downarrow$	0.1386	0.1190	0.1187	0.1187	0.1187
Living-rm-3	PSNR $\uparrow$	35.06	33.86	27.44	29.75	29.67
	SSIM $\uparrow$	0.8836	0.8813	0.8840	0.8774	0.8771
	LPIPS $\downarrow$	0.0672	0.0685	0.0665	0.0718	0.0719
Staircase	PSNR $\uparrow$	15.28	18.98	20.25	20.59	20.67
	SSIM $\uparrow$	0.7360	0.8012	0.8300	0.8338	0.8346
	LPIPS $\downarrow$	0.1277	0.1333	0.1223	0.1194	0.1188

Table 7.3: **Effect of maximum light-path depth.** Unknown-lighting training with our full method; evaluation follows Table 7.1. Increasing depth from 2 to 3 typically improves results by enabling $\alpha(\mathbf{x})$ -guided termination and deeper transport, while gains beyond moderate depth often plateau.

Maximum path depth

Table 7.3 studies the influence of the maximum path depth parameter $k_{\max}$ during training. Depth 2 is often insufficient because it forces cache usage at the second bounce regardless of $\alpha(\mathbf{x})$, which limits the optimizer’s ability to select estimators based on reliability. Increasing to $k_{\max}=3$ enables $\alpha(\mathbf{x})$ to meaningfully influence termination and generally improves reconstruction quality. Beyond depth 3, gains often plateau, suggesting most benchmark scenes are sufficiently captured with moderate depths.

Consistent with subsection 7.4.1, Living-room-3 favors shallow depth because transport is dominated by direct illumination.

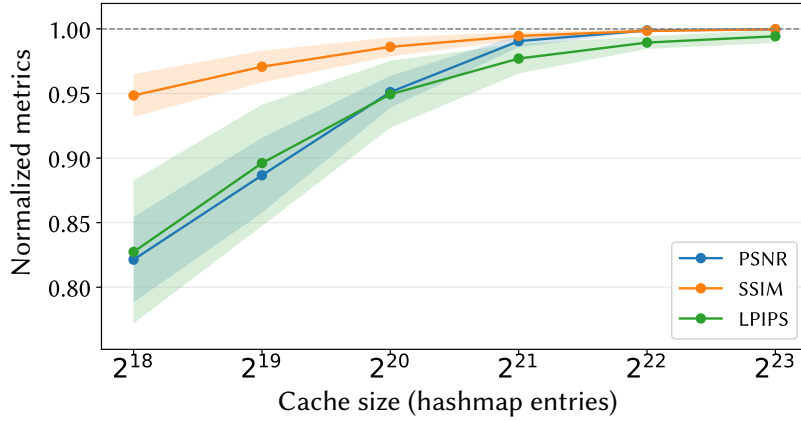


Figure 7.7: **Sensitivity to cache capacity.** Scene-normalized mean PSNR/SSIM/LPIPS performance across all scenes (best per scene = 100%) as a function of cache size (hash map entries, 2^{18} – 2^{23}). Shaded bands indicate bootstrap confidence intervals. Our method is stable across a wide range of capacities, indicating robustness to cache representation size.

Cache capacity

Figure 7.7 evaluates robustness with respect to cache capacity over a wide range of hash map sizes. Performance remains stable across capacities, indicating that the approach is not overly sensitive to the cache representation. Intuitively, when cache approximation quality degrades due to limited capacity, the learned $\alpha(\mathbf{x})$ can reduce reliance on the cache and fall back to the material estimator.

7.4.3 Known lighting material optimization

Convergence in a high-variance scene. Figure 7.8 evaluates optimization in the Veach Ajar scene under known lighting. To isolate convergence effects in a noisy global-illumination setting, we report RGB-space MSE of the painting texture (rather than full-scene texture error). Both the method in this chapter and Hadadan run at 1 spp during optimization. The blended estimator used here converges faster than pure path tracing at low spp by reducing variance and stabilizing gradients.

Limitation of material consistency loss. Figure 7.9 demonstrates a limitation of the material consistency loss (Equation 7.5), which uses the cache as a direct training target for materials. We optimize the albedo of a painting under known lighting while the cache represents outgoing radiance. For diffuse materials, limited cache capacity can blur fine details (the cache is trained on the scene scale); for glossy materials, directional variation is difficult for the cache to capture, and the resulting bias propagates to the material. This experiment motivates our main approach, which relies on separated losses to constrain estimators without requiring the cache to serve as a universally accurate direct supervision signal for outgoing radiance.

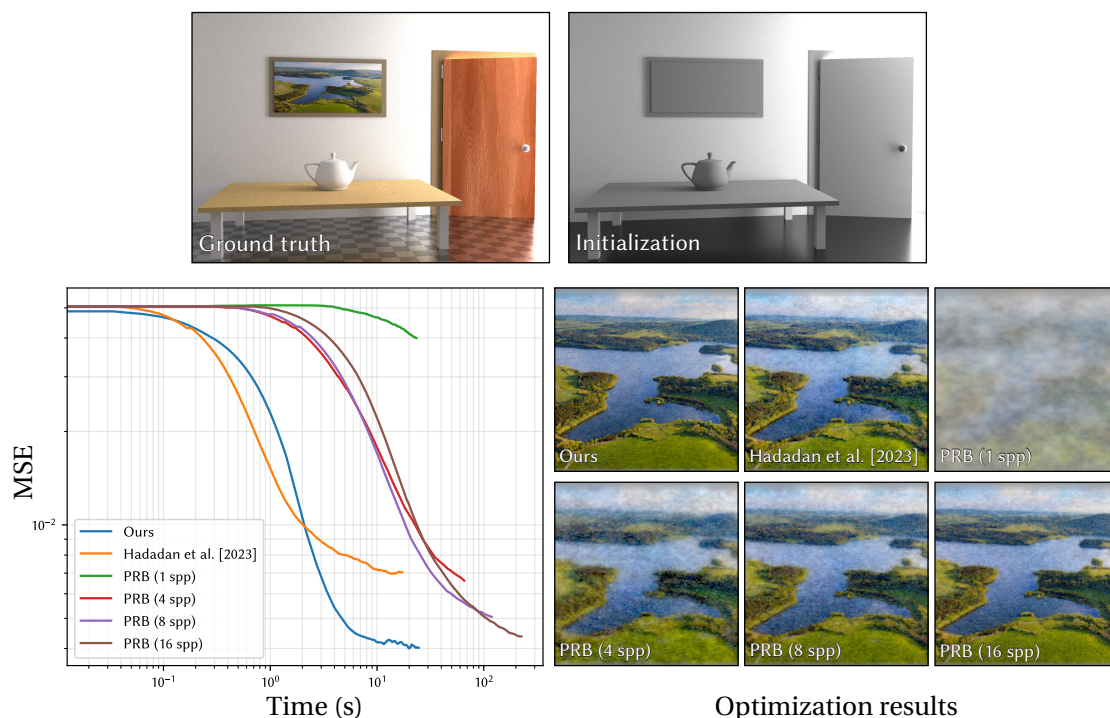


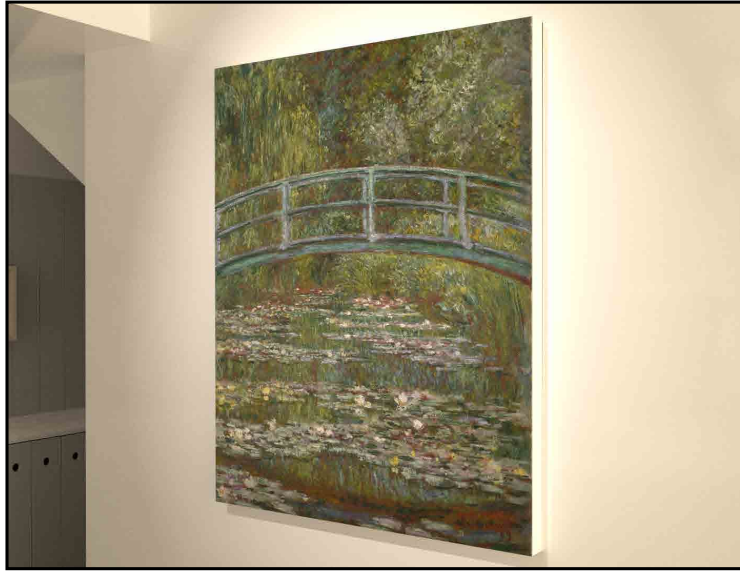
Figure 7.8: **Texture optimization in a noisy scene with known lighting.** We track convergence of the painting on the wall while jointly optimizing all scene materials. Curves show MSE versus wall-clock time for our method, Hadadan et al. [1] (1 spp), and PRB at 1/4/8/16 spp. All runs use 1000 iterations and a relative L_2 loss to avoid overfitting to noise.

7.4.4 Variance-aware optimization of $\alpha(\mathbf{x})$

The main results in this chapter optimize the blending field $\alpha(\mathbf{x})$ to minimize bias (i.e., using the decorrelated gradient estimator). Figure 7.10 explores an alternative variance-aware objective that additionally penalizes variance. On a rough dielectric surface with spatially varying roughness, variance-aware optimization drives $\alpha(\mathbf{x})$ higher overall (favoring the cache more), while still preferring the material estimator in glossy regions where cache bias is pronounced. This suggests variance-aware objectives may improve robustness in extremely noisy settings; we leave a broader evaluation for future work.

7.4.5 Optimization progress

Figure 7.11 visualizes how materials, the cache, and the blending field evolve during unknown-lighting optimization. We show (top) a *material-only* rendering relit with ground-truth lighting (unknown during optimization), (middle) cache values at camera-ray intersections, and (bottom) the learned $\alpha(\mathbf{x})$. As optimization progresses, both the material-only renderings and cache predictions become more consistent with the reference, while $\alpha(\mathbf{x})$ adapts over time to route gradients to the more reliable estimator.



Rendering with diffuse layer + clearcoat specular layer painting material

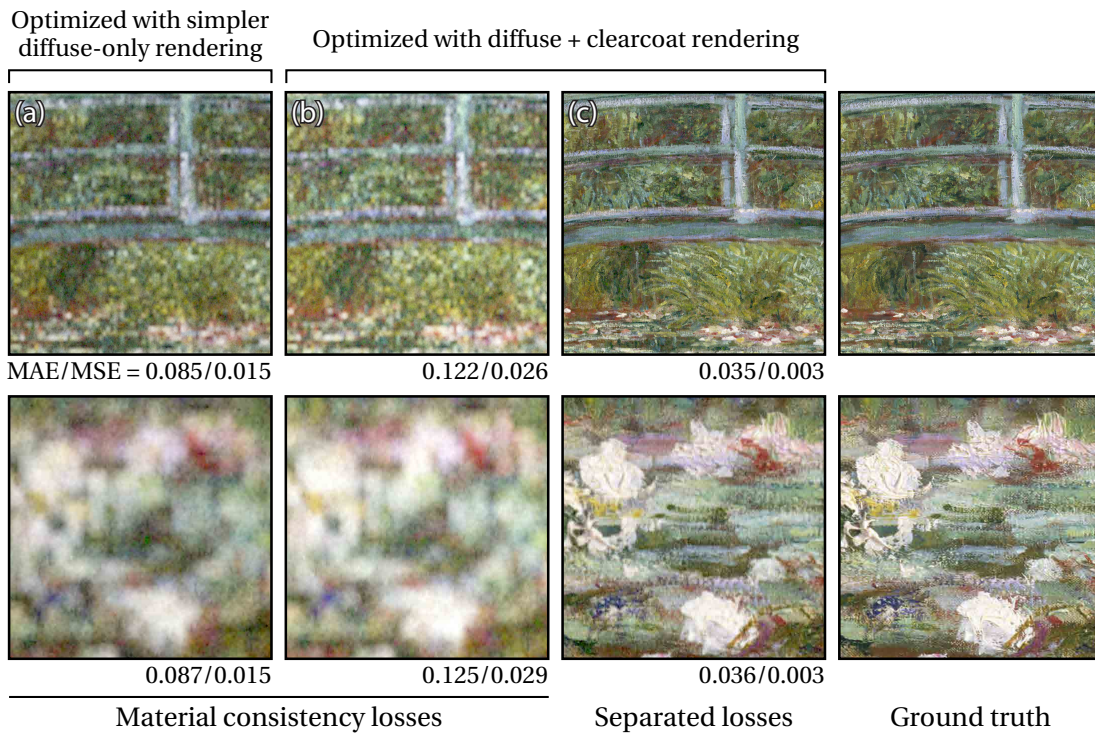


Figure 7.9: Limitation of material consistency loss. We optimize the albedo of a painting under known lighting using the material consistency loss (Equation 7.5), which uses the cache as the training target. The bottom row visualizes the optimized albedo and the ground-truth albedo. **(a)** Diffuse surface: limited cache resolution blurs fine spatial details, degrading the recovered texture. **(b–c)** Glossy surface: the cache struggles with high-frequency view-dependent variation, and bias propagates to the material, causing additional artifacts. *Bridge over a Pond of Water Lilies*, Claude Monet, 1899, public domain.

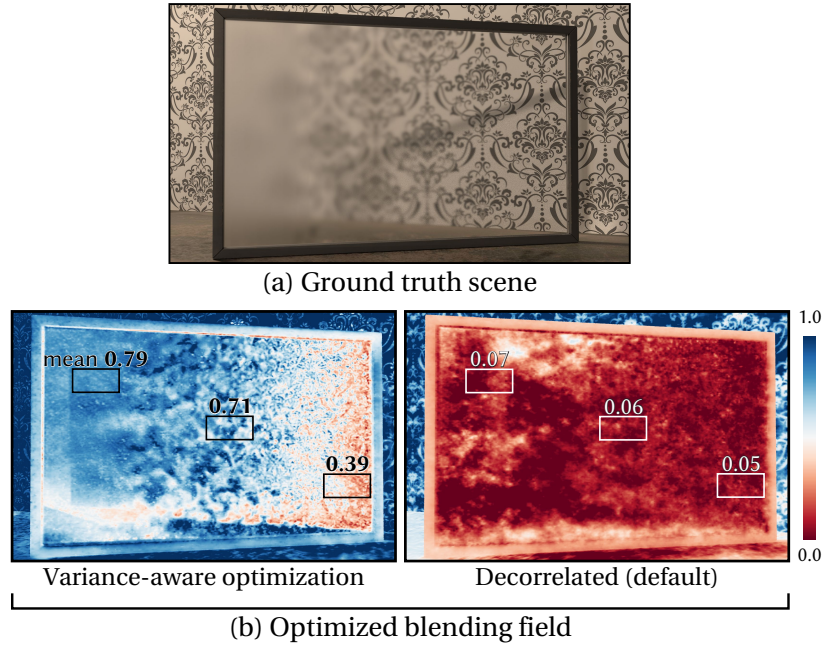


Figure 7.10: **Effect of variance-aware optimization of $\alpha(\mathbf{x})$.** (a) We optimize the blending field α on a rough dielectric with spatially varying roughness (left: diffuse, right: glossy). (b) Comparison of $\alpha(\mathbf{x})$ learned by the default bias-driven (decorrelated) optimization versus variance-aware optimization. Variance-aware optimization favors the cache more overall while still reducing cache reliance in glossy regions where cache bias is pronounced.

7.4.6 Performance analysis

The method incurs negligible computational overhead compared to PRB. Across all scenes, it requires an average of 0.094 seconds per iteration, compared to 0.091 for PRB and 0.088 for Hadadan et al. [1]. Cache evaluation and blending-field queries are efficient, and the small overhead is justified by the substantial improvement in reconstruction quality under unknown lighting.

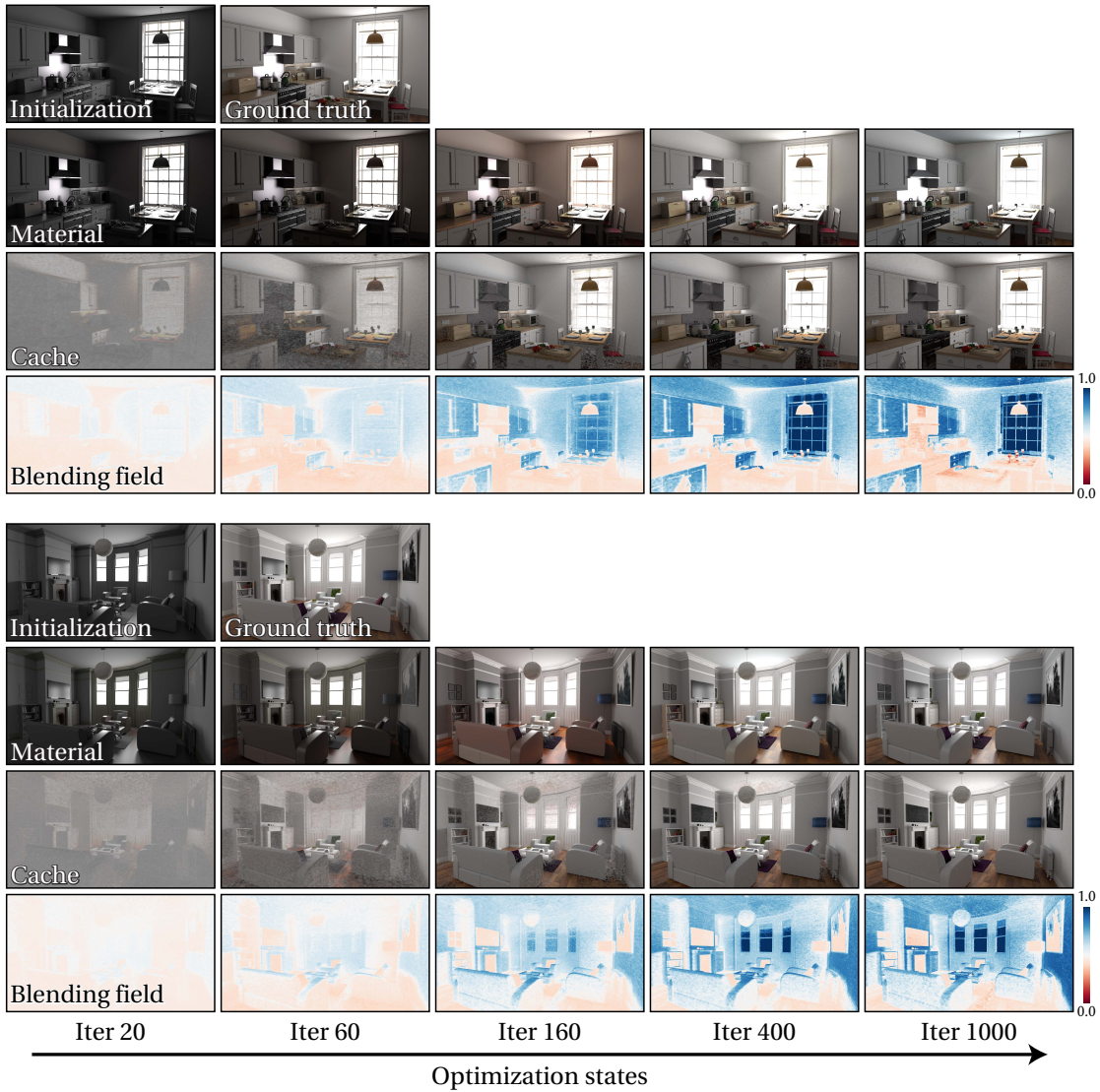


Figure 7.11: **Evolution of materials, cache, and blending field during unknown-lighting optimization.** We visualize selected iterations for each scene: (top) *material-only* renderings relit with ground-truth lighting (unknown during optimization), (middle) cache queried at camera-ray intersections, and (bottom) learned blending field $\alpha(\mathbf{x})$. The blending field adapts over time, increasingly routing gradients to the more reliable estimator as optimization progresses.



Figure 7.12: **Additional optimization progress.** Visualizations complementary to Figure 7.11, showing (top) material-only relit renderings, (middle) cache at camera-ray intersections, and (bottom) $\alpha(x)$ over iterations. The bathroom scene's warm ground-truth illumination makes the neutral-gray initialization appear tinted when rendered under that lighting.

7.5 Conclusion

This chapter explored the transport side of the thesis design space. Rather than relaxing geometry, it introduced a controllable intermediate between stored radiance and fully recursive light transport. The central observation is that a radiance cache is useful here not only as a variance-reduction device but also as a temporary optimization scaffold: it can absorb outgoing radiance that the material estimator cannot yet explain, provided that the training objective still constrains the material and cache estimators individually.

That distinction is what makes the method relevant to inverse rendering rather than only to image fitting. The separated losses allow the cache, materials, and the blending field to be optimized jointly while preserving a meaningful material-only endpoint. In the material-recovery experiments, evaluation is performed after discarding the cache and relighting with the recovered materials alone, so the gains over PRB and the neural radiosity baseline reflect improved decomposition rather than a stronger training-time blend.

Within the broader dissertation narrative, this chapter complements the previous one. *Radiance Surfaces* relaxed geometry while keeping image formation simple; this chapter instead relaxes transport during optimization while still targeting reusable physically based materials. Together, the two chapters support the thesis claim that controlled movement between the classical endpoints of inverse rendering can improve robustness without giving up interpretable scene parameters.

The assumption of known geometry is deliberately restrictive. A natural next step is to combine this transport-side relaxation with joint recovery of geometry, materials, and lighting, potentially connecting to the surface-relaxation ideas developed earlier in the thesis. Sub-surface scattering is another appealing application, since it typically has high variance and requires long light paths that could benefit from early termination.

The experiments are also synthetic in the sense that the model can reproduce the data exactly. Real photographs inevitably contain appearance effects that fall outside the chosen BSDF family and scene model. Understanding whether a cache/material decomposition can absorb some of that model mismatch without destroying the interpretability of the recovered parameters remains an interesting direction for future work.

8 Conclusion

This dissertation set out to study inverse rendering from one of its most challenging angles: the recovery of explicit surfaces under physically based light transport. In the current landscape, this is not the easiest path. Methods based on emissive volumes, neural radiance fields, and Gaussian splatting have shown extraordinary practical success. They are fast, scalable, and often remarkably effective on real data. By comparison, physically based differentiable rendering remains slower, noisier, and far more difficult to make reliable in realistic settings.

Yet physical meaning is precisely what makes these difficulties worth confronting. When the goal is not only to reproduce appearance but also to recover geometry, materials, and illumination that remain meaningful under editing, relighting, reuse, or comparison with measurements, easier approximations are not always enough. The persistent challenges of this field—discontinuous visibility, noisy estimators, and recursive transport—are not accidental obstacles surrounding the problem. They are part of the problem itself. The question is therefore not how to avoid the hard parts of light transport, but how to approach them without losing the quantities that make inverse rendering valuable in the first place.

The contributions of this thesis approach that question from several directions. For derivative computation, the issue is how to account for changes in visibility and geometry without losing the structure of physically based transport. For surface recovery, it is how to soften the optimization problem into distributions over competing hypotheses and still return an explicit surface. For transport, it is how to borrow stability from simpler image formation while continuing to estimate reusable physical parameters. In each case, the easier formulation is not the endpoint; it is a way to reach the harder one more reliably.

What emerges most clearly from these results is that inverse rendering is not well described as a choice between disconnected methods. It is better understood as a continuum of trade-offs. One important axis runs between surface and volume representations; another runs between emissive and physically based image formation. Some of the most useful formulations arise not from committing too early to one extreme, but from moving along these axes in controlled

Chapter 8. Conclusion

ways. In this sense, relaxing a surface into a continuous formulation, or simplifying transport during optimization, is not a departure from physically meaningful inverse rendering. It is one way of making it less brittle, while still returning to scene parameters that retain physical interpretation.

This perspective matters most in settings where recovered parameters must correspond to the world rather than merely reproduce an image. That is especially clear in scientific imaging, medical imaging, and fabrication tasks such as 3D printing, where the recovered structure or material properties must remain meaningful outside the image itself. But the significance of this work reaches beyond any single application area. At its core, it concerns light propagation, one of the fundamental physical processes through which the world becomes observable. For that reason, progress in physically grounded inverse rendering may also matter more broadly—anywhere light transport is used not only for depiction but also for inference, control, or verification.

This remains true in the era of generative models. Plausible images have become easy to produce, often with astonishing quality. But plausibility is not the same as explanation, decomposition, or a physical account of the scene. A convincing image is not yet a faithful scene model. For that reason, physically grounded inverse rendering retains a distinct role alongside generative methods: not only to synthesize convincing results, but to recover scene descriptions that can be interpreted, controlled, verified, and reused.

Much remains unresolved. Physically based differentiable rendering is still noisy, fragile, and far from routine on real data. The gap between controlled formulations and the complexity of real scenes remains substantial. For that reason, this dissertation should not be read as a final answer. Its contribution is smaller, but still worth making: to clarify a direction, to show that some of the field's sharpest divides are not as absolute as they first appear, and to make physically meaningful inverse rendering a little less brittle in practice.

The central aspiration is to turn images into reusable physical explanations: geometry, materials, and illumination that can be edited, relit, simulated, fabricated, or checked against other measurements. Each step toward that goal makes inverse rendering less of an image-matching tool and more of a way to infer scene structure that remains useful beyond the original photograph.

A Appendix for Projective Sampling

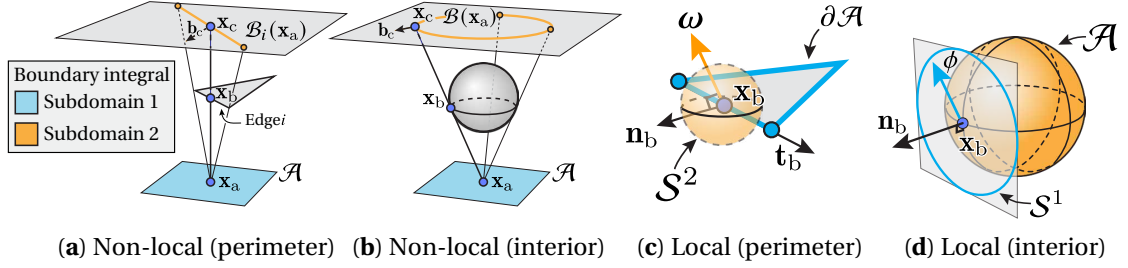


Figure A.1: Visibility-related derivatives arise from the perimeter (e.g., discrete edges of a triangle mesh) and the interiors of shapes (e.g., the smooth interior of a deformation of an ellipsoid). Path-space methods compute an integral over tangential path segments to account for them. Decomposing the integration domain (blue and orange sets) reveals different formulations: **(a)** For the perimeter component, one can integrate over source points $\mathbf{x}_a \in \mathcal{A}$ and the “shadow” $\mathbf{x}_c \in \mathcal{B}_i(\mathbf{x}_a)$ cast by a discrete edge i . **(b)** This also generalizes to the interior, but parameterizing and sampling the projected boundary $\mathcal{B}(\mathbf{x}_a)$ is difficult in general. **(c)** The local formulation instead evaluates a spherical integral at boundary points $\mathbf{x}_b \in \partial\mathcal{A}$ without explicitly considering the neighboring vertices $\mathbf{x}_a$ and $\mathbf{x}_c$. **(d)** The interior can be handled analogously but requires a different partition into an integral over surface positions (orange) and tangential directions (blue). We propose a new local boundary integral that accounts for this component.

A.1 Derivation of the local formulation

We now derive the local formulation in Equation 4.2 from the original path-space formulation in Equation 4.1.

A.1.1 Change of variables in 2D

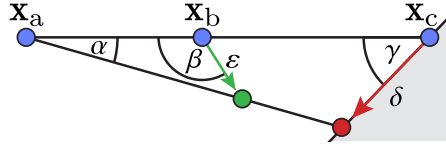
Equation (4.1) includes two *length elements* (differential arclength measurements):

1. The standard length element $dl(\mathbf{x}_c)$ of the line integral.
2. The term $\partial_\theta \mathbf{x}_c \cdot \mathbf{n}_c$, which measures the perpendicular velocity of the discontinuity at $\mathbf{x}_c$ with respect to perturbations of a scene parameter θ .

Appendix A. Appendix for Projective Sampling

Our formulation moves the integration from $\mathbf{x}_c$ to $\mathbf{x}_b$, which requires adapting both terms. We first investigate this change of variables in two dimensions and then generalize our observations to 3D.

Consider the following visualization. Translating the point $\mathbf{x}_b$ by ε units in direction β (green curve and point) produces a corresponding shift by δ units on the receiving surface (red curve and point).



Applying the law of sines twice and using the angle sum theorem relates all known quantities:

$$\frac{\sin \alpha}{\varepsilon} = \frac{\sin(\pi - \alpha - \beta)}{l_{ab}} \quad \text{and} \quad \frac{\sin \alpha}{\delta} = \frac{\sin(\pi - \alpha - \gamma)}{l_{ac}}.$$

Here, $l_{yz} = \|\mathbf{x}_y - \mathbf{x}_z\|$. Algebraically manipulating both equations isolates the angle α , which can then be equated:

$$\cot \alpha = \frac{\varepsilon^{-1} t_{ab} - \cos \beta}{\sin \beta} \quad \text{and} \quad \cot \alpha = \frac{\delta^{-1} t_{ac} - \cos \gamma}{\sin \gamma} \quad \Rightarrow \quad \frac{\varepsilon^{-1} t_{ab} - \cos \beta}{\sin \beta} = \frac{\delta^{-1} t_{ac} - \cos \gamma}{\sin \gamma}.$$

Solving for δ and evaluating the rate of change for small ε yields

$$\left. \frac{\partial \delta}{\partial \varepsilon} \right|_{\varepsilon=0} = \frac{l_{ac} \sin \beta}{l_{ab} \sin \gamma}. \quad (\text{A.1})$$

When $\mathbf{x}_b$ undergoes perpendicular displacement, we have $\beta = 90^\circ$ and the expression simplifies further to:

$$\left. \frac{\partial \delta}{\partial \varepsilon} \right|_{\varepsilon=0} = \frac{l_{ac}}{l_{ab} \sin \gamma}. \quad (\text{A.2})$$

A.1.2 Normal velocity in 3D

The previous 2D result misses an important property that appears in the general oblique case: perpendicular displacement of the discontinuity at $\mathbf{x}_b$ is generally *not perpendicular* when projected onto the receiving surface. This is apparent in Figure A.2, where the green and red arrows and points correspond to the in-plane analysis from Section A.1.1. Observe how the direction $\mathbf{p}_c$ is not parallel to the boundary normal $\mathbf{n}_c$ shown in blue.

Deriving the correct normal velocity requires converting the previous result from a velocity along $\mathbf{p}_c$ into one along $\mathbf{n}_c$. For this, we first define an orthonormal basis $\{\mathbf{t}_c, \mathbf{n}_c, \mathbf{N}_c\}$ that aligns with the boundary at $\mathbf{x}_c$:

$$\mathbf{t}_c = \frac{\mathbf{n}_b \times \mathbf{N}_c}{\|\mathbf{n}_b \times \mathbf{N}_c\|}, \quad \mathbf{n}_c = \mathbf{N}_c \times \mathbf{t}_c = \frac{\mathbf{n}_b - \mathbf{N}_c(\mathbf{n}_b \cdot \mathbf{N}_c)}{\|\mathbf{n}_b \times \mathbf{N}_c\|}. \quad (\text{A.3})$$

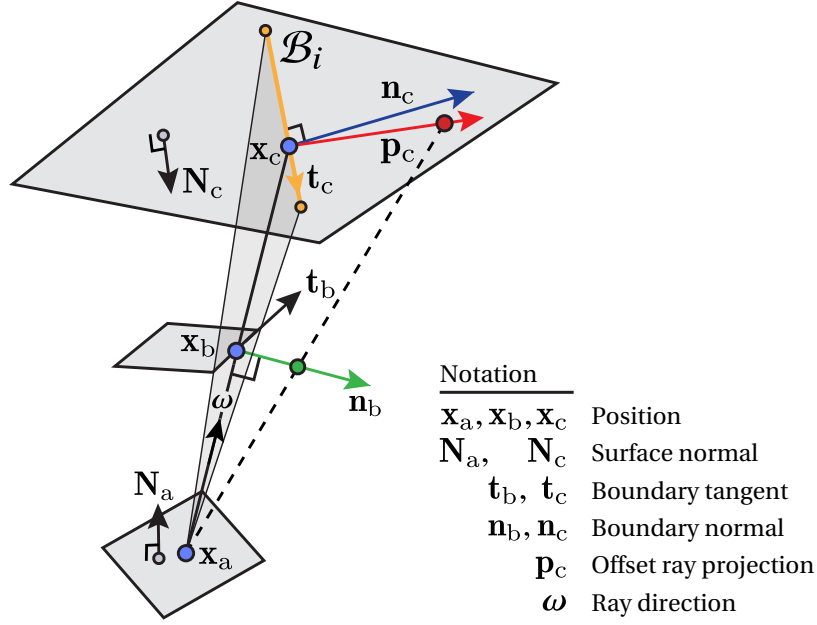


Figure A.2: Silhouette projection onto an oblique surface. Perpendicular displacement of the point $\mathbf{x}_b$ (green point and line) traces out a curve in direction $\mathbf{p}_c$ (red point and line), which is generally not perpendicular to the projected boundary with tangent $\mathbf{t}_c$ (yellow) and normal $\mathbf{n}_b$ (blue).

Perpendicular translation of the ray $\mathbf{x}_a \rightarrow \mathbf{x}_b$ at $\mathbf{x}_b$ yields intersections along $\mathbf{p}_c$ given by

$$\mathbf{p}_c = \frac{\mathbf{N}_c \times \mathbf{t}_b}{\|\mathbf{N}_c \times \mathbf{t}_b\|}. \quad (\text{A.4})$$

The projection of $\mathbf{p}_c$ onto $\mathbf{n}_c$ equals

$$\mathbf{p}_c \cdot \mathbf{n}_c = \frac{(\mathbf{N}_c \times \mathbf{t}_b) \cdot \mathbf{n}_b}{\|\mathbf{n}_b \times \mathbf{N}_c\| \|\mathbf{N}_c \times \mathbf{t}_b\|}. \quad (\text{A.5})$$

In our formulation, the boundary normal at $\mathbf{x}_b$ is (arbitrarily) chosen to be perpendicular to ω , i.e., $\mathbf{n}_b = \mathbf{t}_b \times \omega$.

In the previous 2D analysis, this corresponds to the simplified case in Equation (A.2), which requires the $\sin \gamma$ term that has the following value in our coordinate system:

$$\sin \gamma = \|\mathbf{p}_c \cdot \omega\| = \frac{(\mathbf{N}_c \times \mathbf{t}_b) \cdot \mathbf{n}_b}{\|\mathbf{N}_c \times \mathbf{t}_b\|}. \quad (\text{A.6})$$

Combining Equations (A.2), (A.5), and (A.6) yields

$$\frac{\partial_{\theta} \mathbf{x}_c \cdot \mathbf{n}_c}{\partial_{\theta} \mathbf{x}_b \cdot \mathbf{n}_b} = \frac{\partial \delta}{\partial \varepsilon} (\mathbf{p}_c \cdot \mathbf{n}_c) = \frac{l_{ac}}{l_{ab} \|\mathbf{n}_b \times \mathbf{N}_c\|}. \quad (\text{A.7})$$

Appendix A. Appendix for Projective Sampling

A.1.3 Local formulation: perimeter term

We can now apply our observations to transform Equation 4.1 into its local counterpart. This section analyzes the contribution of boundaries $\mathcal{B}_i$, while Section A.1.4 considers the interior. Our derivation uses the previously derived relationships governing the tangential (Equation A.1) and perpendicular (Equation A.7) length elements:

$$\frac{dl(\mathbf{x}_c)}{dl(\mathbf{x}_b)} = \frac{l_{ac}\|\boldsymbol{\omega} \times \mathbf{t}_b\|}{l_{ab}\|\boldsymbol{\omega} \times \mathbf{t}_c\|} \quad \text{and} \quad \frac{\partial_\theta \mathbf{x}_c \cdot \mathbf{n}_c}{\partial_\theta \mathbf{x}_b \cdot \mathbf{n}_b} = \frac{l_{ac}}{l_{ab}\|\mathbf{n}_b \times \mathbf{N}_c\|}.$$

We also use the following identity, which follows from the definition of $\mathbf{t}_c$ (Equation A.3), the vector triple product rule, and $\boldsymbol{\omega} \cdot \mathbf{n}_b = 0$:

$$\|\boldsymbol{\omega} \times \mathbf{t}_c\| = \frac{\|\boldsymbol{\omega} \times (\mathbf{n}_b \times \mathbf{N}_c)\|}{\|\mathbf{n}_b \times \mathbf{N}_c\|} = \frac{\|\mathbf{n}_b(\boldsymbol{\omega} \cdot \mathbf{N}_c) - \mathbf{N}_c(\boldsymbol{\omega} \cdot \mathbf{n}_b)\|}{\|\mathbf{n}_b \times \mathbf{N}_c\|} = \frac{|\boldsymbol{\omega} \cdot \mathbf{N}_c|}{\|\mathbf{n}_b \times \mathbf{N}_c\|}. \quad (\text{A.8})$$

We begin by converting Equation 4.1 from an integral over the projection $\mathcal{B}_i(\mathbf{x}_a)$ into one over the perimeter $\partial\mathcal{A}$:

$$\begin{aligned} \frac{\partial I}{\partial \theta} &= \int_{\mathcal{A}} \int_{\mathcal{B}_i(\mathbf{x}_a)} L_i(\mathbf{x}_a, \mathbf{x}_c) G(\mathbf{x}_a, \mathbf{x}_c) W_i(\mathbf{x}_c, \mathbf{x}_a) (\partial_\theta \mathbf{x}_c \cdot \mathbf{n}_c) dl(\mathbf{x}_c) dA(\mathbf{x}_a) \\ &= \int_{\mathcal{A}} \int_{\partial\mathcal{A}} L_i(\mathbf{x}_a, \mathbf{x}_c) G(\mathbf{x}_a, \mathbf{x}_c) W_i(\mathbf{x}_c, \mathbf{x}_a) \frac{l_{ac}\|\boldsymbol{\omega} \times \mathbf{t}_b\|}{l_{ab}\|\boldsymbol{\omega} \times \mathbf{t}_c\|} \frac{l_{ac}}{l_{ab}\|\mathbf{n}_b \times \mathbf{N}_c\|} (\partial_\theta \mathbf{x}_b \cdot \mathbf{n}_b) dl(\mathbf{x}_b) dA(\mathbf{x}_a), \end{aligned}$$

where $\mathbf{x}_c$ is now parameterized by $\mathbf{x}_a$ and $\mathbf{x}_b$. Next, we change the order of integration and perform a change of variables to turn the surface integral at $\mathbf{x}_a$ into one over solid angles at $\mathbf{x}_b$ via $dA(\mathbf{x}_a) = l_{ab}^2 / |\boldsymbol{\omega} \cdot \mathbf{N}_a| d\boldsymbol{\omega}$:

$$= \int_{\partial\mathcal{A}} \int_{\mathcal{S}^2} L_i(\mathbf{x}_b, \boldsymbol{\omega}) W_i(\mathbf{x}_c, \boldsymbol{\omega}) \underbrace{\frac{|\boldsymbol{\omega} \cdot \mathbf{N}_a| |\boldsymbol{\omega} \cdot \mathbf{N}_c|}{l_{ac}^2}}_{\text{Geometric term}} \underbrace{\frac{l_{ac}\|\boldsymbol{\omega} \times \mathbf{t}_b\|}{l_{ab}\|\boldsymbol{\omega} \times \mathbf{t}_c\|}}_{\text{Tangential}} \underbrace{\frac{l_{ac}}{l_{ab}\|\mathbf{n}_b \times \mathbf{N}_c\|}}_{\text{Perpendicular}} \underbrace{\frac{l_{ab}^2}{|\boldsymbol{\omega} \cdot \mathbf{N}_a|}}_{\mathcal{A} \rightarrow \mathcal{S}^2} (\partial_\theta \mathbf{x}_b \cdot \mathbf{n}_b) d\boldsymbol{\omega} dl(\mathbf{x}_b).$$

Many factors subsequently cancel in this expression:

$$= \int_{\partial\mathcal{A}} \int_{\mathcal{S}^2} L_i(\mathbf{x}_b, \boldsymbol{\omega}) W_i(\mathbf{x}_b, \boldsymbol{\omega}) \frac{|\boldsymbol{\omega} \cdot \mathbf{N}_c| \|\boldsymbol{\omega} \times \mathbf{t}_b\|}{\|\boldsymbol{\omega} \times \mathbf{t}_c\| \|\mathbf{n}_b \times \mathbf{N}_c\|} (\partial_\theta \mathbf{x}_b \cdot \mathbf{n}_b) d\boldsymbol{\omega} dl(\mathbf{x}_b).$$

Rewriting $\|\boldsymbol{\omega} \times \mathbf{t}_c\|$ using Equation (A.8) enables further simplification and leads to a remarkably simple result:

$$= \int_{\partial\mathcal{A}} \int_{\mathcal{S}^2} L_i(\mathbf{x}_b, \boldsymbol{\omega}) W_i(\mathbf{x}_b, \boldsymbol{\omega}) \underbrace{\|\boldsymbol{\omega} \times \mathbf{t}_b\|}_{=:\sin \phi} (\partial_\theta \mathbf{x}_b \cdot \mathbf{n}_b) d\boldsymbol{\omega} dl(\mathbf{x}_b). \quad (\text{A.9})$$

A.1.4 Local formulation: interior term

Geometric displacement of the *interior* (i.e., away from perimeters $\partial\mathcal{A}$) can also contribute to the derivative $\partial I / \partial \theta$. This case arises in the presence of curved geometry and can be safely

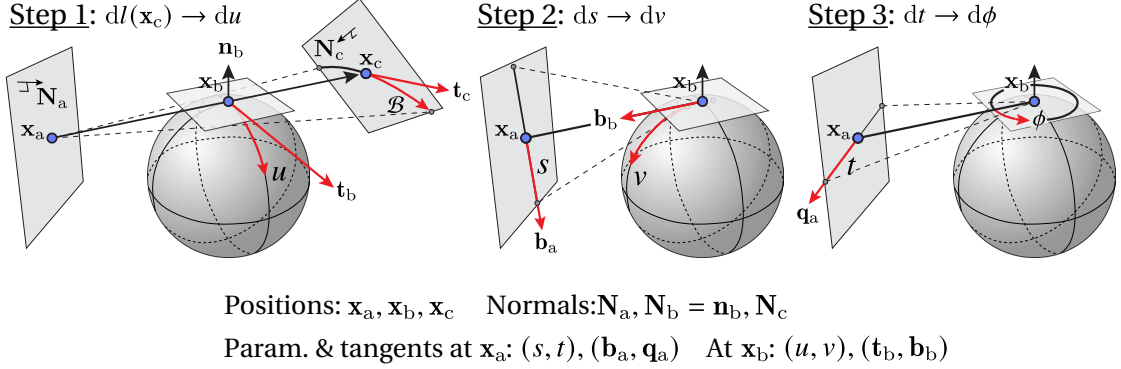


Figure A.3: The three reparameterization steps used to derive a local formulation of the interior term.

ignored when the scene consists only of polygonal meshes.

However, the previous approach of turning the distant length element $dl(\mathbf{x}_c)$ into its local counterpart $dl(\mathbf{x}_b)$ is no longer sufficient: integration must consider the full interior via an *area element* $dA(\mathbf{x}_b)$, which requires a more elaborate reparameterization across dimensions, accomplished using two local parameterizations: one at $\mathbf{x}_b$ with coordinates (u, v) , and one at $\mathbf{x}_a$ with coordinates (s, t) . The complete transformation involves three successive steps illustrated in Figure A.3:

1. The first step removes the surface at $\mathbf{x}_c$ from the equation so that the remaining steps can focus on the path segment $\mathbf{x}_a \rightarrow \mathbf{x}_b$.

This requires converting the length element on the projected boundary $dl(\mathbf{x}_c)$ into that of the local parameter du (Figure A.3, left) and transforming the boundary's normal velocity $\partial_\theta \mathbf{x}_c \cdot \mathbf{n}_c$ into $\partial_\theta \mathbf{x}_b \cdot \mathbf{n}_b$. Note that the projected normal $\mathbf{n}_b$ and surface normal $\mathbf{N}_b$ coincide in the smooth case (they were different in Section A.1.3).

2. The second step turns the length element ds at $\mathbf{x}_a$ into dv at $\mathbf{x}_b$ (Figure A.3, middle).
3. One dimension remains at this point, which controls the direction of the path segment within the tangent space at $\mathbf{x}_b$. We model this as an angle ϕ and transform dt at $\mathbf{x}_a$ into $d\phi$ (Figure A.3, right).

We choose parameterizations that align with the boundary geometry to simplify the derivation, but the final result is *intrinsic* (i.e., independent of how the surface is parameterized). In particular, we require that the (u, v) -parameterization at $\mathbf{x}_b$ has the following properties:

$$\mathbf{x}_b(0, 0) = \mathbf{x}_b, \quad \frac{\partial}{\partial u} \mathbf{x}_b(u, 0) = \mathbf{t}_b \text{ (where } \|\mathbf{t}_b\| = 1), \quad \frac{\partial}{\partial v} \mathbf{x}_b(0, v) = \mathbf{b}_b = -\boldsymbol{\omega}. \quad (\text{A.10})$$

Furthermore, we assume that $\mathbf{x}_b(u, 0)$ parameterizes the occlusion boundary $\partial \mathcal{A}(\mathbf{x}_a)$ seen from vertex $\mathbf{x}_a$ over a small neighborhood $u \in (-\varepsilon, \varepsilon)$. This boundary is defined as $\partial \mathcal{A}(\mathbf{x}_a) = \{\mathbf{x} \in$

Appendix A. Appendix for Projective Sampling

$\mathcal{A} \mid (\mathbf{x}_a - \mathbf{x}) \cdot \mathbf{N}_x = 0\}$. Note that the tangential directions $\mathbf{t}_b$ and $\mathbf{b}_b$ are generally *not* orthogonal.

The $\mathbf{x}_a(s, t)$ parameterization is chosen to satisfy

$$\mathbf{x}_a(0, 0) = \mathbf{x}_a, \quad \frac{\partial}{\partial s} \mathbf{x}_a(s, 0) = \mathbf{b}_a = \frac{\mathbf{N}_a \times (\boldsymbol{\omega} \times \mathbf{N}_b)}{\|\mathbf{N}_a \times (\boldsymbol{\omega} \times \mathbf{N}_b)\|}, \quad \frac{\partial}{\partial t} \mathbf{x}_a(0, t) = \mathbf{q}_a = \frac{\mathbf{N}_a \times \mathbf{N}_b}{\|\mathbf{N}_a \times \mathbf{N}_b\|}. \quad (\text{A.11})$$

Once more, the tangential directions $\mathbf{b}_a$ and $\mathbf{q}_a$ are generally *not* orthogonal. The following relationships will be used later and can be derived using vector cross product identities and the orthogonality of $\mathbf{N}_b$ and $\boldsymbol{\omega}$.

$$\|\mathbf{b}_a \times \mathbf{q}_a\| = \frac{|\boldsymbol{\omega} \cdot \mathbf{N}_a|}{\|\mathbf{N}_a \times (\boldsymbol{\omega} \times \mathbf{N}_b)\| \|\mathbf{N}_a \times \mathbf{N}_b\|}, \quad (\text{A.12})$$

$$\|\mathbf{b}_b \times \mathbf{q}_a\| = \frac{|\boldsymbol{\omega} \cdot \mathbf{N}_a|}{\|\mathbf{N}_a \times \mathbf{N}_b\|}, \quad (\text{A.13})$$

$$|\mathbf{b}_b \cdot \mathbf{b}_a| = \frac{|\boldsymbol{\omega} \cdot \mathbf{N}_a|}{|\mathbf{N}_a \times (\boldsymbol{\omega} \times \mathbf{N}_b)|}. \quad (\text{A.14})$$

A.1.5 Step 1: $dl(\mathbf{x}_c) \rightarrow du$ and normal velocity

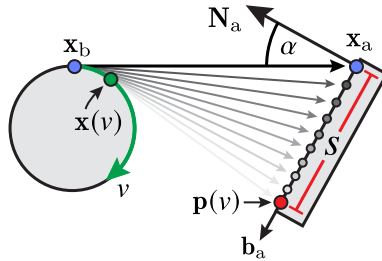
The first step is unchanged, and we can reuse the previous results from Equations (A.1) and (A.7):

$$\frac{dl(\mathbf{x}_c)}{du} = \frac{l_{ac} \|\boldsymbol{\omega} \times \mathbf{t}_b\|}{l_{ab} \|\boldsymbol{\omega} \times \mathbf{t}_c\|} \quad \text{and} \quad \frac{\partial_{\theta} \mathbf{x}_c \cdot \mathbf{n}_c}{\partial_{\theta} \mathbf{x}_b \cdot \mathbf{n}_b} = \frac{l_{ac}}{l_{ab} \|\mathbf{n}_b \times \mathbf{N}_c\|}. \quad (\text{A.15})$$

A.1.6 Step 2: $ds \rightarrow dv$

This step is easiest to understand in a 2D plane containing the point $\mathbf{x}_a$ along with the directions $\mathbf{b}_a$ and $\mathbf{b}_b$. While the parameterizations $\mathbf{x}_a(s, 0)$ and $\mathbf{x}_b(0, v)$ are not necessarily constrained to this plane, their tangents lie within it, which is enough to study the (first-order) relationship of the length elements.

Consider a tangential ray starting at $\mathbf{x}(0, v)$. Both the offset and the ray direction depend on v , though only the latter turns out to matter in this analysis. Intersecting this ray with the plane at $\mathbf{x}_a$ yields the intersection $\mathbf{p}(v)$ whose velocity we would like to compute.



This problem becomes easier to model by fixing additional (arbitrary) aspects of the boundary

geometry:

$$\mathbf{x}(v) = \begin{pmatrix} x(v) \\ y(v) \end{pmatrix}, \quad \mathbf{x}_a = \begin{pmatrix} 0 \\ l_{ab} \end{pmatrix}, \quad \mathbf{x}_b = \mathbf{x}(0) = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad \mathbf{x}'(0) = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad \mathbf{N}_a = \begin{pmatrix} \cos \alpha \\ \sin \alpha \end{pmatrix}, \quad \mathbf{b}_a = \begin{pmatrix} -\sin \alpha \\ \cos \alpha \end{pmatrix} \quad (\text{A.16})$$

The intersection with the plane at parameter value v is given by

$$p(v) = \mathbf{x}(v) - \mathbf{x}'(v) \frac{(\mathbf{x}(v) - \mathbf{x}_a) \cdot \mathbf{N}_a}{\mathbf{x}'(v) \cdot \mathbf{N}_a} \quad (\text{A.17})$$

and lies at a distance of $s(v) = \mathbf{t}_a \cdot (\mathbf{p}(v) - \mathbf{p}(0))$. The rate of change of s with respect to v is given by

$$s'(v) = \frac{(x'y'' - x''y')((l_{ab} - x)\cos \alpha - y\sin \alpha)}{(x'\cos \alpha + y'\sin \alpha)^2}. \quad (\text{A.18})$$

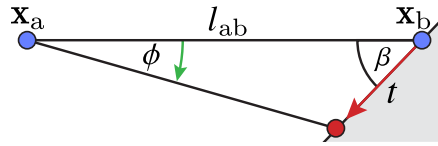
After incorporating the constraints from Equation (A.16) at $v = 0$, we have

$$\left. \frac{ds}{dv} \right|_{v=0} = \frac{l_{ab} y''}{\cos \alpha} = \frac{l_{ab} \kappa}{\cos \alpha}, \quad (\text{A.19})$$

where κ is the curvature of the plane curve at $\mathbf{x}_b$ and $\cos \alpha = |\mathbf{b}_b \cdot \mathbf{b}_a|$. Ordinarily, we would not expect to encounter curvature when deriving a line element, as it is a second-order curve property. It shows up here because rays are tangential at $\mathbf{x}(u)$: the setup therefore already involves a first derivative of position, which produces a second derivative during the subsequent differentiation.

A.1.7 Step 3: $dt \rightarrow d\phi$

The mapping resembles the setup from Section A.1.1.



Using the law of sines, we find

$$t(\phi) = \frac{l_{ab} \sin \gamma}{\sin(\phi + \beta)} \quad (\text{A.20})$$

with the following rate of change at $\phi = 0$:

$$\left. \frac{dt}{d\phi} \right|_{\phi=0} = \frac{l_{ab}}{\sin \beta}. \quad (\text{A.21})$$

The sine of the angle is $\sin \beta = \|\mathbf{b}_b \times \mathbf{q}_a\|$ in our coordinate system.

Appendix A. Appendix for Projective Sampling

A.1.8 Assembling the parts

Taking stock, the new positional/azimuthal coordinates relate to the previous integral over path segments as follows (numbers above “=” signs indicate the source equation). For $dv \leftrightarrow ds$:

$$\begin{aligned} ds &\stackrel{(A.19)}{=} \frac{l_{ab} \kappa}{|\mathbf{b}_b \cdot \mathbf{b}_a|} dv, \\ &\stackrel{(A.14)}{=} \frac{l_{ab} \kappa \|\mathbf{N}_a \times (\boldsymbol{\omega} \times \mathbf{N}_b)\|}{|\boldsymbol{\omega} \cdot \mathbf{N}_a|} dv, \end{aligned} \quad (A.22)$$

for $d\phi \leftrightarrow dt$:

$$\begin{aligned} dt &\stackrel{(A.21)}{=} \frac{l_{ab}}{\|\mathbf{b}_b \times \mathbf{q}_a\|} d\phi, \\ &\stackrel{(A.13)}{=} \frac{l_{ab} \|\mathbf{N}_a \times \mathbf{N}_b\|}{|\boldsymbol{\omega} \cdot \mathbf{N}_a|} d\phi, \end{aligned} \quad (A.23)$$

and the combination of $du \leftrightarrow dl(\mathbf{x}_c)$ with normal velocity:

$$\begin{aligned} (\partial_\theta \mathbf{x}_c \cdot \mathbf{n}_c) dl(\mathbf{x}_c) &\stackrel{(A.15)}{=} \frac{l_{ac}^2 \|\boldsymbol{\omega} \times \mathbf{t}_b\|}{l_{ab}^2 \|\boldsymbol{\omega} \times \mathbf{t}_c\| \|\mathbf{N}_b \times \mathbf{N}_c\|} du \\ &\stackrel{(A.8)}{=} \frac{l_{ac}^2 \|\boldsymbol{\omega} \times \mathbf{t}_b\|}{l_{ab}^2 |\boldsymbol{\omega} \cdot \mathbf{N}_c|} du. \end{aligned} \quad (A.24)$$

We can also express the area elements at $\mathbf{x}_b$ and $\mathbf{x}_a$:

$$dA(\mathbf{x}_b) \stackrel{(A.10)}{=} \|\mathbf{b}_b \times \mathbf{t}_b\| du dv = \|\boldsymbol{\omega} \times \mathbf{t}_b\| du dv, \quad (A.25)$$

$$\begin{aligned} dA(\mathbf{x}_a) &\stackrel{(A.11)}{=} \|\mathbf{b}_a \times \mathbf{q}_a\| ds dt \\ &\stackrel{(A.12)}{=} \frac{|\boldsymbol{\omega} \cdot \mathbf{N}_a|}{\|\mathbf{N}_a \times (\boldsymbol{\omega} \times \mathbf{N}_b)\| \|\mathbf{N}_a \times \mathbf{N}_b\|} ds dt \end{aligned} \quad (A.26)$$

$$\stackrel{(A.22, A.23)}{=} \frac{l_{ab}^2 \kappa}{|\boldsymbol{\omega} \cdot \mathbf{N}_a|} d\phi dv. \quad (A.27)$$

These can be further combined as follows:

$$(\partial_\theta \mathbf{x}_c \cdot \mathbf{n}_c) dl(\mathbf{x}_c) dA(\mathbf{x}_a) \stackrel{(A.24, A.27)}{=} \frac{l_{ac}^2 \kappa \|\boldsymbol{\omega} \times \mathbf{t}_b\|}{|\boldsymbol{\omega} \cdot \mathbf{N}_a| |\boldsymbol{\omega} \cdot \mathbf{N}_c|} d\phi du dv \quad (A.28)$$

$$\stackrel{(A.25)}{=} \frac{l_{ac}^2 \kappa}{|\boldsymbol{\omega} \cdot \mathbf{N}_a| |\boldsymbol{\omega} \cdot \mathbf{N}_c|} d\phi dA(\mathbf{x}_b). \quad (A.29)$$

Finally, we can insert this expression into Equation 4.1, where it cancels the geometric term and produces a curvature-weighted integral over surface positions and tangential directions:

$$\begin{aligned} \frac{\partial I}{\partial \theta} &= \int_{\mathcal{A}} \int_{\mathcal{B}(\mathbf{x}_a)} L_i(\mathbf{x}_a, \mathbf{x}_c) G(\mathbf{x}_a, \mathbf{x}_c) W_i(\mathbf{x}_c, \mathbf{x}_a) (\partial_\theta \mathbf{x}_c \cdot \mathbf{n}_c) dl(\mathbf{x}_c) dA(\mathbf{x}_a) \\ &\stackrel{(A.29)}{=} \int_{\mathcal{A}} \int_{\mathcal{S}^1} L_i(\mathbf{x}_b, \phi) W_i(\mathbf{x}_b, \phi) \kappa(\phi) (\partial_\theta \mathbf{x}_b \cdot \mathbf{n}_b) d\phi dA(\mathbf{x}_b). \end{aligned} \quad (A.30)$$

A.1 Derivation of the local formulation

All quantities L_i, W_i, κ above are parameterized by an (arbitrary) azimuth angle within the tangent plane at $\mathbf{x}_b$.

Equation (4.1) is based on the material form of [27], which corresponds to the UV path parameterization we discussed in Section 3.5.2. This design choice is orthogonal to the derivation, and writing it in a parameterization that uses symmetric motion across the boundary yields the result in Equation (4.2).

B Appendix for Radiance Surfaces

B.1 Derivation of our loss in NeRF form

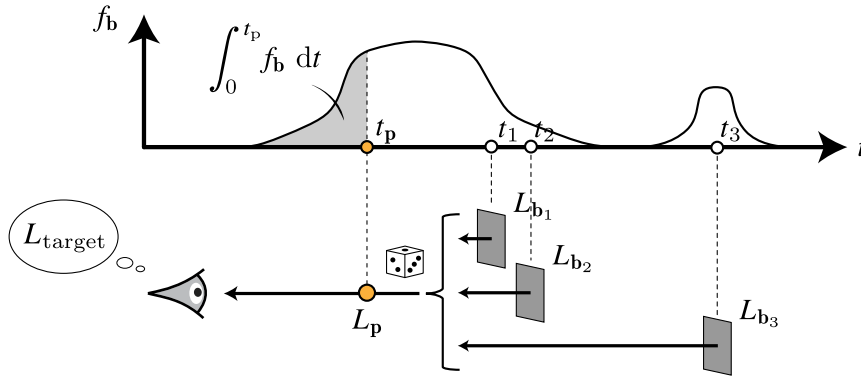


Figure B.1: **Expected background loss.** We derive an analytic expectation over all possible background surfaces for a given candidate position at t_p .

Given the radiance field loss and the stochastic background, a naive implementation of our method would first sample a surface $\mathcal{M}_b$ from a distribution to use as the perturbation background and then, for each sampled $\mathcal{M}_b$, sample multiple candidates to solve the non-local perturbation problem. Naively applying this strategy would result in an inefficient implementation. In the following, we derive an expectation of the losses over all potential background surfaces (Figure B.1) for a specific candidate.

Let f_b be the probability distribution of the background surface along a ray, where $\int_0^{t_{\max}} f_b(t) dt = 1$. Without loss of generality, we focus on one candidate position $\mathbf{p}$ at distance t_p along the ray. To avoid notational clutter, we denote the color error metric $\ell(L, L_{\text{target}})$ as $\ell(L)$. The expectation of the losses in the form of Equation 6.2 of the main text, evaluated locally at $\mathbf{p}$, is

given by:

$$\begin{aligned}
 \mathbb{E}[\mathcal{L}(\mathbf{p})] &= \int_{t_p}^{t_{\max}} \mathcal{L}(\mathbf{p}) f_{\mathbf{b}}(t) dt \\
 &= \int_{t_p}^{t_{\max}} (\alpha_{\mathbf{p}} \ell(L_{\mathbf{p}}) + (1 - \alpha_{\mathbf{p}}) \ell(L_t)) f_{\mathbf{b}}(t) dt \\
 &= \left(1 - \int_0^{t_p} f_{\mathbf{b}}(t) dt\right) \alpha_{\mathbf{p}} \ell(L_{\mathbf{p}}) + \\
 &\quad (1 - \alpha_{\mathbf{p}}) \int_{t_p}^{t_{\max}} \ell(L_t) f_{\mathbf{b}}(t) dt.
 \end{aligned} \tag{B.1}$$

Let $\mathbb{E}_{t>t_p}[\ell(L_t)]$ be the expectation of the error metric for $t > t_p$. We can rewrite the result as:

$$\mathbb{E}[\mathcal{L}(\mathbf{p})] = \underbrace{\left(1 - \int_0^{t_p} f_{\mathbf{b}}(t) dt\right)}_{\text{weight}} \left(\underbrace{\alpha_{\mathbf{p}} \ell(L_{\mathbf{p}})}_{\text{candidate}} + (1 - \alpha_{\mathbf{p}}) \underbrace{\mathbb{E}_{t>t_p}[\ell(L_t)]}_{\text{background}} \right). \tag{B.2}$$

Equation B.2 reflects an aggregated form of non-local perturbation, in which the background color is treated as an expectation over all possible background surfaces rather than a fixed value. The weight term captures the probability of selecting a background surface located behind the perturbation position. The expectation computation should not change the loss landscape, so the weight term should not be differentiated during optimization.

Next, we analyze the discrete case of the loss expectation along a ray with m sampled points, assuming the free-flight background distribution $f_{\mathbf{b}}$. For simplicity, we denote the occupancy and color at position $\mathbf{p}_i$ as α_i and L_i , respectively. The sum of all losses is given by:

$$\begin{aligned}
 \mathcal{L}_{\text{ray}} &= \sum_{i=1}^m \hat{\mathbb{E}}[\mathcal{L}(\mathbf{p}_i)] \\
 &= \sum_{i=1}^m \left[\underbrace{\left(\prod_{k=1}^{i-1} (1 - \alpha_k)\right)}_{\text{weight}} \left(\underbrace{\alpha_i \ell(L_i)}_{\text{candidate}} + (1 - \alpha_i) \underbrace{\hat{\mathbb{E}}_{t>t_i}[\ell(L_t)]}_{\text{background}} \right) \right],
 \end{aligned} \tag{B.3}$$

where

$$\hat{\mathbb{E}}_{t>t_i}[\ell(L_t)] = \sum_{j=i+1}^m \left(\prod_{t=i+1}^{j-1} (1 - \alpha_t) \right) \alpha_j \ell(L_j). \tag{B.4}$$

Not all variables in this loss function are meant to be differentiated. Specifically, the weight term and the background expectation are treated as constants in the optimization process and are excluded from differentiation (i.e., detached). To indicate which terms should be differentiated, we underline them in the derivation as $\underline{(\cdot)}$.

Below, we reformulate the loss function into a structure similar to the NeRF loss function. We

Appendix B. Appendix for Radiance Surfaces

take the notational liberty of using argmin to transform the equation so that its derivatives and minimizers are preserved, although the loss value may differ by a constant.

Equation B.3 then becomes equivalent to minimizing:

$$\begin{aligned}
 & \operatorname{argmin} \sum_{i=1}^m \left[\left(\prod_{k=1}^{i-1} (1 - \alpha_k) \right) \left(\underline{\alpha}_i \ell(\underline{L}_i) + (1 - \underline{\alpha}_i) \hat{\mathbb{E}}_{t>t_i}[\ell(L_t)] \right) \right] \\
 &= \operatorname{argmin} \underbrace{\sum_{i=1}^m \left(\prod_{k=1}^{i-1} (1 - \alpha_k) \right) \alpha_i \ell(\underline{L}_i)}_{(a)} + \underbrace{\sum_{i=1}^m \left(\prod_{k=1}^{i-1} (1 - \alpha_k) \right) \underline{\alpha}_i \ell(\underline{L}_i)}_{(b)} \\
 & \quad + \underbrace{\sum_{i=1}^m \left(\prod_{k=1}^{i-1} (1 - \alpha_k) \right) \left((1 - \underline{\alpha}_i) \hat{\mathbb{E}}_{t>t_i}[\ell(L_t)] \right)}_{(c)}, \tag{B.5}
 \end{aligned}$$

where the terms (a) and (b) arise from an application of the product rule. Reordering the double summation in term (c) yields:

$$\begin{aligned}
 (c) &\stackrel{(B.4)}{=} \sum_{i=1}^m \sum_{j=i+1}^m \left(\prod_{k=1}^{i-1} (1 - \alpha_k) \right) \left((1 - \underline{\alpha}_i) \left(\prod_{t=i+1}^{j-1} (1 - \alpha_t) \right) \alpha_j \ell(L_j) \right) \\
 &= \sum_{j=1}^m \sum_{i=1}^{j-1} \left(\prod_{k=1}^{i-1} (1 - \alpha_k) \right) \left((1 - \underline{\alpha}_i) \left(\prod_{t=i+1}^{j-1} (1 - \alpha_t) \right) \alpha_j \ell(L_j) \right). \tag{B.6}
 \end{aligned}$$

We rename the indices $i \leftrightarrow j$ and then simplify the expression to:

$$\begin{aligned}
 (B.6) &= \sum_{i=1}^m \sum_{j=1}^{i-1} \left(\prod_{k=1}^{j-1} (1 - \alpha_k) \right) \left((1 - \underline{\alpha}_j) \left(\prod_{t=j+1}^{i-1} (1 - \alpha_t) \right) \alpha_i \ell(L_i) \right) \\
 &= \sum_{i=1}^m \sum_{j=1}^{i-1} \prod_{\substack{k=1 \\ k \neq j}}^{i-1} (1 - \alpha_k) (1 - \underline{\alpha}_j) \alpha_i \ell(L_i). \tag{B.7}
 \end{aligned}$$

Combining term (c) in the form of Equation B.7 with term (b), we obtain:

$$\begin{aligned}
 (b) + (c) &= \sum_{i=1}^m \left(\prod_{k=1}^{i-1} (1 - \alpha_k) \underline{\alpha}_i + \sum_{j=1}^{i-1} \prod_{\substack{k=1 \\ k \neq j}}^{i-1} (1 - \alpha_k) (1 - \underline{\alpha}_j) \alpha_i \right) \ell(L_i) \\
 &= \sum_{i=1}^m \left(\prod_{k=1}^{i-1} (1 - \underline{\alpha}_k) \underline{\alpha}_i \ell(L_i) \right) + c_1, \tag{B.8}
 \end{aligned}$$

where c_1 is a constant. Finally, we can insert this result back into Equation B.5 to obtain a loss

in which all variables can be differentiated:

$$\begin{aligned}
 & \text{argmin } (a) + (b) + (c) \\
 \stackrel{(B.8)}{=} & \text{argmin} \sum_{i=1}^m \left(\prod_{k=1}^{i-1} (1 - \underline{\alpha}_k) \underline{\alpha}_i \ell(L_i) + \prod_{k=1}^{i-1} (1 - \alpha_k) \alpha_i \ell(\underline{L}_i) \right) \\
 = & \text{argmin} \sum_{i=1}^m \left(\prod_{k=1}^{i-1} (1 - \underline{\alpha}_k) \underline{\alpha}_i \ell(\underline{L}_i) \right). \tag{B.9}
 \end{aligned}$$

This final result is equivalent to the one shown in Figure 6.2 of the main document. This also shows that we do not need to produce additional samples to evaluate $\mathbb{E}_{t > t_p}[\ell(L_t)]$.

B.2 Design space of the background distribution

subsection 6.2.3 of the main document proposes the stochastic background surface. Designing the background surface distribution f_b (Figure B.1) involves a tradeoff between exploitation and exploration and offers a wide design space.

On one hand, choosing a background surface close to the model’s current best guess (i.e., in high-occupancy regions) ensures that only improving perturbations will be accepted. One example is the deterministic strategy of always using the 0.5 level set. It aggressively selects the first potential surface with more than 50% confidence along the ray as the background, ignoring any further possibilities.

On the other hand, exploring more possibilities enhances the algorithm’s robustness in scenes with complex occlusions. The free-flight background distribution is a softer version of the deterministic strategy. Instead of using a threshold to binarize the occupancy field, it stochastically decides whether to use a surface as the perturbation background during ray traversal.

Many other designs are possible. One design unique to our method is the *color-dependent background distribution*. Unlike the free-flight distribution, which relies only on the occupancy value, this approach also considers how well each potential surface aligns with the target color, measured by $\ell(L_p, L_{\text{target}})$. This additional information enables us to discard high-occupancy surfaces that poorly match the target color, which may result from overly aggressive optimization. Specifically, we compute an effective occupancy α' as a modification of the original occupancy α :

$$\alpha' = \frac{\alpha}{1 + c \ell(L_p, L_{\text{target}})}.$$

When the color matches well, the transformation is neutral, but for misaligned colors, it reduces the effective occupancy, lowering the likelihood of selecting such surfaces as a background. In our experiments, we used $c = 16$. As shown in Figure B.2, this color-dependent distribution can be more efficient when penetrating incorrect surfaces.

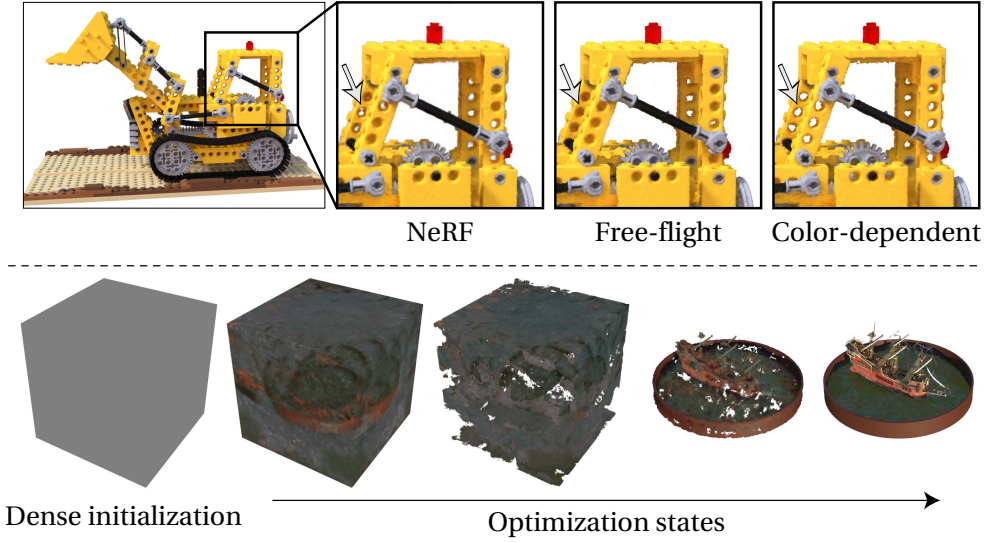


Figure B.2: **Benefits of the color-dependent background distribution.** (Top) After 2000 iterations, the color-dependent variant explores further along the ray and clears regions that were optimized too aggressively faster. (Bottom) In a contrived experiment where the scene is densely initialized, the optimizer first attempts to bake the images onto the cube. The color-dependent variant can penetrate high-occupancy regions, while others get stuck. All experiments were conducted with the same hyperparameters in the Instant NGP [68] codebase.

This expanded design space is particularly compelling. Traditional reconstruction methods mostly optimize in image space, interacting with the 3D scene only through a rendering algorithm (surface-based or volume-based). As a result, the design space in image space is quite limited, with little to do beyond computing a loss.

In contrast, our method operates directly in the scene space. Background distributions can be tailored to focus on regions of interest. This flexibility enables the development of new optimization strategies that are unattainable in image-space methods. The color-dependent background distribution is an example that actively guides the optimization to skip regions that are *believed* to be wrong, regardless of the occupancy value.

In this chapter, we focus on the free-flight distribution to highlight the dual-loss relationship with NeRF leaving the exploration of other distributions for future work. Only the result shown in Figure B.2 uses the color-dependent distribution.

B.3 Additional experiments and results

Interior topology changes Methods based on local surface evolution struggle with interior topological changes, such as transforming a sphere into a torus. This is because they primarily rely on deforming visibility silhouettes to change the overall shape, but these silhouettes often

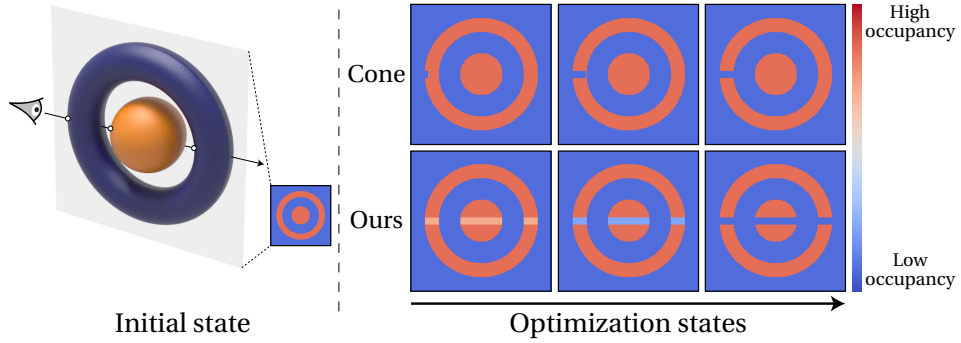
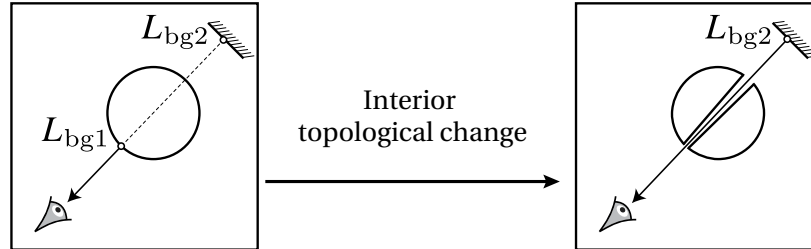


Figure B.3: **Interior topology changes.** **Left:** We test interior topological changes in a scene where orange matches the target background color better than indigo. **Right:** We show optimization states by visualizing a 2D slice of the occupancy field. The cone perturbation strategy [65] gets stuck after penetrating the torus once because it can only see through a single obstacle.

do not exist in regions away from the outer contour.

Correctly handling such topological changes requires a significant modification, such as cutting a cone through the entire object to expose the occluded background. This type of change is beyond the reach of common derivative-based methods, which can only account for infinitesimal perturbations.



Mehta et al. [65] propose a cone-shaped perturbation strategy to test whether exposing the background improves the match to the target color for physically based rendering applications. This approach significantly improves convergence in scenes that require hole penetration compared with conventional surface evolution methods.

However, this strategy can also affect scenes for which the topology is already correct. In such cases, only local refinement is needed, and the cone perturbation may bias the derivative in the wrong direction. Additionally, the cone perturbation strategy can only penetrate a single obstacle, limiting its ability to handle complex real-world scenes that require penetration through multiple layers of geometry (Figure B.3). Our stochastic background strategy addresses these challenges by considering additional background possibilities, enabling more robust optimization for complex scenes.

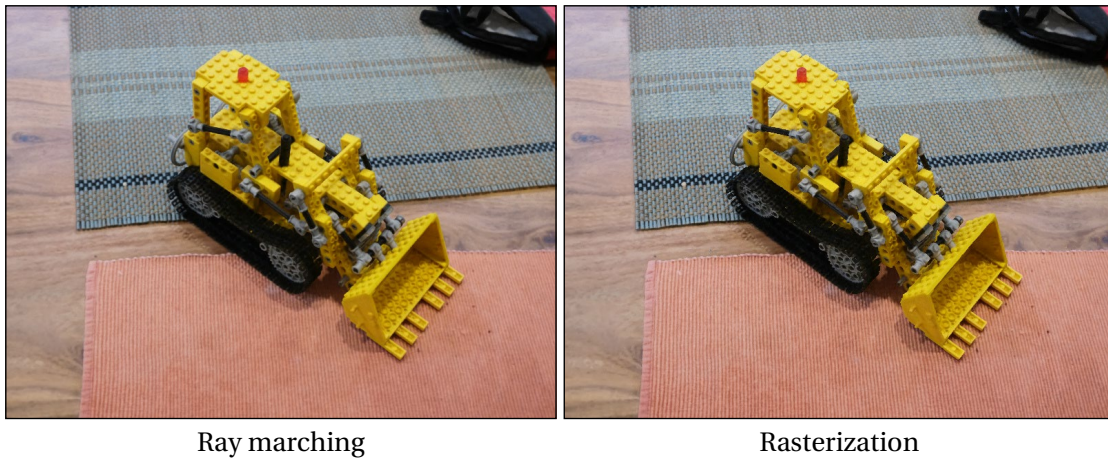


Figure B.4: **Rendering methods.** Visual comparison of the same surface scene rendered by ray marching (left) and mesh rasterization (right). Both methods produce nearly identical results.

Rendering Once the occupancy field trained with our algorithm has converged, it should have a value of 0 in empty space and 1 on the surface. Since our field representation is continuous in practice, we aim for a near-Heaviside step function on the surface. In Figure 6.11, we show an additional level set rendering result for an outdoor scene to demonstrate that our method can achieve this and that any level set can be used. In this chapter, we use 0.5 as the threshold.

We propose two methods for rendering the level set. The first method involves ray marching with a small step size. In this approach, we immediately return the color of the first sample point that hits the surface (when occupancy exceeds 0.5), without any weighting or color blending. The second method involves extracting a triangle mesh using marching cubes or TSDF fusion, then rasterizing the mesh to obtain the hit point location and querying the color network for the final color.

Both methods produce nearly identical visual results, as shown in Figure B.4.

Codebase This chapter primarily focuses on the theoretical development of a surface-based scene reconstruction algorithm, while the specifics of the model implementation are largely independent of the core algorithm. For example, the Instant NGP codebase is optimized for speed and designed for object-centric scenes, resulting in suboptimal details in the far-field background (Figure B.5). Our results inherit these advantages and limitations.

Decaying Laplacian For simple scenes with sufficient observations, Laplacian smoothing as a post-processing step can effectively refine surface geometry. However, this approach has limitations in more challenging scenarios. As shown in Figure B.6, we analyze a highly underconstrained scene, captured only from the front, with shiny surfaces that exhibit rapid

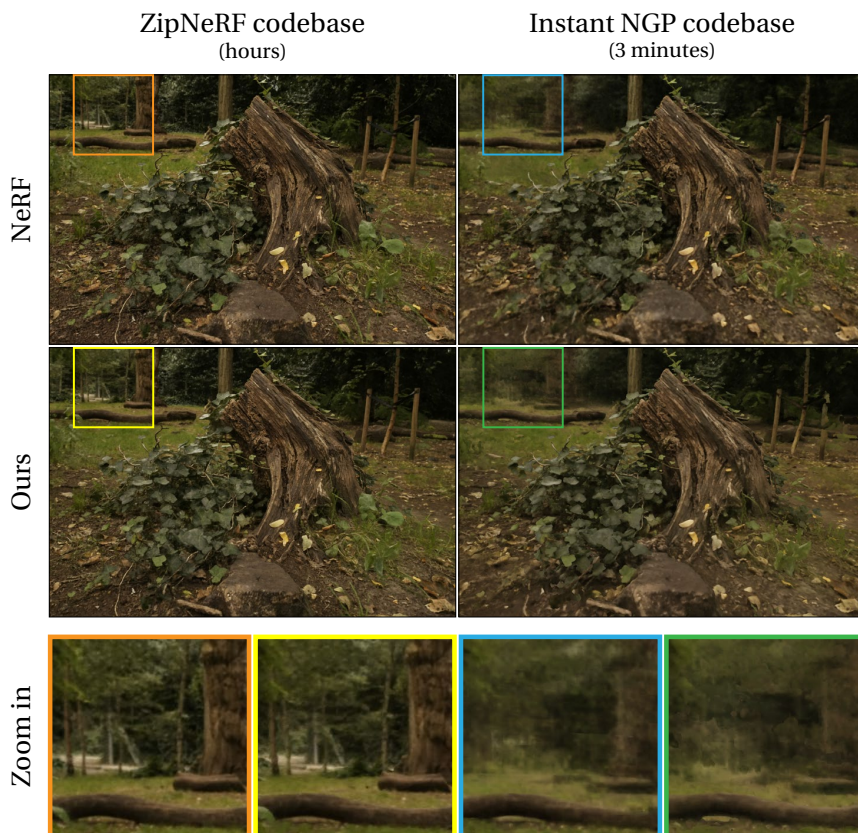


Figure B.5: **Codebase effects.** Qualitative comparison of NeRF and our method in the Instant NGP and ZipNeRF codebases, illustrating that implementation choices affect both methods.

color changes with viewing angle. Here, training without Laplacian smoothing achieves good novel-view synthesis but results in geometry errors, particularly at the bottom of the can.

Applying a Laplacian as a post-processing step requires many iterations to address these issues and may degrade geometry in other regions. In contrast, training our algorithm with an exponentially decaying Laplacian is more efficient. Consequently, the results in Figure 6.13 in the main document are obtained by training our algorithm with an exponentially decaying Laplacian.

Miscellaneous Table B.1, Table B.2, and Table B.3 show the complete PSNR, SSIM, and LPIPS results in the Instant NGP codebase. Table B.4 shows the complete PSNR results in the ZipNeRF codebase. Table B.5 shows the complete Chamfer distance results on the DTU dataset.

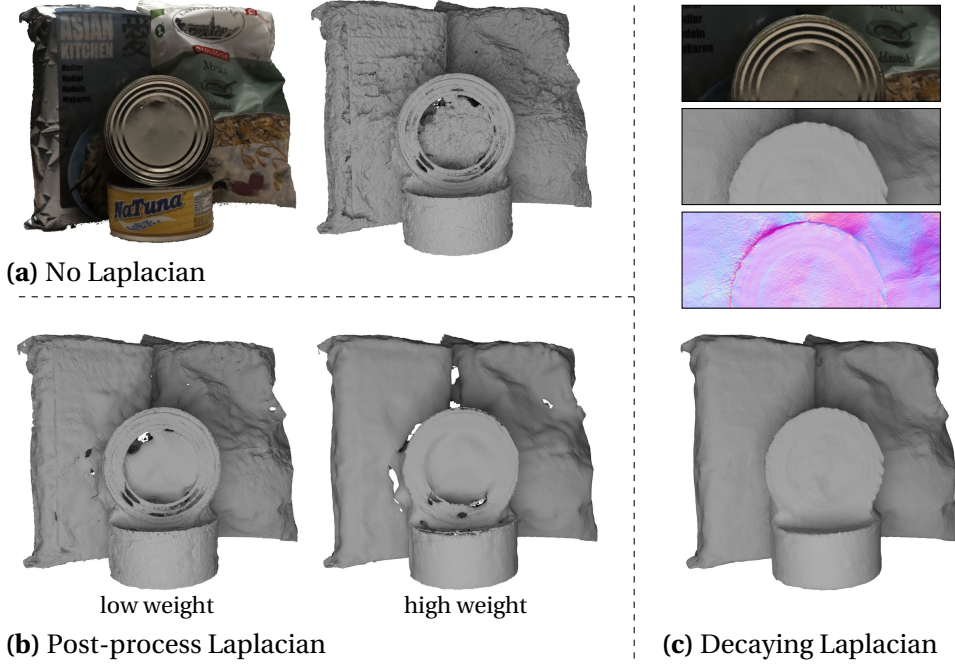


Figure B.6: **Decaying Laplacian.** For highly underconstrained scenes with shiny surfaces and limited viewing angles, training the method with an exponentially decaying Laplacian is more effective than applying the Laplacian only as a post-processing step.

B.4 Volume relaxation

This section details a heuristic volume relaxation of our method. While we do not claim this is the only way to relax our method, it provides a straightforward and effective way to retain the surface-like properties of the scene while enabling volumetric blending in regions where the surface representation is insufficient.

We propose the following loss function as a relaxed volumetric version of the loss in Equation B.3. The notation $\underline{(\cdot)}$ is consistent with section B.1, where it denotes terms that are differentiated during optimization:

$$\mathcal{L}_{\text{ray}}^{\text{vol}} = \sum_{i=1}^m \left[\left(\prod_{k=1}^{i-1} (1 - \alpha_k) \right) \ell \left(\underline{\alpha_i L_i} + (1 - \underline{\alpha_i}) \mathbb{E}_{t > t_i} [L_t], L_{\text{goal}} \right) \right], \quad (\text{B.10})$$

where the error metric ℓ now compares against a modified target color L_{goal} :

$$L_{\text{goal}} = \frac{L_{\text{target}} - L_{\text{prev}}}{T_{\text{prev}}} = \frac{L_{\text{target}} - \sum_{j=1}^{i-1} \left(\prod_{k=1}^{j-1} (1 - \alpha_k) \right) \alpha_j L_j}{\prod_{j=1}^{i-1} (1 - \alpha_j)}. \quad (\text{B.11})$$

Equation B.10 is derived from two key modifications to the radiance field loss (Equation B.3):

- We now blend colors instead of error metrics, allowing volumetric blending at the i -th sample.
- The i -th sample no longer needs to match the target color L_{target} directly. Instead, its goal adjusts for the color contribution of prior samples L_{prev} and the transmittance from the camera to the i -th sample T_{prev} .

Empirically, this relaxed loss performs well as a volume reconstruction algorithm. However, when used to refine a converged surface scene, this loss often converts the entire scene into a volumetric representation, even in regions where the surface representation is already visually adequate. This happens because a surface representation is essentially a special case of a volume with fewer degrees of freedom, and fitting colors in a volume generally reduces the loss more easily than fitting colors on a surface.

To prevent over-relaxation, we propose a heuristic to detect locations where volume relaxation is unnecessary. Specifically, when the local loss without blending at a given position is no worse than the local loss with blending:

$$\ell(L_i, L_{\text{goal}}) \leq \ell(\alpha_i L_i + (1 - \alpha_i) \mathbb{E}_{t > t_i} [L_t], L_{\text{goal}}), \quad (\text{B.12})$$

we use the local loss without blending in Equation B.10. This comparison does not introduce any overhead, as all necessary values are already available. Our experimental results show that this heuristic is effective in preserving Heaviside-like occupancy values in most areas while allowing for volumetric blending in challenging regions (Figure 6.11).

We emphasize again that the volume relaxation step is a heuristic and not a fundamental part of our method. All results are obtained without this relaxation in this chapter unless explicitly stated.

B.5 Implementation details

All results were generated and measured on a Linux workstation with an AMD Ryzen 7950X processor and an NVIDIA RTX 4090 graphics card.

Instant NGP codebase We used the default hyperparameter configuration file (base.json) provided by the authors and retained the original sampling strategy. However, we made two key modifications to the codebase to accommodate our method:

- We reduced the ray marching step size from $1/1024$ to $1/2048$ to achieve a finer surface resolution.
- The maximum buffer size for storing temporary samples was increased from $16 \times$ target batch size to $128 \times$ target batch size to accommodate the increased number of

Appendix B. Appendix for Radiance Surfaces

rays spawned in each iteration.

Since INGP does not natively support automatic differentiation, we manually implemented derivative propagation for our method in the codebase, similar to how the framework trains NeRF.

For the geometry reconstruction experiments shown in Figure 6.13 (main document), we used a L_1 loss to improve convergence in dark regions. Models were trained for 10000 iterations (reduced from the default 35000), with the Laplacian weight decaying exponentially to 2×10^{-5} . The Laplacian was estimated via finite differences using six neighboring samples with an epsilon of 1/1024 (approximately 1 mm for a unit cube).

Rendering times were measured without DLSS.

ZipNeRF codebase We used the default hyperparameter configuration file (360.gin) along with the original adaptive sampling strategy. As ZipNeRF’s adaptive sampling is tailored for volume reconstruction, it may not be optimal for our method. However, we deliberately left these components unchanged to avoid intrusive changes and keep the comparison focused on the core idea.

Warm start During training, our algorithm can sometimes push occupancy values in certain regions (e.g., peripheral or camera-adjacent areas) too high during the early stages, resulting in floaters in the final reconstruction. This occurs because the background is insufficiently explored at the beginning, leading to overly aggressive optimization of temporarily favorable candidates. Although NeRF encounters similar issues, recovery is particularly challenging in our case because the occupancy values of these floaters can approach 1.

For INGP training, we can mitigate this issue by adjusting the learning rate schedule at the cost of slower convergence. Empirically, we also found it effective to impose a moving upper bound on occupancy values, gradually relaxing this constraint during training. Specifically, at iteration i , we bound the occupancy value by $\alpha_{\max} = 0.1 + 0.9 \times i/1000$. This constraint is active only during the first 1000 iterations, which correspond to the first few seconds of training. Additionally, we observed that our relaxed training strategy is less prone to floaters. For novel view synthesis tasks, we trained the relaxed variant for 5000 iterations as a warm start.

The floater issue also arises in the ZipNeRF codebase. For simplicity, we used NeRF training as a warm start during the first 5% of training iterations and did not bound occupancy values.

Table B.1: PSNR comparison using the Instant NGP codebase. Ours uses surface rendering, while the relaxed variant and NeRF use volume rendering.

	Bicycle	Bonsai	Counter	Garden	Kitchen	Room	Stump	Flowers	Treehill
Ours	22.53	31.22	26.67	23.81	28.58	29.59	24.16	19.76	21.79
Ours (relaxed)	22.66	31.81	26.95	24.04	29.14	29.75	24.43	19.98	21.97
NeRF	22.66	31.45	26.79	23.97	29.33	29.17	23.96	19.95	21.82

Table B.2: SSIM comparison using the Instant NGP codebase.

	Bicycle	Bonsai	Counter	Garden	Kitchen	Room	Stump	Flowers	Treehill
Ours	0.673	0.918	0.872	0.686	0.866	0.896	0.769	0.577	0.692
Ours (relaxed)	0.682	0.927	0.882	0.695	0.878	0.902	0.784	0.590	0.698
NeRF	0.675	0.924	0.877	0.687	0.877	0.893	0.776	0.586	0.692

Table B.3: LPIPS comparison using the Instant NGP codebase.

	Bicycle	Bonsai	Counter	Garden	Kitchen	Room	Stump	Flowers	Treehill
Ours	0.578	0.241	0.315	0.547	0.236	0.306	0.475	0.618	0.599
Ours (relaxed)	0.642	0.244	0.335	0.672	0.234	0.324	0.497	0.676	0.645
NeRF	0.658	0.256	0.354	0.625	0.239	0.362	0.514	0.699	0.692

Table B.4: PSNR comparison using the ZipNeRF codebase.

	Bicycle	Bonsai	Counter	Garden	Kitchen	Room	Stump	Flowers	Treehill
Ours	24.10	31.24	26.38	26.14	30.22	31.07	25.96	20.99	23.12
NeRF	25.50	33.20	28.16	27.62	32.01	32.44	27.11	22.11	23.85

Table B.5: Chamfer Distance comparison on the DTU dataset with NeuS [66] and NeuS2 [71].

	Scan24	Scan37	Scan40	Scan55	Scan63	Scan65	Scan69	Scan83
Ours (1 minute)	0.81	0.77	0.66	0.40	1.08	0.90	0.88	1.42
NeuS (8 hours)	0.83	0.98	0.56	0.37	1.13	0.59	0.60	1.45
NeuS2 (5 minutes)	0.56	0.76	0.49	0.37	0.92	0.71	0.76	1.22
	Scan97	Scan105	Scan106	Scan110	Scan114	Scan118	Scan122	
Ours	1.20	0.75	0.68	1.07	0.61	0.55	0.63	
NeuS	0.95	0.78	0.52	1.43	0.36	0.45	0.45	
NeuS2	1.08	0.63	0.59	0.89	0.40	0.48	0.55	

C Appendix for Radiance Caching

C.1 Decorrelated gradient estimator

In Equation 7.3.1, we claimed that naively optimizing the material consistency loss leads to biased gradients. The issue has the same origin as the bias that arises when differentiating Monte Carlo renderers with noisy estimates, and we adopt a decorrelation strategy similar to that used by Azinović et al. [90].

Consider the material consistency loss at a single bounce. For brevity, let $C = C(\mathbf{x}, \omega)$ be the target outgoing radiance and $C'(\omega') = C(\mathbf{x}', -\omega')$ be the incident radiance from the cache. The loss is

$$\mathcal{L} = \left(C - \mathbb{E}_{\omega'} \left[C'(\omega') \cdot f(\omega') \cdot \frac{\cos \theta'}{p(\omega')} \right] \right)^2, \quad (\text{C.1})$$

where C and C' are treated as known (from the detached cache), and we optimize the BSDF $f(\omega') \equiv f(\mathbf{x}, \omega', \omega)$. Let $R = C - \mathbb{E}[C' \cdot f \cdot \cos \theta' / p]$ denote the residual. The true gradient is

$$\nabla_f \mathcal{L} = -2 R \cdot \mathbb{E} \left[\nabla_f \left(C' \cdot f \cdot \frac{\cos \theta'}{p} \right) \right]. \quad (\text{C.2})$$

Naive estimator (biased). If we use the same sample ω'_1 for both the residual and the gradient, we obtain

$$\widehat{\nabla_f \mathcal{L}} = -2 \left(C - C'(\omega'_1) \cdot f(\omega'_1) \cdot \frac{\cos \theta'_1}{p_1} \right) \cdot \nabla_f \left(C'(\omega'_1) \cdot f(\omega'_1) \cdot \frac{\cos \theta'_1}{p_1} \right). \quad (\text{C.3})$$

The two factors are correlated through the shared sample ω'_1 , so $\mathbb{E}[A \cdot B] \neq \mathbb{E}[A] \cdot \mathbb{E}[B]$. This estimator is biased: in the extreme single-sample case, it asks each sampled direction to match C individually, whereas only the integral over all directions should match C .

Decorrelated estimator (unbiased). To obtain an unbiased gradient, we sample two independent directions: ω'_1 for the residual and ω'_2 for the gradient:

$$\widehat{\nabla_f \mathcal{L}} = -2 \left(C - C'(\omega'_1) \cdot f(\omega'_1) \cdot \frac{\cos \theta'_1}{p_1} \right) \cdot \nabla_f \left(C'(\omega'_2) \cdot f(\omega'_2) \cdot \frac{\cos \theta'_2}{p_2} \right). \quad (\text{C.4})$$

Since ω'_1 and ω'_2 are independent, the expectation factorizes correctly:

$$\mathbb{E} \left[\widehat{\nabla_f \mathcal{L}} \right] = -2 \mathbb{E}[R] \cdot \mathbb{E} \left[\nabla_f \left(C' \cdot f \cdot \frac{\cos \theta'}{p} \right) \right] = \nabla_f \mathcal{L}. \quad (\text{C.5})$$

C.2 Alpha optimization via blended termination losses

In subsection 7.3.3, we described optimizing α by differentiating the blended image loss. Here, we discuss an alternative approach that optimizes α more directly, without adding overhead. The idea, used by Zhang et al. [6], is to define two losses at each bounce—one assuming that we terminate using the cache and one assuming that we continue using the material—and let α blend between them.

At each bounce i along a light path, we compare the pixel color with the radiance estimate obtained by each strategy:

$$\mathcal{L}_{\text{cache},i} = (I - T_i \cdot C_i)^2, \quad (\text{C.6})$$

$$\mathcal{L}_{\text{mat},i} = \left(I - T_i \cdot \frac{f_i \cos \theta_i}{p_i} L_o^{(i+1)} \right)^2, \quad (\text{C.7})$$

where T_i is the path throughput from the camera to bounce i , and I is the reference pixel color. We then optimize α_i to minimize the blended loss:

$$\mathcal{L}_{\text{blend},i} = (1 - \alpha_i) \mathcal{L}_{\text{mat},i} + \alpha_i \mathcal{L}_{\text{cache},i}. \quad (\text{C.8})$$

Intuitively, α_i learns to favor whichever estimator yields a smaller loss at that point.

Optimizing α . This formulation gives unbiased gradients for α . To see why, observe that the gradient is simply a difference of two loss values:

$$\nabla_{\alpha_i} \mathcal{L}_{\text{blend},i} = \mathcal{L}_{\text{cache},i} - \mathcal{L}_{\text{mat},i}. \quad (\text{C.9})$$

Since α_i lies outside the nonlinear (squared) part of the loss, there is no product of correlated terms. Each loss value is an unbiased estimate of its expectation, so:

$$\mathbb{E} [\nabla_{\alpha_i} \mathcal{L}_{\text{blend},i}] = \mathbb{E} [\mathcal{L}_{\text{cache},i}] - \mathbb{E} [\mathcal{L}_{\text{mat},i}]. \quad (\text{C.10})$$

The gradient pushes α_i toward the estimator with lower expected loss.

Appendix C. Appendix for Radiance Caching

Limitations for cache and material. One might hope to use these losses to optimize the cache or material parameters as well. However, this leads to biased gradients—even for the cache, which itself has no estimation variance.

The issue is the path throughput T_i . Consider the gradient for the cache:

$$\nabla_{C_i} \mathcal{L}_{\text{cache},i} = -2 T_i \cdot (I - T_i \cdot C_i). \quad (\text{C.11})$$

The throughput T_i appears twice: once as a direct multiplier and once inside the residual. Both instances use the same one-sample estimate, so they are correlated, and the expectation does not factorize correctly. This is the same bias issue discussed in section C.1.

To obtain unbiased gradients, we would need two independent paths connecting the same pixel to the same point $\mathbf{x}_i$ —one for the residual and one for the gradient multiplier. This is not efficient in practice, so we conclude that these blended termination losses should only be used to optimize α , not the cache or material parameters.

Bias-only vs. variance-aware. As with the image-space losses (subsection 7.3.3), we can optimize α for bias alone or for combined bias and variance.

For variance-aware optimization, we compute $\mathcal{L}_{\text{cache},i}$ and $\mathcal{L}_{\text{mat},i}$ with a single sample. The expected squared error includes both bias and variance:

$$\mathbb{E}[(I - T \cdot R)^2] = \text{bias}^2 + \text{variance}, \quad (\text{C.12})$$

so α is pushed toward the estimator with lower mean squared error.

For bias-only optimization, we split the squared loss into two terms estimated with independent samples A and B :

$$\mathcal{L}_{\text{cache},i} = (I - T_i^A \cdot C_i) \cdot (I - T_i^B \cdot C_i). \quad (\text{C.13})$$

Since A and B are independent, the expectation factorizes:

$$\mathbb{E}[(I - T^A \cdot R)(I - T^B \cdot R)] = (I - \mathbb{E}[T \cdot R])^2 = \text{bias}^2, \quad (\text{C.14})$$

removing the variance term from the optimization objective.

C.3 Termination strategies for separated losses

The separated loss (subsection 7.3.2) requires choosing where to terminate paths in each iteration. Here, we describe these strategies in detail.

Uniform depth sampling The simplest approach samples a termination depth k uniformly from $\{1, \dots, k_{\text{max}}\}$ at each iteration. All pixels share the same k , producing a coherent image for

the loss. This baseline is easy to implement and works well in practice.

α -aware termination Uniform sampling may terminate paths where α is low, indicating that the cache is unreliable. To mitigate this, we extend paths beyond the sampled depth deterministically until they reach a more reliable region:

1. Sample a minimum depth k uniformly at random.
2. Starting from bounce k , compute the accumulated product $\prod_{i \geq k} (1 - \alpha_i)$.
3. Terminate at the first bounce where this product drops below 0.5; no additional random number is drawn.

Intuitively, we continue until enough “trust in the cache” has accumulated along the path.

Throughput-based termination An alternative strategy measures accumulated trust from the start of the path. We compute the product $(1 - \alpha_1)(1 - \alpha_2) \cdots (1 - \alpha_{k-1})$, which represents how strongly the path has favored the material estimator. At each iteration, we draw a threshold $\tau \in [0, 1]$ and terminate at the first bounce where this product drops below τ . This naturally biases termination toward high- α regions.

Comparison In our benchmarks, uniform depth sampling performs approximately 0.4 dB worse than the α -aware strategies. The α -aware and throughput-based strategies perform comparably, with no distinguishable difference between them. We use α -aware termination for all main results.

Computing all losses simultaneously Rather than sampling one termination depth, we can compute losses at all depths in a single pass by storing each $\hat{I}_k$ in a separate output buffer. The total loss is then $\sum_k w_k \mathcal{L}_k$ with weights w_k . This avoids sampling variance but requires $O(\text{depth})$ memory per pixel. We use the sampling approach in all experiments for its lower memory footprint.

Per-ray vs. per-iteration termination The termination criterion should be decided once per iteration, not independently per ray. If termination were decided per ray—for example, by terminating at bounce k with probability α_k —the expected image with infinitely many samples would recover the blended estimator from Equation 7.3, losing the direct supervision that separated losses provide.

Bibliography

- [1] Saeed Hadadan, Geng Lin, Jan Novák, Fabrice Rousselle, and Matthias Zwicker. Inverse global illumination using a neural radiometric prior. In *ACM SIGGRAPH 2023 Conference Proceedings*, SIGGRAPH '23, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400701597. doi: 10.1145/3588432.3591553. URL <https://doi.org/10.1145/3588432.3591553>.
- [2] 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.
- [3] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkuehler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.*, 42(4), July 2023. ISSN 0730-0301. doi: 10.1145/3592433. URL <https://doi.org/10.1145/3592433>.
- [4] Ziyi Zhang, Nicolas Roussel, and Wenzel Jakob. Projective sampling for differentiable rendering of geometry. *ACM Trans. Graph.*, 42(6), December 2023. ISSN 0730-0301. doi: 10.1145/3618385. URL <https://doi.org/10.1145/3618385>.
- [5] Ziyi Zhang, Nicolas Roussel, and Wenzel Jakob. Many-worlds inverse rendering. *ACM Trans. Graph.*, 45(1), October 2025. ISSN 0730-0301. doi: 10.1145/3767318. URL <https://doi.org/10.1145/3767318>.
- [6] Ziyi Zhang, Nicolas Roussel, Thomas Muller, Tizian Zeltner, Merlin Nimier-David, Fabrice Rousselle, and Wenzel Jakob. Radiance surfaces: Optimizing surface representations with a 5d radiance field loss. In *Proceedings of the Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers*, SIGGRAPH Conference Papers '25, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400715402. doi: 10.1145/3721238.3730713. URL <https://doi.org/10.1145/3721238.3730713>.
- [7] Ziyi Zhang, Delio Vicini, Sebastian Winberg, Stephan Garbin, and Wenzel Jakob. Radiance caching for differentiable path tracing. *ACM Trans. Graph.*, 45(4), July 2026. doi: 10.1145/3811398. URL <https://doi.org/10.1145/3811398>.
- [8] Zichen Wang, Xi Deng, Ziyi Zhang, Wenzel Jakob, and Steve Marschner. A simple approach to differentiable rendering of sdf. In *SIGGRAPH Asia 2024 Conference Papers*, SA '24, New York, NY, USA, 2024. Association for Computing Machinery. ISBN

Bibliography

9798400711312. doi: 10.1145/3680528.3687573. URL <https://doi.org/10.1145/3680528.3687573>.
- [9] Lovro Nuic, Ziyi Zhang, Korbinian Sager, and Wenzel Jakob. Inverse rendering for discrete x-ray computed tomography. *ACM Trans. Graph.*, 45(4), July 2026. doi: 10.1145/3811391. URL <https://doi.org/10.1145/3811391>.
- [10] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation (3rd ed.)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, October 2016. ISBN 9780128006450.
- [11] Eric Veach. *Robust Monte Carlo Methods for Light Transport Simulation*. PhD thesis, Stanford University, Stanford, CA, December 1997.
- [12] Eric Veach and Leonidas J. Guibas. Optimally combining sampling techniques for monte carlo rendering. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*. Association for Computing Machinery, 1995.
- [13] Bruce Walter, Stephen R Marschner, Hongsong Li, and Kenneth E Torrance. Microfacet models for refraction through rough surfaces. *Rendering techniques*, 2007:18th, 2007.
- [14] Louis G Henyey and Jesse Leonard Greenstein. Diffuse radiation in the galaxy. *Astrophysical Journal*, vol. 93, p. 70-83 (1941)., 93:70–83, 1941.
- [15] Shuang Zhao, Ravi Ramamoorthi, and Kavita Bala. High-order similarity relations in radiative transfer. *ACM Trans. Graph.*, 33(4), July 2014. ISSN 0730-0301. doi: 10.1145/2601097.2601104. URL <https://doi.org/10.1145/2601097.2601104>.
- [16] Jan Novák, Iliyan Georgiev, Johannes Hanika, and Wojciech Jarosz. Monte Carlo methods for volumetric light transport simulation. *Computer Graphics Forum (Proceedings of Eurographics - State of the Art Reports)*, 37(2), May 2018. doi: 10/gd2jqj.
- [17] Wenzel Jakob, Adam Arbree, Jonathan T. Moon, Kavita Bala, and Steve Marschner. A radiative transfer framework for rendering materials with anisotropic structure. *ACM Trans. Graph.*, 29(4), jul 2010. ISSN 0730-0301. doi: 10.1145/1778765.1778790. URL <https://doi.org/10.1145/1778765.1778790>.
- [18] Eric Heitz, Jonathan Dupuy, Cyril Crassin, and Carsten Dachsbacher. The sggx microflake distribution. *ACM Transactions on Graphics (TOG)*, 34(4):1–11, 2015.
- [19] Jonathan Dupuy, Eric Heitz, and Eugene d’Eon. Additional progress towards the unification of microfacet and microflake theories. In *EGSR (EI&I)*, pages 55–63, 2016.
- [20] Matthias Seeger. Gaussian processes for machine learning. *International journal of neural systems*, 14(02):69–106, 2004.
- [21] Oliver Williams and Andrew Fitzgibbon. Gaussian process implicit surfaces. In *Gaussian Processes in Practice*, 2006.

-
- [22] Dario Seyb, Eugene d'Eon, Benedikt Bitterli, and Wojciech Jarosz. From microfacets to participating media: A unified theory of light transport with stochastic geometry. *ACM Transactions on Graphics (TOG)*, 43(4):1–17, 2024.
- [23] Bailey Miller, Hanyu Chen, Alice Lai, and Ioannis Gkioulekas. Objects as volumes: A stochastic geometry view of opaque solids. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 87–97, June 2024.
- [24] Kehan Xu, Benedikt Bitterli, Eugene d'Eon, and Wojciech Jarosz. Practical gaussian process implicit surfaces with sparse convolutions. *ACM Trans. Graph.*, 44(6), December 2025. ISSN 0730-0301. doi: 10.1145/3763329. URL <https://doi.org/10.1145/3763329>.
- [25] Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. Differentiable monte carlo ray tracing through edge sampling. *ACM Trans. Graph.*, 37(6), December 2018. ISSN 0730-0301. doi: 10.1145/3272127.3275109. URL <https://doi.org/10.1145/3272127.3275109>.
- [26] Cheng Zhang, Lifan Wu, Changxi Zheng, Ioannis Gkioulekas, Ravi Ramamoorthi, and Shuang Zhao. A differential theory of radiative transfer. *ACM Trans. Graph.*, 38(6), November 2019. ISSN 0730-0301. doi: 10.1145/3355089.3356522. URL <https://doi.org/10.1145/3355089.3356522>.
- [27] Cheng Zhang, Bailey Miller, Kai Yan, Ioannis Gkioulekas, and Shuang Zhao. Path-space differentiable rendering. *ACM Trans. Graph.*, 39(4), August 2020. ISSN 0730-0301. doi: 10.1145/3386569.3392383. URL <https://doi.org/10.1145/3386569.3392383>.
- [28] Merlin Nimier-David, Sébastien Speierer, Benoît Ruiz, and Wenzel Jakob. Radiative backpropagation: an adjoint method for lightning-fast differentiable rendering. *ACM Trans. Graph.*, 39(4), August 2020. ISSN 0730-0301. doi: 10.1145/3386569.3392406. URL <https://doi.org/10.1145/3386569.3392406>.
- [29] Delio Vicini, Sébastien Speierer, and Wenzel Jakob. Path replay backpropagation: differentiating light paths using constant memory and linear time. *ACM Trans. Graph.*, 40(4), July 2021. ISSN 0730-0301. doi: 10.1145/3450626.3459804. URL <https://doi.org/10.1145/3450626.3459804>.
- [30] Tizian Zeltner, Sébastien Speierer, Iliyan Georgiev, and Wenzel Jakob. Monte carlo estimators for differential light transport. *ACM Trans. Graph.*, 40(4), July 2021. ISSN 0730-0301. doi: 10.1145/3450626.3459807. URL <https://doi.org/10.1145/3450626.3459807>.
- [31] Merlin Nimier-David, Delio Vicini, Tizian Zeltner, and Wenzel Jakob. Mitsuba 2: a retargetable forward and inverse renderer. *ACM Trans. Graph.*, 38(6), November 2019. ISSN 0730-0301. doi: 10.1145/3355089.3356498. URL <https://doi.org/10.1145/3355089.3356498>.
- [32] Ingo Wald, Sven Woop, Carsten Benthin, Gregory S. Johnson, and Manfred Ernst. Embree: a kernel framework for efficient cpu ray tracing. *ACM Trans. Graph.*, 33(4), July

Bibliography

2014. ISSN 0730-0301. doi: 10.1145/2601097.2601199. URL <https://doi.org/10.1145/2601097.2601199>.
- [33] 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 SIGGRAPH 2010 Papers*, SIGGRAPH '10, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781450302104. doi: 10.1145/1833349.1778803. URL <https://doi.org/10.1145/1833349.1778803>.
- [34] Tomas Möller and Ben Trumbore. Fast, minimum storage ray/triangle intersection. In *ACM SIGGRAPH 2005 Courses*, SIGGRAPH '05, page 7–es, New York, NY, USA, 2005. Association for Computing Machinery. ISBN 9781450378338. doi: 10.1145/1198555.1198746. URL <https://doi.org/10.1145/1198555.1198746>.
- [35] Andreas Griewank and Andrea Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*, volume 105. SIAM, 2008.
- [36] Ioannis Gkioulekas, Anat Levin, and Todd Zickler. An evaluation of computational imaging techniques for heterogeneous inverse scattering. In *European Conference on Computer Vision (ECCV)*, pages 685–701. Springer International Publishing, 2016. doi: 10.1007/978-3-319-46487-9_42.
- [37] Osborne Reynolds. *Papers on Mechanical and Physical Subjects: The Sub-Mechanics of the Universe*, volume 3 of *Collected Works*. Cambridge University Press, Cambridge, 1903.
- [38] Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. Differentiable monte carlo ray tracing through edge sampling. *ACM Transactions on Graphics (TOG)*, 37(6): 1–11, 2018.
- [39] Kai Yan, Christoph Lassner, Brian Budge, Zhao Dong, and Shuang Zhao. Efficient estimation of boundary integrals for path-space differentiable rendering. *ACM Transactions on Graphics (TOG)*, 41(4):1–13, 2022.
- [40] Guillaume Loubet, Nicolas Holzschuch, and Wenzel Jakob. Reparameterizing discontinuous integrands for differentiable rendering. *ACM Trans. Graph.*, 38(6), November 2019. ISSN 0730-0301. doi: 10.1145/3355089.3356510. URL <https://doi.org/10.1145/3355089.3356510>.
- [41] Sai Praveen Bangaru, Tzu-Mao Li, and Frédo Durand. Unbiased warped-area sampling for differentiable rendering. *ACM Trans. Graph.*, 39(6), November 2020. ISSN 0730-0301. doi: 10.1145/3414685.3417833. URL <https://doi.org/10.1145/3414685.3417833>.
- [42] Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. Soft rasterizer: A differentiable renderer for image-based 3d reasoning. *The IEEE International Conference on Computer Vision (ICCV)*, October 2019.

-
- [43] Tzu-Mao Li, Michal Lukáč, Michaël Gharbi, and Jonathan Ragan-Kelley. Differentiable vector graphics rasterization for editing and learning. *ACM Trans. Graph.*, 39(6), nov 2020.
- [44] Delio Vicini, Sébastien Speierer, and Wenzel Jakob. Differentiable signed distance function rendering. In *ACM Trans. Graph.* Vicini et al. [63]. ISSN 0730-0301. doi: 10.1145/3528223.3530139. URL <https://doi.org/10.1145/3528223.3530139>. Alias of vicini2022differentiable.
- [45] Sai Praveen Bangaru, Michael Gharbi, Fujun Luan, Tzu-Mao Li, Kalyan Sunkavalli, Milos Hasan, Sai Bi, Zexiang Xu, Gilbert Bernstein, and Fredo Durand. Differentiable rendering of neural sdfs through reparameterization. In *SIGGRAPH Asia 2022 Conference Papers*, SA '22, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450394703. doi: 10.1145/3550469.3555397. URL <https://doi.org/10.1145/3550469.3555397>. Alias of Bangaru2022NeuralSDFReparam.
- [46] Sai Praveen Bangaru, Jesse Michel, Kevin Mu, Gilbert Bernstein, Tzu-Mao Li, and Jonathan Ragan-Kelley. Systematically differentiating parametric discontinuities. *ACM Trans. Graph.*, 40(4):1–18, 2021.
- [47] Yuting Yang, Connelly Barnes, Andrew Adams, and Adam Finkelstein. A δ : autodiff for discontinuous programs-applied to shaders. *ACM Trans. Graph.*, 41(4):1–24, 2022.
- [48] Yang Zhou, Lifan Wu, Ravi Ramamoorthi, and Ling-Qi Yan. Vectorization for fast, analytic, and differentiable visibility. *ACM Transactions on Graphics*, 40(3), July 2021.
- [49] Eric Veach. *Robust Monte Carlo Methods for Light Transport Simulation*. PhD thesis, Stanford University, 1997.
- [50] Herman Hansson Söderlund, Alex Evans, and Tomas Akenine-Möller. Ray tracing of signed distance function grids. *Journal of Computer Graphics Techniques Vol*, 11(3), 2022.
- [51] David Bremer and John F. Hughes. Rapid approximate silhouette rendering of implicit surfaces. In *Proceesings of Implicit Surfaces 98*, 1998.
- [52] Wenzel Jakob, Sébastien Speierer, Nicolas Roussel, and Delio Vicini. Dr.jit: a just-in-time compiler for differentiable rendering. *ACM Trans. Graph.*, 41(4), July 2022. ISSN 0730-0301. doi: 10.1145/3528223.3530099. URL <https://doi.org/10.1145/3528223.3530099>.
- [53] T. S. Trowbridge and K. P. Reitz. Average irregularity representation of a rough surface for ray reflection. *J. Opt. Soc. Am.*, 65(5):531–536, May 1975.
- [54] Baptiste Nicolet, Alec Jacobson, and Wenzel Jakob. Large steps in inverse rendering of geometry. *ACM Trans. Graph.*, 40(6), December 2021. ISSN 0730-0301. doi: 10.1145/3478513.3480501. URL <https://doi.org/10.1145/3478513.3480501>.

Bibliography

- [55] Marios Papas, Thomas Houit, Derek Nowrouzezahrai, Markus Gross, and Wojciech Jarosz. The magic lens: refractive steganography. *ACM Trans. Graph.*, 31(6), November 2012. ISSN 0730-0301. doi: 10.1145/2366145.2366205. URL <https://doi.org/10.1145/2366145.2366205>.
- [56] Mariia Soroka, Christoph Peters, and Steve Marschner. Quadric-based silhouette sampling for differentiable rendering. *ACM Trans. Graph.*, 44(4), July 2025. ISSN 0730-0301. doi: 10.1145/3731146. URL <https://doi.org/10.1145/3731146>.
- [57] Lifan Wu, Nathan Morrical, Sai Praveen Bangaru, Rohan Sawhney, Shuang Zhao, Chris Wyman, Ravi Ramamoorthi, and Aaron Lefohn. Unbiased differential visibility using fixed-step walk-on-spherical-caps and closest silhouettes. *ACM Trans. Graph.*, 44(4), July 2025. ISSN 0730-0301. doi: 10.1145/3731174. URL <https://doi.org/10.1145/3731174>.
- [58] Delio Vicini, Wenzel Jakob, and Anton Kaplanyan. A non-exponential transmittance model for volumetric scene representations. *ACM Transactions on Graphics (TOG)*, 40(4):1–16, 2021.
- [59] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4460–4470, 2019.
- [60] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3504–3515, 2020.
- [61] Merlin Nimier-David, Thomas Müller, Alexander Keller, and Wenzel Jakob. Unbiased inverse volume rendering with differential trackers. *ACM Trans. Graph.*, 41(4), July 2022. ISSN 0730-0301. doi: 10.1145/3528223.3530073. URL <https://doi.org/10.1145/3528223.3530073>.
- [62] Benedikt Bitterli, Srinath Ravichandran, Thomas Müller, Magnus Wrenninge, Jan Novák, Steve Marschner, and Wojciech Jarosz. A radiative transfer framework for non-exponential media. *ACM Trans. Graph.*, 37(6), December 2018. ISSN 0730-0301. doi: 10.1145/3272127.3275103. URL <https://doi.org/10.1145/3272127.3275103>.
- [63] Delio Vicini, Sébastien Speierer, and Wenzel Jakob. Differentiable signed distance function rendering. *ACM Trans. Graph.*, 41(4), July 2022. ISSN 0730-0301. doi: 10.1145/3528223.3530139. URL <https://doi.org/10.1145/3528223.3530139>.
- [64] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL <https://api.semanticscholar.org/CorpusID:6628106>.

-
- [65] Ishit Mehta, Manmohan Chandraker, and Ravi Ramamoorthi. A theory of topological derivatives for inverse rendering of geometry. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 419–429, 2023.
- [66] 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. *NeurIPS*, 2021.
- [67] Antoine Guédon and Vincent Lepetit. Sugar: Surface-aligned gaussian splatting for efficient 3d mesh reconstruction and high-quality mesh rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5354–5363, 2024.
- [68] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4), July 2022. ISSN 0730-0301. doi: 10.1145/3528223.3530127. URL <https://doi.org/10.1145/3528223.3530127>.
- [69] Zhiqin Chen, Thomas Funkhouser, Peter Hedman, and Andrea Tagliasacchi. Mobilenerf: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16569–16578, 2023.
- [70] Christian Reiser, Stephan Garbin, Pratul Srinivasan, Dor Verbin, Richard Szeliski, Ben Mildenhall, Jonathan Barron, Peter Hedman, and Andreas Geiger. Binary opacity grids: Capturing fine geometric detail for mesh-based view synthesis. *ACM Transactions on Graphics (TOG)*, 43(4):1–14, 2024.
- [71] Yiming Wang, Qin Han, Marc Habermann, Kostas Daniilidis, Christian Theobalt, and Lingjie Liu. Neus2: Fast learning of neural implicit surfaces for multi-view reconstruction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3295–3306, 2023.
- [72] Rasmus Jensen, Anders Dahl, George Vogiatzis, Engin Tola, and Henrik Aanæs. Large scale multi-view stereopsis evaluation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 406–413, 2014.
- [73] Yao Yao, Zixin Luo, Shiwei Li, Jingyang Zhang, Yufan Ren, Lei Zhou, Tian Fang, and Long Quan. Blendedmvs: A large-scale dataset for generalized multi-view stereo networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1790–1799, 2020.
- [74] Qiancheng Fu, Qingshan Xu, Yew Soon Ong, and Wenbing Tao. Geo-neus: Geometry-consistent neural implicit surfaces learning for multi-view reconstruction. *Advances in Neural Information Processing Systems*, 35:3403–3416, 2022.

Bibliography

- [75] Danpeng Chen, Hai Li, Weicai Ye, Yifan Wang, Weijian Xie, Shangjin Zhai, Nan Wang, Haomin Liu, Hujun Bao, and Guofeng Zhang. Pgsr: Planar-based gaussian splatting for efficient and high-fidelity surface reconstruction. *arXiv preprint arXiv:2406.06521*, 2024.
- [76] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, et al. Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 559–568, 2011.
- [77] Dor Verbin, Peter Hedman, Ben Mildenhall, Todd Zickler, Jonathan T. Barron, and Pratul P. Srinivasan. Ref-NeRF: Structured view-dependent appearance for neural radiance fields. *CVPR*, 2022.
- [78] Thomas Müller, Fabrice Rousselle, Jan Novák, and Alexander Keller. Real-time neural radiance caching for path tracing. *ACM Trans. Graph.*, 40(4), July 2021. ISSN 0730-0301. doi: 10.1145/3450626.3459812. URL <https://doi.org/10.1145/3450626.3459812>.
- [79] Gregory J. Ward, Francis M. Rubinstein, and Robert D. Clear. A ray tracing solution for diffuse interreflection. *SIGGRAPH Comput. Graph.*, 22(4):85–92, June 1988. ISSN 0097-8930. doi: 10.1145/378456.378490. URL <https://doi.org/10.1145/378456.378490>.
- [80] G. Greger, P. Shirley, P.M. Hubbard, and D.P. Greenberg. The irradiance volume. *IEEE Computer Graphics and Applications*, 18(2):32–43, 1998. doi: 10.1109/38.656788.
- [81] J. Krivanek, P. Gautron, S. Pattanaik, and K. Bouatouch. Radiance caching for efficient global illumination computation. *IEEE Transactions on Visualization and Computer Graphics*, 11(5):550–561, 2005. doi: 10.1109/TVCG.2005.83.
- [82] Jaroslav Křivánek, Pascal Gautron, Greg Ward, Okan Arikan, and Henrik Wann Jensen. Practical global illumination with irradiance caching. In *ACM SIGGRAPH 2007 Courses*, SIGGRAPH '07, page 1–es, New York, NY, USA, 2007. Association for Computing Machinery. ISBN 9781450318235. doi: 10.1145/1281500.1281617. URL <https://doi.org/10.1145/1281500.1281617>.
- [83] Wojciech Jarosz, Craig Donner, Matthias Zwicker, and Henrik Wann Jensen. Radiance caching for participating media. *ACM Trans. Graph.*, 27(1), March 2008. ISSN 0730-0301. doi: 10.1145/1330511.1330518. URL <https://doi.org/10.1145/1330511.1330518>.
- [84] Julio Marco, Adrian Jarabo, Wojciech Jarosz, and Diego Gutierrez. Second-order occlusion-aware volumetric radiance caching. *ACM Trans. Graph.*, 37(2), July 2018. ISSN 0730-0301. doi: 10.1145/3185225. URL <https://doi.org/10.1145/3185225>.
- [85] Nikolaus Binder, Sascha Fricke, and Alexander Keller. Fast path space filtering by jittered spatial hashing. In *ACM SIGGRAPH 2018 Talks*, SIGGRAPH '18, New York,

- NY, USA, 2018. Association for Computing Machinery. ISBN 9781450358200. doi: 10.1145/3214745.3214806. URL <https://doi.org/10.1145/3214745.3214806>.
- [86] Ari Silvennoinen and Jaakko Lehtinen. Real-time global illumination by precomputed local reconstruction from sparse radiance probes. *ACM Trans. Graph.*, 36(6), November 2017. ISSN 0730-0301. doi: 10.1145/3130800.3130852. URL <https://doi.org/10.1145/3130800.3130852>.
- [87] Mikhail Dereviannykh, Dmitrii Klepikov, Johannes Hanika, and Carsten Dachsbacher. Neural two-level monte carlo real-time rendering. In *Computer Graphics Forum*, page e70050. Wiley Online Library, 2025.
- [88] Advanced Micro Devices AMD. Amd fsr radiance caching. <https://gpuopen.com/amd-fsr-radiancecaching/>, 2025. [Online; accessed 14-January-2026].
- [89] NVIDIA. NVIDIA RTXGI, 2024. URL <https://github.com/NVIDIAGameWorks/RTXGI>. [Online; accessed 14-January-2026].
- [90] 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.
- [91] Shuang Zhao, Lifan Wu, Frédo Durand, and Ravi Ramamoorthi. Downsampling scattering parameters for rendering anisotropic media. *ACM Trans. Graph.*, 35(6), December 2016. ISSN 0730-0301. doi: 10.1145/2980179.2980228. URL <https://doi.org/10.1145/2980179.2980228>.
- [92] Pramook Khungurn, Daniel Schroeder, Shuang Zhao, Kavita Bala, and Steve Marschner. Matching real fabrics with micro-appearance models. *ACM Trans. Graph.*, 35(1), December 2015. doi: 10.1145/2818648.
- [93] Zdravko Velinov, Marios Papas, Derek Bradley, Paulo Gotardo, Parsa Mirdehghan, Steve Marschner, Jan Novák, and Thabo Beeler. Appearance capture and modeling of human teeth. *ACM Trans. Graph.*, 37(6), December 2018. ISSN 0730-0301. doi: 10.1145/3272127.3275098. URL <https://doi.org/10.1145/3272127.3275098>.
- [94] Baptiste Nicolet, Fabrice Rousselle, Jan Novak, Alexander Keller, Wenzel Jakob, and Thomas Müller. Recursive control variates for inverse rendering. *ACM Trans. Graph.*, 42(4), July 2023. ISSN 0730-0301. doi: 10.1145/3592139. URL <https://doi.org/10.1145/3592139>.
- [95] Yash Belhe, Bing Xu, Sai Praveen Bangaru, Ravi Ramamoorthi, and Tzu-Mao Li. Importance sampling brdf derivatives. *ACM Trans. Graph.*, 43(3), April 2024. ISSN 0730-0301. doi: 10.1145/3648611. URL <https://doi.org/10.1145/3648611>.
- [96] Zhimin Fan, Pengcheng Shi, Mufan Guo, Ruoyu Fu, Yanwen Guo, and Jie Guo. Conditional mixture path guiding for differentiable rendering. *ACM Trans. Graph.*, 43(4), July 2024. ISSN 0730-0301. doi: 10.1145/3658133. URL <https://doi.org/10.1145/3658133>.

Bibliography

- [97] Wesley Chang, Venkataram Sivaram, Derek Nowrouzezahrai, Toshiya Hachisuka, Ravi Ramamoorthi, and Tzu-Mao Li. Parameter-space restir for differentiable and inverse rendering. In *ACM SIGGRAPH 2023 Conference Proceedings*, SIGGRAPH '23, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400701597. doi: 10.1145/3588432.3591512. URL <https://doi.org/10.1145/3588432.3591512>.
- [98] Yu-Chen Wang, Chris Wyman, Lifan Wu, and Shuang Zhao. Amortizing samples in physics-based inverse rendering using restir. *ACM Trans. Graph.*, 42(6), December 2023. ISSN 0730-0301. doi: 10.1145/3618331. URL <https://doi.org/10.1145/3618331>.
- [99] Haolin Lu, Delio Vicini, Wesley Chang, and Tzu-Mao Li. Vector-valued monte carlo integration using ratio control variates. *ACM Trans. Graph.*, 44(4), July 2025. ISSN 0730-0301. doi: 10.1145/3731175. URL <https://doi.org/10.1145/3731175>.
- [100] Philippe Weier, J  r  my Riviere, Ruslan Guseinov, Stephan Garbin, Philipp Slusallek, Bernd Bickel, Thabo Beeler, and Delio Vicini. Practical inverse rendering of textured and translucent appearance. *ACM Trans. Graph.*, 44(4), July 2025. ISSN 0730-0301. doi: 10.1145/3730855. URL <https://doi.org/10.1145/3730855>.
- [101] Wesley Chang, Xuanda Yang, Yash Belhe, Ravi Ramamoorthi, and Tzu-Mao Li. Spatiotemporal bilateral gradient filtering for inverse rendering. In *SIGGRAPH Asia 2024 Conference Papers*, SA '24, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400711312. doi: 10.1145/3680528.3687606. URL <https://doi.org/10.1145/3680528.3687606>.
- [102] Philippe Weier, Marc Droske, Johannes Hanika, Andrea Weidlich, and Ji  r   Vorba. Optimised path space regularisation. *Computer Graphics Forum (Proc. EGSR)*, 40(4), 2021. doi: 10.1111/cgf.14347.
- [103] Kai Yan, Vincent Pegoraro, Marc Droske, Ji  r   Vorba, and Shuang Zhao. Differentiating variance for variance-aware inverse rendering. In *SIGGRAPH Asia 2024 Conference Papers*, SA '24, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400711312. doi: 10.1145/3680528.3687603. URL <https://doi.org/10.1145/3680528.3687603>.
- [104] Xiuming Zhang, Pratul P. Srinivasan, Boyang Deng, Paul Debevec, William T. Freeman, and Jonathan T. Barron. Nerfactor: neural factorization of shape and reflectance under an unknown illumination. *ACM Trans. Graph.*, 40(6), December 2021. ISSN 0730-0301. doi: 10.1145/3478513.3480496. URL <https://doi.org/10.1145/3478513.3480496>.
- [105] Yuanqing Zhang, Jiaming Sun, Xingyi He, Huan Fu, Rongfei Jia, and Xiaowei Zhou. Modeling indirect illumination for inverse rendering. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. doi: 10.1109/CVPR52688.2022.01809.
- [106] Haian Jin, Isabella Liu, Peijia Xu, Xiaoshuai Zhang, Songfang Han, Sai Bi, Xiaowei Zhou, Zexiang Xu, and Hao Su. Tensor: Tensorial inverse rendering. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

-
- [107] Jiakai Sun, Weijing Zhang, Zhanjie Zhang, Tianyi Chu, Guangyuan Li, Lei Zhao, and Wei Xing. Joint optimization of triangle mesh, material, and light from neural fields with neural radiance cache, 2025.
- [108] Chun Gu, Xiaofei Wei, Zixuan Zeng, Yuxuan Yao, and Li Zhang. Irgs: Inter-reflective gaussian splatting with 2d gaussian ray tracing. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.
- [109] Binbin Huang, Zehao Yu, Anpei Chen, Andreas Geiger, and Shenghua Gao. 2d gaussian splatting for geometrically accurate radiance fields. In *ACM SIGGRAPH 2024 Conference Papers*, SIGGRAPH '24, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400705250. doi: 10.1145/3641519.3657428. URL <https://doi.org/10.1145/3641519.3657428>.
- [110] Yao Yao, Jingyang Zhang, Jingbo Liu, Yihang Qu, Tian Fang, David McKinnon, Yanghai Tsin, and Long Quan. Neilf: Neural incident light field for physically-based material estimation. In *European Conference on Computer Vision (ECCV)*, 2022.
- [111] Haoqian Wu, Zhipeng Hu, Lincheng Li, Yongqiang Zhang, Changjie Fan, and Xin Yu. Nefii: Inverse rendering for reflectance decomposition with near-field indirect illumination. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023. doi: 10.1109/CVPR52729.2023.00418.
- [112] Jingyang Zhang, Yao Yao, Shiwei Li, Jingbo Liu, Tian Fang, David McKinnon, Yanghai Tsin, and Long Quan. Neilf++: Inter-reflectable light fields for geometry and material estimation. In *IEEE International Conference on Computer Vision (ICCV)*, 2023.
- [113] Sean Wu, Shamik Basu, Tim Broedermann, Luc Van Gool, and Christos Sakaridis. PBR-NeRF: Inverse rendering with physics-based neural fields. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.
- [114] Jingwang Ling, Ruihan Yu, Feng Xu, Chun Du, and Shuang Zhao. Nerf as a non-distant environment emitter in physics-based inverse rendering. In *ACM SIGGRAPH 2024 Conference Papers*, SIGGRAPH '24, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400705250. doi: 10.1145/3641519.3657404. URL <https://doi.org/10.1145/3641519.3657404>.
- [115] Benjamin Attal, Dor Verbin, Ben Mildenhall, Peter Hedman, Jonathan T Barron, Matthew O'Toole, and Pratul P Srinivasan. Flash cache: Reducing bias in radiance cache based inverse rendering. In *European Conference on Computer Vision (ECCV)*, pages 20–36. Springer, 2024.
- [116] Kaiwen Jiang, Jia-Mu Sun, Zilu Li, Dan Wang, Tzu-Mao Li, and Ravi Ramamoorthi. Differentiable light transport with gaussian surfels via adapted radiosity for efficient relighting and geometry reconstruction. *ACM Trans. Graph.*, 44(6), December 2025. ISSN 0730-0301. doi: 10.1145/3763305. URL <https://doi.org/10.1145/3763305>.

Bibliography

- [117] Yohan Poirier-Ginter, Jeffrey Hu, Jean-Francois Lalonde, and George Drettakis. Editable physically-based reflections in raytraced gaussian radiance fields. In *Proceedings of the SIGGRAPH Asia 2025 Conference Papers*, SA Conference Papers '25, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400721373. doi: 10.1145/3757377.3763971. URL <https://doi.org/10.1145/3757377.3763971>.
- [118] Saeed Hadadan and Matthias Zwicker. Neural differential radiance field: Learning the differential space using a neural network. In Bin Sheng, Lei Bi, Jinman Kim, Nadia Magnenat-Thalmann, and Daniel Thalmann, editors, *Advances in Computer Graphics*, pages 93–104, Cham, 2024. Springer Nature Switzerland.
- [119] Jaakko Lehtinen, Jacob Munkberg, Jon Hasselgren, Samuli Laine, Tero Karras, Miika Aittala, and Timo Aila. Noise2Noise: Learning image restoration without clean data. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2965–2974. PMLR, 10–15 Jul 2018. URL <https://proceedings.mlr.press/v80/lehtinen18a.html>.
- [120] Merlin Nimier-David. *Differentiable Physically Based Rendering: Algorithms, Systems and Applications*. PhD dissertation, École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland, 2022. URL <https://merl.in/phd-thesis.pdf>. Thesis No. 9123.