

Efficient Differentiable Rendering: Approximate and Physics-based

vorgelegt von

Markus Andreas Worchel, M. Sc.

an der Fakultät IV – Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
- Dr.-Ing. -

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender:	Prof. Dr. Guillermo Gallego (TU Berlin)
Gutachter:	Prof. Dr. Marc Alexa (TU Berlin)
Gutachter:	Prof. Wenzel Jakob, PhD (EPFL)
Gutachter:	Univ.Prof. Dr.techn. Michael Wimmer (TU Wien)

Tag der wissenschaftlichen Aussprache: 28. Mai 2026

Berlin 2026

Abstract

Rendering is traditionally seen as a forward process that generates an image from a description of a virtual scene. In recent years, this scope was significantly broadened and rendering algorithms have been used to solve several challenging problems across computer vision, medicine, and other domains. This development is based on the ability to differentiate rendering algorithms, which means being able to quantify how variations of the virtual scene affect a generated image. Among others, this allows solving inverse problems, i.e., finding a representation of an environment that explains given (image) observations, automatically using gradient-based methods.

In rendering, one typically distinguishes between physically-based algorithms that are costly but accurately simulate light transport, and rendering algorithms that are focused on interactive performance but use various approximations for light transport. The reality of differentiable rendering has shown that both classes have their merit, and even further that inverse applications have brought up rendering algorithms that are specifically tailored to certain optimization tasks. Regardless of the application, efficiency is a universal goal in differentiable rendering, among others, because it enables faster exploration of larger solution spaces and even broader applicability. In this thesis, we tackle efficiency in three ways unique to differentiable rendering.

First, we make building blocks from real-time rendering available to differentiable rendering by ensuring that derivatives are well-defined while maintaining efficiency. This enables using cheap approximations for certain light transport effects in differentiable renderers. For one, we show that one of the oldest approximations for shadows, shadow maps, can be adapted to work in differentiable rendering frameworks. And further, we introduce a more general family of moment-based approximations to differentiable rendering. We provide a robust and fast implementation that is based on a proof of differentiability, and show that the resulting techniques yield significantly higher accuracy than existing approaches while being equally efficient.

Second, we reduce the computational cost and memory requirements of differentiating physically-based rendering algorithms. This not only improves efficiency but also broadens their applicability, making them practical in new application domains. More specifically, we revisit an algorithm for efficient reverse-mode differentiation, and introduce a variant that correctly accounts for derivatives with respect to scene geometry. We then turn our focus towards time-resolved rendering and show how the same algorithm can be extended beyond the steady-state setting, introducing *time-resolved path replay backpropagation*. We demonstrate that this algorithm has broad applications beyond graphics, enabling efficient and fully-differentiable path tracing for geometric acoustics.

Lastly, we develop a rendering algorithm that exploits the structure of inverse applications, using efficient approximations of light transport tailored to the task. We demonstrated this with the example of 3D surface reconstruction from images, by combining efficient triangle rasterization with the expressiveness of neural rendering. This design enables efficient 3D reconstruction in environments with arbitrary materials and illumination, without having to resort to costly light transport simulations.

Zusammenfassung

Rendering wird traditionell als ein Vorwärtsprozess betrachtet, der ein Bild aus der Beschreibung einer virtuellen Szene generiert. In den vergangenen Jahren hat sich der Anwendungsbereich deutlich ausgeweitet und Rendering-Algorithmen wurden eingesetzt, um verschiedenste Probleme im maschinellen Sehen, in der Medizin, oder in anderen Feldern zu lösen. Diese Entwicklung basiert hauptsächlich darauf, Rendering-Algorithmen differenzieren zu können, also quantifizieren zu können, welche Auswirkungen Veränderungen der virtuellen Szene auf das generierte Bild haben. Dies ermöglicht unter anderem, inverse Problemstellungen – also das Finden einer Szene, die eine vorgegebene Beobachtung (z. B. ein Bild) besonders plausibel wiedergibt – automatisiert mit Hilfe von Gradientenabstiegsverfahren zu lösen.

Typischerweise wird beim Rendering zwischen zwei Klassen von Algorithmen unterschieden: physikalisch-basierte, die kostspielig sind, aber dafür genaue Ergebnisse liefern, und solche mit Fokus auf interaktive Anwendungen, die zwar sehr effizient sind, dafür aber nur approximative Ergebnisse erzeugen. Die Vergangenheit hat gezeigt, dass für differenzierbares Rendering beide Klassen eine Daseinsberechtigung haben, und darüber hinaus sogar gänzlich neue Rendering-Algorithmen entwickelt wurden, die an gewisse Problemstellungen gekoppelt sind. Unabhängig von der Anwendung ist Effizienz ein allgemeines Ziel, denn dadurch wird das noch schnellere Ergründen noch größerer Lösungsräume und ein weitreichenderer Einsatz ermöglicht. In dieser Arbeit beschäftigen wir uns auf drei Arten mit der Effizienz von differenzierbarem Rendering.

Zuerst ermöglichen wir den Einsatz diverser Komponenten aus der Echtzeitgrafik in diesem Kontext, indem wir deren Differenzierbarkeit und Effizienz sicherstellen. Das erlaubt es, günstige Approximationen gewisser Lichttransporteffekte in differenzierbaren Renderern einzusetzen. Genauer zeigen wir zunächst, dass einer der ältesten Algorithmen zur Erzeugung von Schatten, „Shadow Mapping“, für das differenzierbare Rendering angepasst werden kann. Des Weiteren führen wir eine allgemeinere Familie von Approximationen, basierend auf Momenten, ein. Wir stellen eine robuste und performante Implementierung bereit, welche auf einem theoretischen Beweis der Differenzierbarkeit basiert. Wir zeigen, dass diese Techniken im Vergleich zu existierenden Ansätzen eine deutlich höhere Genauigkeit liefern bei vergleichbaren Kosten.

Zweitens reduzieren wir den Rechenaufwand und den Speicherverbrauch, die zum Differenzieren physikalisch-basierter Algorithmen benötigt werden. Das erhöht nicht nur die Effizienz dieser Algorithmen, sondern erweitert auch deren Anwendbarkeit und Praktikabilität in neuen Feldern. Wir schauen uns zunächst einen existierenden Algorithmus für effizientes Differenzieren im Rückwärtsmodus an und führen eine neue Variante ein, die korrekte Ableitungen bezüglich der Szenengeometrie erzeugt. Wir wenden uns dann dem zeitaufgelösten Rendering zu und zeigen, dass derselbe Algorithmus dahingehend erweitert werden kann. Wir zeigen, dass unser Algorithmus, Time-Resolved Path Replay Backpropagation, ein breites Anwendungsspektrum außerhalb der Computergrafik hat und für effizientes, differenzierbares Path Tracing im Kontext von geometrischer Akustik verwendet werden kann.

Schließlich entwickeln wir einen Rendering-Algorithmus, der die Struktur eines in-

versen Problems ausnutzt und effiziente, anwendungsspezifische Approximationen für Lichttransport verwendet. Wir zeigen das am Beispiel der 3D-Oberflächenrekonstruktion aus Bildern und kombinieren effiziente Dreiecksrasterung mit der Ausdruckskraft von neuronalem Rendering. Dieses Verfahren ermöglicht eine effiziente 3D-Rekonstruktion in Umgebungen mit beliebigen Materialien und Beleuchtungsbedingungen, ohne dass kostspielige Lichttransportsimulationen erforderlich sind.

Acknowledgements

First and foremost, I am deeply grateful to my advisor Marc Alexa. For the trust he put in me, the constant support, and the guidance he provided over the past few years. I do not take it for granted that he gave me the freedom to pursue the research questions I was most excited about, despite them *slightly* deviating from typical questions in geometry. His knowledge and the many insightful perspectives contributed substantially to the success of my research projects.

I would like to thank Wenzel Jakob, Michael Wimmer, and Guillermo Gallego for serving on the thesis committee and dedicating their time, not only by reviewing my thesis but also by physically attending the defense.

I was fortunate to spend my journey almost in equal parts between two labs. In the IMC group at Fraunhofer HHI, I took my first steps in research. I am grateful to have been given the opportunity to contribute to various research projects with immediate real-world impact, especially those related to cultural heritage. I would like to thank Rodrigo Diaz, Marcus Zepp, Mykyta Kovalenko, Weiwen Hu, and Decai Chen for the many fun moments and insightful discussions, as well as Oliver Schreer, Ingo Feldmann, and Peter Eisert for their guidance. In the Computer Graphics group at TU Berlin, I matured as a researcher. Being surrounded by so many talented colleagues and students has been an incredible learning experience. Thanks to Anna Koch for involving me in my first research project at TU Berlin – witnessing the collaboration and submission cycles from the inside was an invaluable lesson. Thanks to Ugo Finnendahl, Maximilian Kohlbrenner, Tobias Djuren, Hendrik Meyer, Dimitris Bogiokas, and Gabriel Dogadov for making this an enjoyable time, the fun moments we had on our journeys around the world, and, of course, the many deep discussions about research and life. Thanks to Gaëlle Fer-Arslan for always having our back on the administrative side of things.

Thanks to the collaborators I have been fortunate to work with over the years, in particular Ugo Finnendahl, Tobias Jüterbock, Rodrigo Diaz, Weiwen Hu, and Daniel Wujecki, without whom many of the projects in this thesis would not have been possible. Apart from my colleagues and advisors, various researchers from the graphics community have influenced my approach to research, visualization, and writing, including but not limited to Alec Jacobson, Baptiste Nicolet, Daniele Panozzo, Delio Vicini, Hsueh-Ti Derek Liu, Keenan Crane, Merlin Nimier-David, Silvia Sellán, Tzu-Mao Li, and Wenzel Jakob. I am also grateful to the European Union for funding part of the research in this thesis.

Ich danke meinen guten Freunden Robert, Tim und Julian für ihren stetigen Rückhalt.

Tiefsten Dank an meine Eltern Petra und Frank für ihre Unterstützung in all den Jahren und die Freiheit, jederzeit meinen eigenen Weg gehen zu können, und natürlich an meinen Bruder Thomas, von dem ich schon früh viel über Computer lernen konnte.

Ich bin unendlich glücklich, dass ich diesen Weg zusammen mit Isabelle gehen durfte. Ohne ihre Liebe, ihre unermüdliche Unterstützung und ihr Verständnis in stressigen Phasen wäre diese Reise nur schwer zu bewältigen gewesen.

Contents

Abstract	iii
Zusammenfassung	v
1 Introduction	1
1.1 Overview and Contributions	3
1.2 Publications	3
1.2.1 Other Contributions and Publications	4
2 Background	7
2.1 Physically-based Rendering	7
2.1.1 Representation of a Virtual Scene	7
2.1.2 The Rendering Equation	10
2.1.3 Monte Carlo Methods	11
2.2 Real-Time Rendering Techniques	13
2.2.1 Point Light Sources	13
2.2.2 Shadow Mapping	15
2.3 Differentiation Techniques	16
2.4 Differentiable Rendering	21
2.4.1 Applications	22
2.4.2 Discontinuities and Visibility	23
2.4.3 Path Replay Backpropagation	27
3 Differentiable Shadow Mapping	31
3.1 Method	32
3.1.1 Discontinuities in Shadow Mapping	33
3.1.2 An Inspiration from Real-Time Rendering	33
3.1.3 Differentiable Pre-Filtered Shadow Mapping	34
3.2 Implementation and Parameter Exploration	37
3.3 Applications	42
3.3.1 Monocular Pose Estimation	42
3.3.2 Face Reconstruction from Profile Shadows	44
3.3.3 Shadow Art	45
3.3.4 Interactive Modeling from Shadows	47
3.4 Conclusion	47
4 Differentiable Moment Bounds	51
4.1 Background: Bounding Measures with Moments	52
4.1.1 A Geometric Perspective and General Observations	53
4.1.2 Computing the Unique Canonical Representation	56
4.2 Differentiating Moment-based Bounds	57
4.2.1 Partitioning the Domain	58

4.2.2	Approaching Singularities	59
4.2.3	Weight (Derivative) at Singularities	61
4.3	Implementation	61
4.3.1	Explicit Algorithm and Reverse-Mode Differentiation	62
4.3.2	Fused Implementation	64
4.4	Applications	69
4.4.1	Differentiable Shadow Mapping	69
4.4.2	Differentiable Visibility for Volume Rendering	73
4.5	Conclusion	81
5	Radiative Backpropagation with Non-Static Geometry	83
5.1	Rederiving the Differential Radiance Equations	84
5.1.1	Differential Scattering	84
5.1.2	Differential Transport	85
5.1.3	Case Study	86
5.2	Non-Static Geometry and Detached Sampling	88
5.2.1	Detached Sampling of Directions	89
5.2.2	Detached Sampling of Positions	90
5.3	Implementation and Experiments	93
5.3.1	Case Study – Revisited	93
5.3.2	Unique Discontinuities	93
5.3.3	Radiative Backpropagation for Interior Path Derivatives	97
5.4	Conclusion	100
6	Time-Resolved Path Replay Backpropagation	103
6.1	Background: Acoustic Path Tracing	105
6.2	Method	106
6.2.1	A Functional Perspective of Rendering Integrals	107
6.2.2	Differentiable Rendering in Acoustics	108
6.2.3	Path Replay for Temporal Derivatives	110
6.2.4	Backwarding with Resolution-Independent Storage	112
6.3	Implementation and Validation	115
6.3.1	Verification of Primal Simulation	116
6.3.2	Validation of Derivatives	117
6.4	Applications	121
6.4.1	Optimization of Material Parameters	121
6.4.2	Optimization of Scene Geometry	122
6.5	Conclusion	125
7	Neural Deferred Shading	127
7.1	Method	129
7.1.1	Neural Deferred Shading	129
7.1.2	Objective Function	130
7.1.3	Optimization Schedule	132
7.2	Experiments	132
7.2.1	3D Reconstruction	132

7.2.2	Analysis of the Neural Shader	134
7.2.3	Ablation Studies	136
7.2.4	Failure Cases	138
7.3	Conclusion	139
8	Conclusion	141
8.1	Outlook	142
A	Differentiable Moment Bounds	145
A.1	Transmittance as Opacity	145
A.2	Accuracy Verification Results	145
A.3	Proofs	146
A.3.1	Orthogonal Polynomials	146
A.3.2	Intermediate Statements	148
A.3.3	Proof of Theorem 5	149
A.3.4	Proof of Theorem 6	149
A.3.5	Proof of Theorem 7	150
A.3.6	Proof of Lemma 1	150
A.3.7	Proof of Lemma 2	150
B	Radiative Backpropagation with Non-Static Geometry	153
B.1	An Attached Approach for Detached Sampling	153
B.2	Derivation of the Three-Point Form Pseudocode	154
C	Time-Resolved Path Replay Backpropagation	157
C.1	Time-Resolved PRB in the Three-Point Form	157
C.2	Acoustic BSDF	158
C.3	Room Acoustics Metrics	161
	Bibliography	163

List of Figures

1.1	Derivatives of rendered images	2
2.1	An image generated with physically-based rendering (geometric optics).	8
2.2	Monte Carlo integration in 2D.	11
2.3	An object rendered using different types of point light sources	14
2.4	Shadow mapping and bias	15
2.5	Rasterization and discontinuous visibility	25
3.1	Differentiable shadow mapping in a complex scene	31
3.2	Supported types of point light sources	37
3.3	Effect of anti-aliasing in differentiable shadow mapping	39
3.4	Effect of shadow map resolution and filter size	40
3.5	Derivatives of shadows cast by a highly-tessellated mesh	40
3.6	Effect of filter size on convergence	41
3.7	Setting of the light estimation experiments	41
3.8	Quantitative results of a light estimation experiments	42
3.9	Pose estimation results	43
3.10	Face reconstruction from synthetic shadows.	45
3.11	Face reconstruction from real silhouettes of public figures	46
3.12	Shadow art with one view	46
3.13	Shadow art with two views	47
3.14	Differentiable shadows as a modeling tool	48
4.1	Overview of differentiable moment-based techniques	51
4.2	Derivatives of moment bounds for a discrete measure	57
4.3	Geometric relationship of canonical representation and moment vector .	60
4.4	Runtime and memory requirements of implementations	65
4.5	Runtime and memory requirements with increasing number of bounds .	67
4.6	Numerical stability near singularities	68
4.7	Optimizing the light position and albedo of an object	70
4.8	Runtime of VSM and MSM for appearance decomposition	71
4.9	3D mesh reconstruction from a single image	72
4.10	Singularities in moment shadow mapping	73
4.11	Effect of the differentiable rasterizer on an optimization	74
4.12	Effect of the differentiable rasterizer on the shadow map derivatives . . .	74
4.13	Varying quality of transmittance approximation with number of moments	77
4.14	Optimizing the pose of simple analytic shapes	78
4.15	Performing shadow art by using differentiable volume rendering	79
4.16	Multi-view 3D reconstruction with approximate visibility	80
5.1	Bias of existing PRB implementations in a simple case study	87
5.2	Three-point PRB derivatives in a simple case study	95
5.3	Three-point PRB derivatives in a simple, discontinuity-free case study . .	96

5.4	Biased combination of PRB with projective sampling	98
5.5	A simple inverse rendering task that fails due to bias	99
5.6	Combination of PRB interior and projective sampling boundary derivatives	101
6.1	Differentiable geometric acoustic path tracing	103
6.2	Functional perspective of Monte Carlo sample	108
6.3	Reparameterization for discontinuity handling	109
6.4	Validation of the primal simulation	116
6.5	Handling of time derivatives	117
6.6	Variance of the energy histogram derivatives	118
6.7	Derivative comparison of AD and TPRB	119
6.8	Forward simulation of a hallway scene with different path lengths	120
6.9	Optimizing absorption parameters of a simple room	121
6.10	Optimization towards a target spectrogram	122
6.11	Reconstructing an emitter position in the shoebox	123
6.12	Optimization of the ceiling reflector angles in an auditorium	124
7.1	Teaser of neural deferred shading	127
7.2	Overview of the 3D reconstruction pipeline	129
7.3	Architecture of the neural shader	130
7.4	Qualitative comparison on the DTU dataset	132
7.5	Equal time reconstruction	134
7.6	View synthesis for an input view and a new view	135
7.7	Positional feature analysis and simple material editing	135
7.8	Ablation study of the objective function.	136
7.9	Ablation study of different initial meshes	137
7.10	Ablation study of the positional encoding	137
7.11	Ablation study of the positional part network size	138
7.12	Failure cases of our reconstruction approach	139

List of Tables

- 3.1 Quantitative results for the pose estimation experiments 44
- 3.2 Quantitative results for face reconstruction from shadows 45
- 6.1 Comparison of PRB and AD 120
- 7.1 Quantitative results for reconstruction of objects from the DTU dataset . 133
- 7.2 Average runtime of one gradient descent iteration of IDR and our method 134
- A.1 Accuracy of the implementations 146

CHAPTER 1

Introduction

Computer graphics has traditionally seen rendering as a *forward* process: generate an image from a description of a virtual scene, which includes the geometry of objects, their materials, and camera parameters. The main artifact of interest has been the *image*, which serves as the final output in typical applications for rendering systems, like movie production, visualization, or video games.

In recent years, it has become clear that rendering algorithms have a much broader scope. Instead of merely serving as image factories, they have been used to solve challenging problems across computer vision, medicine, robotics, manufacturing, and computational design more generally. This particular development is based on the ability to relate the outputs back to the inputs. For rendering, this means being able to quantify how variations of the virtual scene affect the generated image (see Fig. 1.1). The reason this is useful is that many tasks in the above domains can be formulated as *inverse* problems: given an observation, for example, a photograph, what (real-world) environment has likely led to it? Fundamentally, the idea is to take a virtual scene and adjust its parameters until rendered images match the observations, in which case the virtual scene provides a possible explanation. A key realization is that when the effect of scene parameters on the generated image can be quantified, this process can be done in a structured and automated manner.

If seen abstractly, rendering algorithms are simply mathematical functions that map some inputs (the parameters of the scene) to some outputs (the image). The relation between inputs and outputs of a function is described by the derivatives. Differentiable rendering seeks to apply this to rendering algorithms, so that the primary artifact of interest is not only the image but also its *derivatives* with respect to scene parameters. Given the complexity of modern renderers, it might appear infeasible to differentiate these systems. However, on the lowest level, all of them simply perform a sequence of numerical computations according to mathematical models and equations, most of which can be viewed and differentiated in isolation.

The various building blocks of rendering algorithms have traditionally differed depending on the application, although they have been converging in recent years. On the one hand, physically-based rendering aims at generating images that faithfully reproduce the appearance of the real world. It simulates the propagation of light through a scene and its interaction with objects by solving the physical equations of light transport. This enables accurate reproduction of various phenomena, including soft shadows, interreflection, and caustics. The accuracy comes at the price of efficiency, such that these systems are typically used in offline scenarios, for example, movie production. On the other hand, various rendering algorithms focus on efficiency and interactive scenarios,

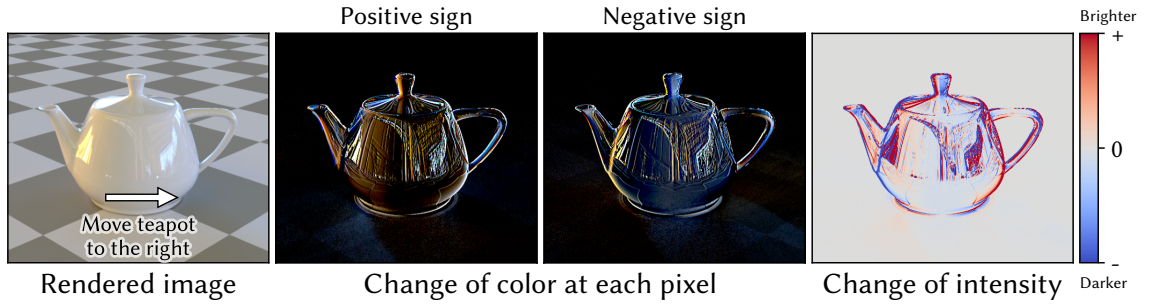


Figure 1.1: Differentiable rendering is concerned with computing derivatives of rendered images with respect to parameters of a virtual scene. The derivatives capture the effect of infinitesimal perturbations of the scene on each pixel, here shown for a rightward movement of the teapot (middle). The signed quantity is usually averaged over channels and displayed using color mapping to visualize the change of intensity (right).

which typically requires several approximations of light transport. The aim is to generate images that appear plausible in real-time applications like video games or interactive visualization.

The reality of differentiable rendering has shown that both classes have their merit, and the appropriate balance between physical accuracy and efficiency strongly depends on the specific inverse application. Even further, many solutions rely on rendering algorithms that are tailored to the requirements of the task at hand. While these methods often build upon components of traditional rendering systems, they may also introduce new approximations or techniques that are primarily motivated by the optimization objectives rather than forward image synthesis.

Regardless of the application, improving the efficiency of differentiable rendering algorithms in terms of runtime and memory requirements is a universal goal. First, reducing computational cost lowers energy consumption, which is increasingly important in the context of climate change and power-efficient computing. Second, as improvements in hardware continue to slow down, progress increasingly depends on developing more efficient algorithms and software. Third, even with current hardware capabilities, efficient methods enable more rapid exploration of a larger space of possible solutions. Lastly, efficiency is a deciding factor in the practicability of solutions, so increasing efficiency also broadens the range of potential applications.

While efficiency in differentiable rendering shares the challenges of traditional forward rendering, as it also involves image synthesis, it comes with its own set of challenges and opportunities. In this thesis, we focus on the latter and tackle efficiency in three ways specific to differentiable rendering:

1. We make building blocks from real-time rendering available by ensuring that derivatives are well-defined while maintaining efficiency. This enables using cheap approximations for certain light transport effects in a differentiable renderer.
2. We reduce the computational cost and memory requirements of differentiating physically-based rendering algorithms. This not only improves efficiency but also broadens their applicability, making them practical in new application domains.

3. We develop rendering algorithms that exploit the structure of inverse applications, using efficient approximations of light transport tailored to the task. This solves inverse problems in complex environments without explicitly simulating light.

1.1 Overview and Contributions

This thesis contributes to efficient differentiable rendering in various ways. We will contextualize the different contributions in the separate chapters and only provide a brief overview here:

Chapter 3: An approach for differentiating approximations of cast shadows from point light sources.

Chapter 4: An approach for differentiating approximations of cast shadows and volumetric transmittance from point light sources.

Chapter 5: An algorithm for efficient reverse-mode differentiation of light transport simulations with parameters that potentially affect the scene geometry.

Chapter 6: An algorithm for efficient reverse-mode differentiation of energy transport simulations that are resolved in time, and an approach for fully-differentiable geometric acoustic simulations.

Chapter 7: A rendering algorithm tailored to 3D multi-view reconstruction that efficiently approximates light transport effects using machine learning techniques.

1.2 Publications

All contributions in this thesis have been published in peer-reviewed journals and conferences. The content is freely adapted from these publications:

Chapter 3 ([Worchel and Alexa, 2023b]):

Differentiable Shadow Mapping for Efficient Inverse Graphics

Markus Worchel and Marc Alexa

Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR 2023)

Chapter 4 ([Worchel and Alexa, 2025]):

Moment Bounds are Differentiable:

Efficiently Approximating Measures in Inverse Rendering

Markus Worchel and Marc Alexa

ACM Transactions on Graphics (Proceedings of SIGGRAPH 2025)

Chapter 5 ([Worchel et al., 2025]):

Radiative Backpropagation with Non-Static Geometry

Markus Worchel[†], Ugo Finnendahl[†], Marc Alexa

Proceedings of the Eurographics Symposium on Rendering (EGSR 2025)

Thanks for substantial contributions: Ugo Finnendahl (mathematical derivation for pseudocode)

Chapter 6 ([Finnendahl et al., 2025]):

Differentiable Geometric Acoustic Path Tracing using Time-Resolved Path Replay Backpropagation

Ugo Finnendahl[†], Markus Worchel[†], Tobias Jüterbock, Daniel Wujecki, Fabian Brinkmann, Stefan Weinzierl, Marc Alexa

ACM Transactions on Graphics (Proceedings of SIGGRAPH 2025)

Thanks for substantial contributions: Ugo Finnendahl ((T)PRB iterative formulas, TPRB in three-point form, validation/applications) and Tobias Jüterbock (applications)

Chapter 7 ([Worchel et al., 2022]):

Multi-View Mesh Reconstruction with Neural Deferred Shading

Markus Worchel[†], Rodrigo Diaz[†], Weiwen Hu, Oliver Schreer, Ingo Feldmann, Peter Eisert

Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR 2022)

Thanks for substantial contributions: Rodrigo Diaz (network ablation studies) and Weiwen Hu (comparison to COLMAP)

[†] Equal contribution (joint first authors)

1.2.1 Other Contributions and Publications

The following peer-reviewed articles were published during the course of the doctoral program, but their content is not covered by this thesis:

Differentiable Rendering of Parametric Geometry

Markus Worchel and Marc Alexa

ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia 2023)

Ernst Grube:

A Contemporary Witness and His Memories Preserved with Volumetric Video.

Markus Worchel, Marcus Zepp, Weiwen Hu, Oliver Schreer, Ingo Feldmann, and Peter Eisert

Eurographics Workshop on Graphics and Cultural Heritage (EG GCH 2020)

Interpolating Splines over Triangulated Surfaces by Blending Vertex-Centric Local Geometries

Tobias Djuren, Ugo Finnendahl, Maximilian Kohlbrenner, Markus Worchel, Marc Alexa
Computers & Graphics (2025)

Hardware Design and Accurate Simulation of Structured-Light Scanning for Benchmarking of 3D Reconstruction Algorithms

Anna Koch[†], Yurii Piadyk[†], Markus Worchel, Marc Alexa, Claudio Silva, Denis Zorin, Daniele Panozzo.

Conference on Neural Information Processing Systems (NeurIPS 2021), Datasets and Benchmarks Track

Capture and 3D Video Processing of Volumetric Video

Oliver Schreer, Ingo Feldmann, Sylvain Renault, Marcus Zepp, Markus Worchel, Peter Eisert, and Peter Kauff

IEEE International Conference on Image Processing (ICIP 2019)

[†] Equal contribution (joint first authors)

CHAPTER 2

Background

Images of virtual scenes can be generated in different ways: physically-based rendering (Section 2.1) simulates light according to physical principles, whereas efficient real-time systems commonly make simplifying assumptions to achieve interactive performance (Section 2.2). While these approaches differ in accuracy and computational cost, both ultimately produce images as the result of a numerical computation. Differentiable rendering (Section 2.4) uses this perspective and applies techniques for differentiating programs (Section 2.3) to traditional rendering algorithms to be able to compute derivatives of rendered images with respect to arbitrary scene parameters.

2.1 Physically-based Rendering

Physically-based rendering (PBR) is the process of simulating light transport in virtual scenes to synthesize images that reproduce the appearance of the real world. While light has particle and wave characteristics, simulations often ignore wave effects, following the principles of *geometric optics*: light travels along straight lines in homogeneous media, it can be absorbed or reflected, and it refracts when experiencing a change in velocity. Many effects of light transport can be convincingly reproduced using this model (Fig. 2.1), and it also serves as the theoretical framework for this thesis. In this section, we will provide a brief overview of relevant concepts and refer readers to Pharr et al. [2016] for a more comprehensive treatment.

To simulate light transport, the virtual environment, or *scene*, first requires a mathematical and digital representation (Section 2.1.1). Only then can the physical equations of light transport (Section 2.1.2) be solved computationally (Section 2.1.3).

2.1.1 Representation of a Virtual Scene

In a common abstraction of the real world, a virtual environment, or *scene*, is assumed to consist of *surfaces*, i.e., infinitely thin shells that represent the boundaries of objects, which are made of *materials* that determine how a surface interacts with incident light. *Emitters* radiate light while *sensors* register it to produce an image or any other form of measurement. This thesis is most concerned with surface light transport, so we do not discuss *participating media* (e.g., fog or fire). The aforementioned concepts are briefly introduced in the following.



Figure 2.1: An image generated with physically-based rendering (geometric optics).

Surface Geometry

Objects in a scene that interact with or emit light are commonly represented by surfaces, i.e., a representation of their boundary. There are many different ways to describe surfaces mathematically and represent them in a computer. We will only introduce those that are most relevant for this thesis.

Triangle Meshes Triangle meshes are piecewise linear representations where the surface is explicitly described by an assembly of flat, triangular facets. They are most ubiquitous in computer graphics and rendering, among others, because graphics processing units (GPUs) possess specialized rasterization hardware to efficiently display them to a screen. Mathematically, they can be described by vertex positions $\mathbf{V} = (\mathbf{v}_i \in \mathbb{R}^3)_{i=1}^n$ and a set of index tuples $\mathcal{T} = \{(i, j, k) \in \mathbb{N}^3\}$, where each represents a triangle by the vertex indices of its vertices. The surface normal is constant over a triangle (i, j, k) , and it is given by $\mathbf{n} = (\mathbf{v}_j - \mathbf{v}_i) \times (\mathbf{v}_k - \mathbf{v}_i)$ (the index order determines the orientation).

Implicit Surfaces Surfaces can also be represented implicitly as the zero-level set of some function $\phi : \mathbb{R}^3 \rightarrow \mathbb{R}$, such that the set of all surface points is $\{\mathbf{x} \in \mathbb{R}^3 \mid \phi(\mathbf{x}) = 0\}$. In contrast to triangle meshes, this representation is more challenging to render because displaying it effectively amounts to solving a root-finding problem. This generally has to be done numerically (e.g., using variants of ray marching [Tuy and Tuy, 1984]). Signed distance functions (SDFs) are a special class of implicit surfaces where the value $\phi(\mathbf{x})$ is the signed distance from $\mathbf{x}$ to the closest point on the surface. The surface normal is given by the spatial gradient $\mathbf{n} = \frac{\nabla_{\mathbf{x}}\phi}{\|\nabla_{\mathbf{x}}\phi\|_2}$. For signed distance functions, this is simply $\mathbf{n} = \nabla_{\mathbf{x}}\phi$ as they fulfill the Eikonal equation $\|\nabla_{\mathbf{x}}\phi\|_2 = 1$ where ϕ is differentiable.

Emitters

Emitters generate the light that is simulated. Generally, objects in the scene or parts of them can be characterized as emitters, in which case they are referred to as *area* emitters.

Interactive rendering often uses emitters that are not tied to specific objects in the scene (Section 2.2.1), e.g., directional emitters as an approximation for the sun.

Light itself is electromagnetic radiation that can be perceived by the human visual system, where different wavelengths are generally seen as different colors. While precisely handling the spectral characteristics of light is crucial for physical accuracy, we restrict our computations to a tristimulus representation, with red, green, and blue components, as is common in many rendering systems.

The emitted light is described in terms of radiance $L_e(\mathbf{p}, \boldsymbol{\omega})$, which is the radiant flux (or radiant power) per unit area and solid angle. Notice that the emission is spatio-directional, so it can vary for each point $\mathbf{p}$ on the emitter surface and direction $\boldsymbol{\omega}$.

Materials

Materials determine how objects, or more precisely, their surfaces, interact with incident light, including how it is reflected in different directions or transmitted through the surface. The choice of material influences an object's appearance, for example, whether it appears to be made of brushed metal or wood. Generally, this behavior is described by a bidirectional scattering distribution function (BSDF), and for the case of opaque objects, by a bidirectional reflectance distribution function (BRDF):

$$f(\mathbf{p}, \boldsymbol{\omega}, \boldsymbol{\omega}_i). \quad (2.1)$$

At any point $\mathbf{p}$ on a surface, the BSDF (or BRDF) represents the ratio of infinitesimal outgoing radiance in direction $\boldsymbol{\omega}$ to the infinitesimal incident irradiance (i.e., radiance incident over an infinitesimal projected solid angle) from direction $\boldsymbol{\omega}_i$. Eq. (2.4) shows how this fits into the light transport framework.

Example: Lambertian The Lambertian BRDF is one of the simplest models and describes a perfectly diffuse surface, where incident light from direction $\boldsymbol{\omega}_i$ is reflected uniformly so that the surface appearance is independent of the viewing direction $\boldsymbol{\omega}$:

$$f(\mathbf{p}, \boldsymbol{\omega}, \boldsymbol{\omega}_i) = \frac{\rho}{\pi}. \quad (2.2)$$

The reflectance $\rho \in [0, 1]$ determines the fraction of reflected incident light, and the normalization by π ensures energy conservation (no more light is reflected than what is incident). While this is an idealized model, i.e., no real-world material behaves exactly this way, it is commonly used throughout this thesis in synthetic experiments.

Sensors

Light can be scattered by objects in a scene, and ultimately it is measured by a sensor. While sensors can generate any kind of measurement, the most commonly used in this thesis are cameras that generate an image.

A single measurement can be described by the measurement equation [Veach, 1998]:

$$I = \int_{\mathcal{M} \times \mathcal{S}^2} W_e(\mathbf{p}, \boldsymbol{\omega}) L_i(\mathbf{p}, \boldsymbol{\omega}) dA(\mathbf{p}) d\sigma_{\mathbf{p}}^\perp(\boldsymbol{\omega}), \quad (2.3)$$

where $L_i(\mathbf{p}, \boldsymbol{\omega})$ is the incident radiance and $W_e(\mathbf{p}, \boldsymbol{\omega})$ is the spatio-directional sensitivity of the sensor for this particular measurement. Here, $\mathcal{M}$ denotes the union of all surfaces in the scene, and dA is the area measure. $\mathcal{S}^2$ represents the unit sphere, and $d\sigma_p^\perp$ is the *projected* solid angle measure, incorporating a cosine foreshortening factor (see Eq. (2.5)).

While this form covers arbitrary measurements of radiance in the scene, the sensitivity W_e will typically be restricted to a small (rectangular) image sensor in our settings, which is considered to be part of the scene $\mathcal{M}$. An image would then correspond to multiple measurements $I_1, \dots, I_n$ over the sensor surface, if n is the number of pixels.

2.1.2 The Rendering Equation

In Section 2.1.1, we have covered how light is represented, emitted, and measured. We now consider how light is transported through the scene to reach the sensor, starting from an emitter. As noted before, we assume that the scene does not contain any participating media, so light travels through the scene in a vacuum and interacts only with object surfaces. Under this assumption, radiance remains constant along straight lines through empty space. For the most part of this thesis, we will also assume a steady state, that is, the energy distribution in the scene has reached an equilibrium. In Chapter 6 we will temporarily lift this last assumption, but discuss the implications there.

The relationship between incident radiance and outgoing radiance for any point in the scene is described by the rendering equation [Kajiya, 1986]:

$$L_o(\mathbf{p}, \boldsymbol{\omega}) = L_e(\mathbf{p}, \boldsymbol{\omega}) + \int_{\mathcal{S}^2} L_i(\mathbf{p}, \boldsymbol{\omega}_i) f(\mathbf{p}, \boldsymbol{\omega}, \boldsymbol{\omega}_i) |\boldsymbol{\omega}_i \cdot \mathbf{n}| d\sigma(\boldsymbol{\omega}_i), \quad (2.4)$$

where L_e is the emitted radiance from a point $\mathbf{p}$ into a direction $\boldsymbol{\omega}$, f is the bidirectional scattering distribution function, $\mathbf{n}$ is the surface normal at $\mathbf{p}$, and the integral is over the sphere $\mathcal{S}^2$, with $d\sigma(\boldsymbol{\omega}_i)$ denoting the solid angle measure. Throughout this thesis, we will often use the common shorthand notation

$$d\boldsymbol{\omega}_i := d\sigma(\boldsymbol{\omega}_i) \quad \text{and} \quad d\boldsymbol{\omega}_i^\perp := |\boldsymbol{\omega}_i \cdot \mathbf{n}| d\boldsymbol{\omega}_i \quad (2.5)$$

for the solid angle measure and the projected solid angle measure, respectively. The outgoing radiance L_o and the incoming radiance L_i are related via the transport relation

$$L_o(\mathbf{p}, \boldsymbol{\omega}) = L_o(\mathbf{r}(\mathbf{p}, \boldsymbol{\omega}), -\boldsymbol{\omega}), \quad (2.6)$$

where $\mathbf{r}(\mathbf{p}, \boldsymbol{\omega})$ is the closest surface point from $\mathbf{p}$ in direction $\boldsymbol{\omega}$.

Substituting the transport relation (2.6) into the rendering equation (2.4)

$$L_o(\mathbf{p}, \boldsymbol{\omega}) = L_e(\mathbf{p}, \boldsymbol{\omega}) + \int_{\mathcal{S}^2} L_o(\mathbf{r}(\mathbf{p}, \boldsymbol{\omega}_i), -\boldsymbol{\omega}_i) f(\mathbf{p}, \boldsymbol{\omega}, \boldsymbol{\omega}_i) |\boldsymbol{\omega}_i \cdot \mathbf{n}| d\sigma(\boldsymbol{\omega}_i), \quad (2.7)$$

shows that L_o appears on both sides of the equation: simulating light transport through a scene requires solving a recursive integral equation.

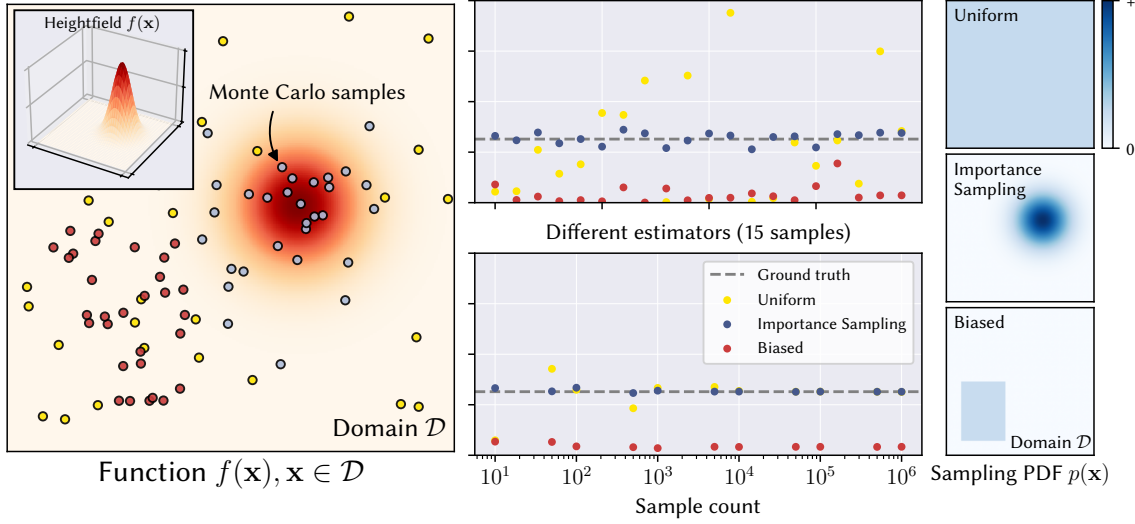


Figure 2.2: Monte Carlo integration in 2D. A function $f(\mathbf{x})$ is defined over the domain $\mathcal{D} := [0, 1]^2$ and the integral $F = \int_{\mathcal{D}} f(\mathbf{x}) \, d\mathbf{x}$ is estimated using a Monte Carlo estimator $F \approx F_N = \frac{1}{N} \sum_{i=1}^N \frac{f(\mathbf{x}_i)}{p(\mathbf{x}_i)}$ by drawing N random samples according to a probability density function (PDF) $p(\mathbf{x})$. An estimator based on uniform sampling is unbiased but shows high variance. Importance sampling targets the shape of the integrand when drawing samples to reduce variance. If the PDF does not cover the support of f , i.e., $p(\mathbf{x}) = 0$ for $|f(\mathbf{x})| > 0$, the estimator is biased.

2.1.3 Monte Carlo Methods

Simulating light transport amounts to solving a recursive integral equation (Section 2.1.2), which effectively means that radiance is carried from the emitters to the sensor along potentially infinitely many light paths: recursively substituting L_o using Eq. (2.7) expands it into a sum of integrals over light paths of increasing length, where each additional surface interaction introduces another integral. Except for the most simplistic scenes, one cannot hope to find closed-form solutions for these high-dimensional integrals, which motivates numerical approaches.

Since the computational complexity of deterministic quadrature rules grows exponentially with the dimensionality of the integration domain, rendering typically relies on Monte Carlo integration (Fig. 2.2): the integral is approximated by randomly sampling points in the high-dimensional integration domain, the *samples*, whose number is independent of the problem dimensionality.

Monte Carlo Estimator To illustrate the approach consider the following integral over a bounded domain $\mathcal{D} \in \mathbb{R}^n$:

$$F = \int_{\mathcal{D}} f(\mathbf{x}) \, d\mathbf{x}. \quad (2.8)$$

A Monte Carlo estimator of this integral is

$$F_N = \frac{1}{N} \sum_{i=1}^N \frac{f(\mathbf{x}_i)}{p(\mathbf{x}_i)}, \quad (2.9)$$

where N is the number of samples, and each sample $\mathbf{x}_i$ is drawn independently from the probability density function $p(\mathbf{x})$, which is nonzero whenever $|f(\mathbf{x})| > 0$.

It is easy to verify that the above estimator is *unbiased*, which means that it returns the true value of the integral on average. Unbiasedness is expressed using the expected value:

$$\mathbb{E}[F_N] = F. \quad (2.10)$$

and bias can be quantified by the difference

$$\beta = \mathbb{E}[F_N] - F. \quad (2.11)$$

Unbiasedness will play a central role in the second part of this thesis (Chapter 5, Chapter 6), not primarily of light transport simulations, but of their derivatives (Section 2.4). In forward rendering, biased estimators will result in visible, yet sometimes tolerable, artifacts. But, as we will see, bias in the derivatives can cause an optimization to fail, even when the forward process is unbiased.

Importance Sampling Apart from bias, *variance* is another way to quantify the quality of a Monte Carlo estimator:

$$\mathbb{V}[F_N] = \mathbb{E} \left[(\mathbb{E}[F_N] - F_N)^2 \right]. \quad (2.12)$$

Intuitively, it measures how much the estimator varies around the expected value, on average. Estimators with high variance occasionally return poor estimates of the true integral, even though they might be unbiased. Using the properties of variance, assuming the $\mathbf{x}_i$ are independent and identically distributed, we get

$$\mathbb{V}[F_N] = \frac{1}{N} \mathbb{V} \left[\frac{f(\mathbf{x}_1)}{p(\mathbf{x}_1)} \right], \quad (2.13)$$

which shows that there are two ways to decrease variance: (1) increasing the sample count, or (2) using a probability density that aligns well with the shape of f : if it were possible to sample exactly proportional to f , then the ratio would be a constant and the variance would be zero.

In the context of rendering, sample generation (and evaluation) is a costly process because each sample effectively corresponds to a light path through the scene. A significant part of rendering research is concerned with designing importance sampling strategies to reduce variance at low sample counts; yet, exactly sampling proportional to f is typically infeasible. However, the integrand in rendering is a product of several factors (e.g., incident radiance and BSDF), where individual components *can* often be importance-sampled. A combination of different sampling strategies leads to *multiple importance sampling*, for which we refer to [Pharr et al., 2016, Section 13.4].

Path Tracing Monte Carlo integration can be directly used to solve the rendering equation. A simple and common algorithm that connects the pieces is path tracing (see [Pharr et al., 2016, Section 13.2] for details). The high-level idea is to compute the integrals in Eq. (2.3) and Eq. (2.4) by gradually constructing light paths through the scene, starting from the sensor: (1) sample a direction, often guided by importance sampling (e.g., the BSDF), (2) trace a ray to determine the next intersection point, (3) evaluate and add the path contribution, and (4) continue with (1). Each light path corresponds to a single sample in a high-dimensional integration domain and contributes to the final estimate. To keep the computation finite while preserving unbiasedness, paths are terminated probabilistically using Russian roulette [Pharr et al., 2016, Section 2.2.4]

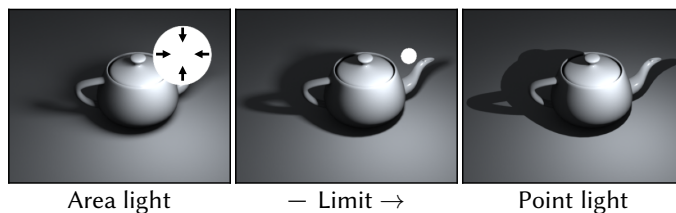
2.2 Real-Time Rendering Techniques

Accurate simulation of light transport (Section 2.1) has traditionally been used in “offline” settings, such as movie production, due to the high computational cost. Applications targeting interactive or real-time performance, for example, visualization or video games, instead rely on simplifying assumptions and approximations of light transport. In these contexts, it is often sufficient to use techniques that produce images that appear convincing to human observers, even if they are not physically accurate, yet are orders of magnitude more efficient. Many of these approaches are shaped by the historical development of real-time rendering, which has traditionally relied on rasterization of triangle meshes. Instead of tracing rays for individual pixels of the camera, rasterization takes a primitive-centric perspective and efficiently determines pixel coverage by projecting triangles to the image plane of the camera, a process that is highly optimized on modern GPUs. As a result, numerous lighting and visibility approximations have been designed specifically to integrate well with rasterization-based pipelines.

2.2.1 Point Light Sources

Physically accurate emitters are typically modeled as area light sources that emit radiance from a finite area in the scene, which belongs to the surface of some object. While this representation closely follows the nature of real light sources, evaluating their contribution generally requires integrating incident radiance over the emitter area. Many real-time systems approximate illumination using simplified models that assume light is emitted from a single *point* in space¹. This greatly simplifies computations and is easy to integrate into rasterization-based rendering.

A possible way of interpreting these non-physical light sources in the context of light transport (Section 2.1) is in terms of Dirac distributions that collapse the integral over solid an-



¹What we call a “point light” in this thesis is also commonly referred to as a “punctual light” [Akenine-Möller et al., 2018], in which case “point light” would imply omnidirectionality.



Figure 2.3: An object rendered using different types of point light sources. Directional lights approximate infinitely distant light sources (e.g., the sun). Omnidirectional point lights emit light in all directions and are commonly used to model small light sources such as light bulbs. Spot lights emit light within a cone-shaped region and are often used to model flashlights or stage lighting.

gles. Depending on how the emission is distributed over directions, this leads to different types of light sources (see Fig. 2.3). A discussion of visibility is deferred to Section 2.2.2.

Directional Lights Directional lights are emitters located at a single point at *infinity*. Hence, their light rays are parallel and illuminate the scene from a single direction ω_l . They are typically used to approximate the sun or other distant light sources whose distance is large relative to the scale of the scene. The contribution of a directional light with radiance E to the outgoing radiance at a point $\mathbf{p}$ into direction ω is

$$E f(\mathbf{p}, \omega, \omega_l) |\omega_l \cdot \mathbf{n}|. \quad (2.14)$$

Omnidirectional Lights Omnidirectional point lights are located at a single, finite point $\mathbf{x}_l$ in the scene. They emit light equally in all directions, but each point $\mathbf{p}$ in the scene is only illuminated from a single direction $\omega_l = \frac{\mathbf{x}_l - \mathbf{x}_p}{\|\mathbf{x}_l - \mathbf{x}_p\|}$:

$$\frac{I}{\|\mathbf{x}_l - \mathbf{x}_p\|^2} f(\mathbf{p}, \omega, \omega_l) |\omega_l \cdot \mathbf{n}|, \quad (2.15)$$

where $\mathbf{x}_p$ is the ambient space position of the surface point $\mathbf{p}$ in $\mathbb{R}^3$. The light contribution decreases according to an inverse-square law, and such lights are commonly used to approximate small light sources such as light bulbs.

Spot Lights Spot lights are, like omnidirectional point lights, located at a single, finite point $\mathbf{x}_l$ and illuminate each point $\mathbf{p}$ in the scene from a direction $\omega_l = \frac{\mathbf{x}_l - \mathbf{x}_p}{\|\mathbf{x}_l - \mathbf{x}_p\|}$. However, the intensity is directionally varying, typically taking the form of a cone:

$$\frac{I(\omega_l)}{\|\mathbf{x}_l - \mathbf{x}_p\|^2} f(\mathbf{p}, \omega, \omega_l) |\omega_l \cdot \mathbf{n}|, \quad (2.16)$$

As with point lights, the contribution exhibits an inverse-square falloff with distance. Spot lights are commonly used to model directed light sources, such as flashlights or stage lighting.

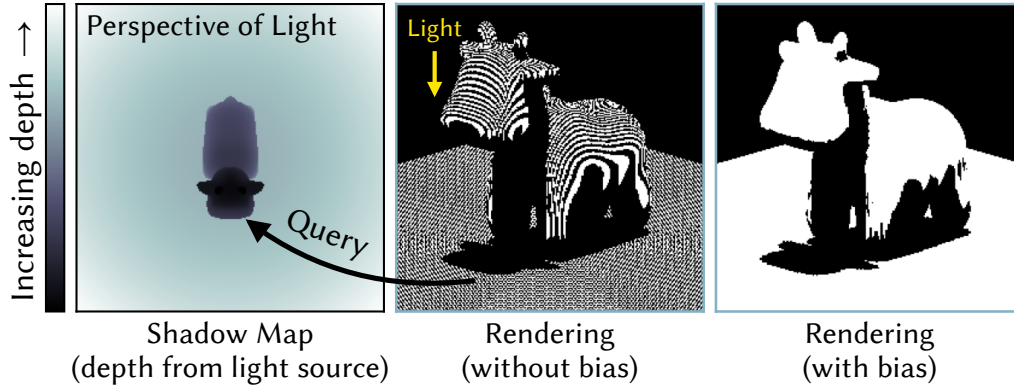


Figure 2.4: Shadow mapping uses a depth map rendered from the perspective of the light to determine if a point in the scene is visible to a point light source. If the depth from a point in the scene to the light source is larger than the queried distance stored in the shadow map, the point is in shadow. Because the shadow map is a raster image, the visibility test suffers from self-shadowing artifacts (“shadow acne”). This can be alleviated by biasing the depths towards the light source for the visibility test.

2.2.2 Shadow Mapping

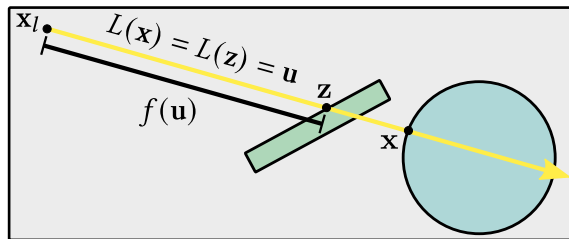
In our discussion of point light sources (Section 2.2.1), we have so far neglected light source visibility: if the path between a light source and a point in the scene is obstructed, the point should be in shadow. In rasterization-based pipelines, light source visibility is typically approximated using some variant of shadow mapping [Williams, 1978].

Shadow mapping is based on the observation that points in a scene are lit by a (point) light source if they are closest to the light source. The *shadow map* is a depth image from the perspective of the light source. If a point in the scene is further from the light than the value stored in the shadow map, it is in the shadow. Otherwise, it has the value stored in the shadow map, is closest to the light source, and is lit by it (Fig. 2.4)

Formally, let the scene be given as the set $\mathcal{X} \subset \mathbb{R}^3$. A point light source is defined by its position $\mathbf{x}_l$. We assume a projective model and represent positions in homogeneous coordinates, so $\mathbf{x}_l$ can be at infinity, creating a directional light source, in which case $\mathbf{x}_l$ is the direction vector ω_l (Section 2.2.1). Denote the distance of $\mathbf{x} \in \mathbb{R}^3$ to the light source as $d : \mathbb{R}^3 \mapsto [0 : d_{\max}]$, where the distance to a directional light source is measured from a plane outside of $\mathcal{X}$ so that distance values are finite and non-negative.

The shadow map is parameterized over the space of rays emanating from $\mathbf{x}_l$. For simplicity we assume that the rays from $\mathbf{x}_l$ to any point in the scene can be parameterized over a single plane and we assume a rectangle $\mathcal{U}$ on this plane to be the parameter space for the rays; if the

light source is placed so that this condition is violated, several planes (i.e., arranged in the form of a box) may be used. The mapping from scene points $\mathbf{x} \in \mathbb{R}^3$ to $\mathcal{U}$ is the linear projective camera transformation $L : \mathbb{R}^3 \mapsto \mathbb{R}^2$. In this setup, the shadow map can



be expressed as

$$f : \mathcal{U} \mapsto \mathbb{R}, \quad f(\mathbf{u}) = \min \{d(\mathbf{x}) : L(\mathbf{x}) = \mathbf{u}, \mathbf{x} \in \mathcal{X}\} \quad (2.17)$$

Given the shadow map, we get the *visibility* of any point in the scene with respect to the light source as

$$V(\mathbf{x}) = \begin{cases} 0 & d(\mathbf{x}) > f(L(\mathbf{x})) \\ 1 & \text{else.} \end{cases} \quad (2.18)$$

It is important to note that the “else” case really reduces to $d(\mathbf{x}) = f(L(\mathbf{x}))$ because f had been constructed to store the smallest possible value of f in direction $L(\mathbf{x})$.

The usefulness of shadow maps arises from the fact that they can be efficiently approximated using graphics hardware: simply render the scene from the perspective of the light source, disabling all lighting computations and storing only the depth values. However, a naive implementation will suffer from severe artifacts. For example, the condition for a point to be lit is $d(\mathbf{x}) \leq f(L(\mathbf{x}))$ in finite precision arithmetic, which may or may not happen for visible points on a smooth surface because of the different discretization of the rendered image and the shadow map. The resulting problem is referred to as “shadow acne” and is commonly tackled by introducing a *bias* β for the depth comparison, considering points with $d(\mathbf{x}) \leq f(L(\mathbf{x})) + \beta$ to be lit (Figure 2.4). However, estimating a suitable bias can be difficult for some scenes [Dou et al., 2014]. While suitably chosen bias attenuates artifacts along smooth surfaces, it cannot remedy aliasing across shadow boundaries. We will defer a discussion of how to mitigate shadow aliasing using “soft” shadows to Chapter 3, because the underlying visibility discontinuities also pose a significant challenge for differentiation.

2.3 Differentiation Techniques

The methods developed throughout this thesis will primarily be used to minimize image-based objectives, which they accomplish by differentiating rendering algorithms. We defer the details of differentiable rendering to Section 2.4 and briefly introduce our notation and relevant techniques for computing derivatives in the following.

Given a multi-variate, vector-valued function

$$\mathbf{f} : \mathbb{R}^n \rightarrow \mathbb{R}^m, \quad \mathbf{f}(\mathbf{x}) = \begin{bmatrix} f_1(\mathbf{x}) \\ \vdots \\ f_m(\mathbf{x}) \end{bmatrix}, \quad (2.19)$$

what we will generally understand as the “derivative of $\mathbf{f}$ ” is its Jacobian

$$\mathbf{J}_{\mathbf{f}}(\mathbf{x}) := \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(\mathbf{x}) & \cdots & \frac{\partial f_1}{\partial x_n}(\mathbf{x}) \\ \vdots & & \vdots \\ \frac{\partial f_m}{\partial x_1}(\mathbf{x}) & \cdots & \frac{\partial f_m}{\partial x_n}(\mathbf{x}) \end{bmatrix}, \quad (2.20)$$

where $\frac{\partial f_i}{\partial x_j}$ denotes the partial derivative of the i -th component with respect to the j -th input. In this thesis, we will often just use the shorthand notation

$$\partial_x f := \frac{\partial f}{\partial x} \quad (2.21)$$

where it makes derivations more readable. In a slight abuse of notation, we will also use the same notation for the Jacobian, so that $\partial_x \mathbf{f}$ really means $\mathbf{J}_f$.

If the components of $\mathbf{f}$ are simple functions based on elementary operations, there is a good chance that we can easily derive an expression for each element of the Jacobian by manually applying differentiation rules. However, the functions encountered throughout this thesis are generally a more complex composition of individual functions:

$$\mathbf{f} = \mathbf{g}_d \circ \dots \circ \mathbf{g}_1. \quad (2.22)$$

While it is often infeasible to derive a single expression for the derivative of the composite $\mathbf{f}$, one can typically derive expressions for the individual functions $\{\mathbf{g}_i\}$, such that the Jacobian is obtained using the chain rule:

$$\mathbf{J}_f = \mathbf{J}_{\mathbf{g}_d} \cdot \dots \cdot \mathbf{J}_{\mathbf{g}_1}. \quad (2.23)$$

While tedious, manually differentiating functions and computing derivatives by explicitly evaluating the resulting expressions *can* have practical benefits in terms of computational efficiency, memory requirements, or other means [Kerbl et al., 2023; Müller et al., 2021] (also see Chapter 4). Although it was shown that this approach to differentiation scales even to entire path tracing simulations [Li et al., 2018a], it is error-prone and inflexible when system requirements change.

Automatic Differentiation

In many cases, one would much rather use *automatic differentiation* (AD). This not only avoids error-prone and tedious derivations but also greatly improved accessibility in recent years: several modern libraries (e.g., PyTorch [Paszke et al., 2019], JAX [Bradbury et al., 2018], or DR.JIT [Jakob et al., 2022a]) expose automatic differentiation tools for machine learning and other optimization tasks. Notice the distinction between automatic and symbolic differentiation: symbolic methods apply differentiation rules to the mathematical expression of a function to obtain an *expression for the derivative*, while automatic differentiation *evaluates derivatives* of a function implemented as a computer program. The details of automatic differentiation have been covered by many textbooks (e.g., by Griewank and Walther [2008]), so we only give a high-level overview here.

First, notice that the decomposition into functions (Eq. (2.22)) is generally not unique. In the case of manual approaches it is often somewhat arbitrarily chosen to coincide with the function definitions in the program code. A key principle of automatic differentiation is decomposing functions at a much finer resolution so that each component of the $\{\mathbf{g}_i\}$ is a small atomic operation whose analytic derivatives are known (e.g., addition or the sine). For example, take the scalar-valued, multi-variate function

$$f(x_1, x_2) = \sin(x_1) + x_1 x_2 - x_2. \quad (2.24)$$

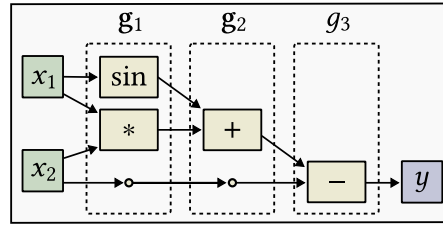
A possible decomposition that follows the above principle would be

$$f(x_1, x_2) = g_3(g_2(g_1(x_1, x_2))), \quad (2.25)$$

with the functions

$$\begin{aligned} g_1(x_1, x_2) &:= (\sin(x_1), x_1 x_2, x_2)^\top \\ g_2(a, b, c) &:= (a + b, c)^\top \\ g_3(d, e) &:= d - e. \end{aligned} \quad (2.26)$$

The relationship between variables is better visualized by the *computational graph*:



It is important that this decomposition into atomic operations does not need to be carried out explicitly, but it naturally follows from the primitives used to write numerical computer programs and the program control flow. The functions g_1 , g_2 , and g_3 therefore merely serve the mental model – knowing the computational graph suffices to calculate the Jacobian according to Eq. (2.23).

However, computing with the full Jacobians is neither required in many applications, nor is it feasible: (intermediate) functions in machine learning and inverse rendering can map millions of parameters to millions of outputs, making this approach prohibitively expensive. Instead, it is sufficient to compute a vector-Jacobian product or a Jacobian-vector product (i.e., a directional derivative). In the following we will introduce specific algorithms for carrying out these operations.

Reverse-mode Reverse-mode differentiation, commonly known as *backpropagation*, computes the vector-Jacobian product $\mathbf{v}^\top \mathbf{J}_f$ with a vector $\mathbf{v} = (v_1, \dots, v_m)^\top$. It is particularly useful for differentiating functions with many input variables and few outputs, for example, for computing the gradient of an objective function (Section 2.4): the partial derivatives of a single output with respect to any number of inputs can be computed simultaneously, with the same time complexity as the function evaluation.

Considering the decomposition of the Jacobian in Eq. (2.23), the idea is to evaluate the product from left to right. The computations performed by this *backward pass* (or alternatively *adjoint pass*) are in reverse order to the *primal pass* (i.e., the function evaluation), effectively traversing the computation graph from the outputs back to the inputs. At a node with output z , the chain rule is applied locally to propagate the sensitivity

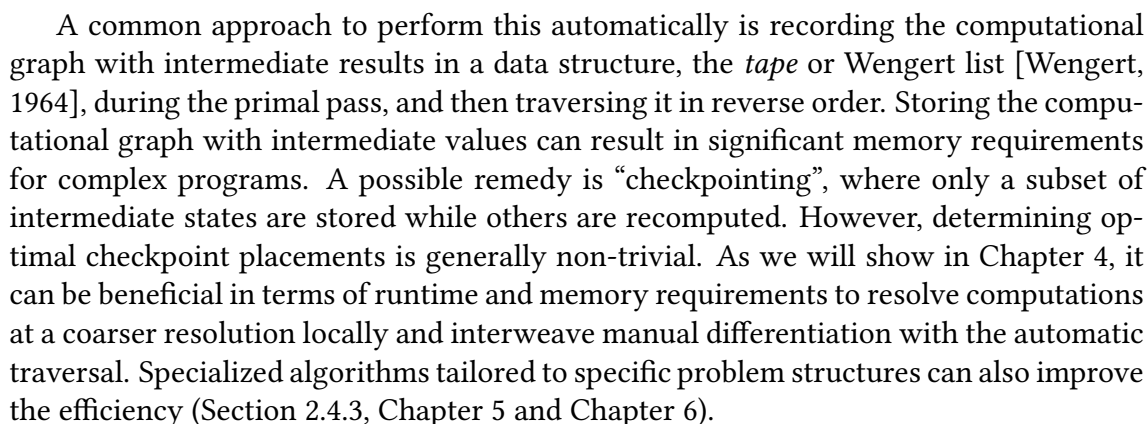
$$\delta z := \sum_{i=1}^m v_i \frac{\partial f_i}{\partial z}, \quad (2.27)$$

or *adjoint*, to its input variables.

Due to the backward traversal, the adjoint δb has already been computed, starting from $\delta y = v_1$. The multiplication contributes the following to the adjoints of x_1 and x_2 :

For both x_1 and x_2 , this is only one of the two contributions in this example:

Generally, the adjoint of a single variable is the sum of the contributions from its dependent variables, all of which can be computed using the principle above:



Forward-mode Forward-mode differentiation computes the Jacobian-vector product $\mathbf{J}_f \mathbf{v}$ – the directional derivative of f in direction $\mathbf{v} = (v_1, \dots, v_n)^\top$. In contrast to reverse-mode, it is useful if a function has few input variables and many outputs: the partial derivatives of any number of outputs with respect to a single input variable can be computed simultaneously, with the same time complexity as the function evaluation. In this thesis, we are mostly interested in gradient-based optimization, and forward-mode differentiation is primarily used for visualization and validation: when differentiating rendering algorithms, it is common to show the effect of a single parameter (e.g., the translation of an object along a given direction) on the output image – this can be efficiently computed using forward-mode.

Considering the decomposition of the Jacobian in Eq. (2.23), the general idea is to evaluate the product from right to left. The computational graph is traversed forward,

from inputs to outputs, in the same order as the primal function evaluation. At a node with output z , the chain rule is used to compute the directional derivative

$$\dot{z} := \sum_{i=1}^n \frac{\partial z}{\partial x_i} v_i \quad (2.31)$$

from the directional derivatives of its inputs.

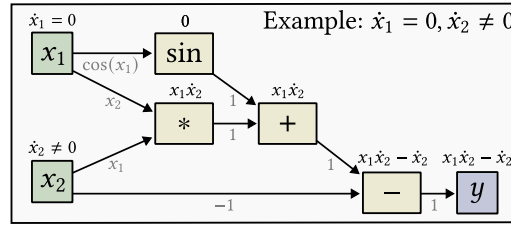
Again, taking the multiplication $b = x_1 x_2$ from the graph above as an example, the directional derivative of the result is

$$\dot{b} = \frac{\partial b}{\partial x_1} \dot{x}_1 + \frac{\partial b}{\partial x_2} \dot{x}_2 = x_2 \dot{x}_1 + x_1 \dot{x}_2, \quad (2.32)$$

which can be computed from the directional derivatives of the inputs:

$$\dot{x}_1 = v_1 \quad \dot{x}_2 = v_2. \quad (2.33)$$

Due to the forward traversal, the directional derivatives of all inputs are given at any node, so this procedure can be continued up to the output:



One way to implement automatic forward-mode differentiation is through dual numbers: each variable z tracks its value and its directional derivative $\dot{z}$. This requires no computational graph and can be carried out alongside the primal computation.

Finite Differences

Finite differences (FD) are a common numerical method for approximating directional derivatives. The general idea is to estimate derivatives by evaluating the function with slightly perturbed inputs. For example, the derivative in direction $\mathbf{v}$ can be approximated using central differences:

$$\mathbf{J}_f(\mathbf{x}) \mathbf{v} \approx \frac{f(\mathbf{x} + h\mathbf{v}) - f(\mathbf{x} - h\mathbf{v})}{2h} \quad (2.34)$$

where h is a small step size that controls the amount of perturbation. While finite differences are easy to implement because they do not require knowledge about the internals of a function, they come with several limitations. For one, the choice of step size h requires an intricate balance: if h is too small, floating point rounding errors dominate the result due to catastrophic cancellation, but if h is too large, the approximation is inaccurate. Moreover, approximating gradients of scalar-valued functions with many inputs is prohibitively expensive because the derivative for each input would require at least one² function evaluation. In this thesis, finite differences will primarily serve as a tool for validating derivatives computed using automatic or manual differentiation.

²Only one additional function evaluation is required per input with forward differences $\frac{f(\mathbf{x}+h\mathbf{v})-f(\mathbf{x})}{h}$.

2.4 Differentiable Rendering

The tools developed in this thesis are primarily aimed at solving optimization problems with an objective, or *loss*, function

$$\mathcal{L}(R(\boldsymbol{\theta})) \in \mathbb{R}, \quad (2.35)$$

where $R(\boldsymbol{\theta})$ generates an image, or another form of measurement, by rendering a virtual scene. Typically, the *scene parameters* $\boldsymbol{\theta} \in \mathbb{R}^n$ characterize the virtual scene by defining cameras, light sources, materials, and geometry (see Section 2.1.1), yet, despite the name, they may include any parameters that affect the rendering output (e.g., parameters of the rendering algorithm). Most often, $\mathcal{L}$ computes the difference between the rendering output and some target – as done for *inverse rendering* problems – but generally it can be any problem-specific, scalar-valued function, which may also take multiple images or measurements generated through rendering (see Section 2.4.1).

We are interested in finding parameters $\boldsymbol{\theta}^*$ that minimize the objective:

$$\boldsymbol{\theta}^* = \arg \min_{\boldsymbol{\theta} \in \mathbb{R}^n} \mathcal{L}(R(\boldsymbol{\theta})). \quad (2.36)$$

Generally, the parameter space is high-dimensional, so an exhaustive search is infeasible. Additionally, one cannot hope to directly solve for $\boldsymbol{\theta}^*$ due to the complexity of $R(\boldsymbol{\theta})$. Instead, these problems are commonly addressed using gradient descent: starting from some initial state $\boldsymbol{\theta}^0$, perform iterative updates of the form

$$\boldsymbol{\theta}^{k+1} \leftarrow \boldsymbol{\theta}^k - \lambda \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}^k)^\top, \quad (2.37)$$

where λ is a sufficiently small step size. Although it is difficult to give exact convergence guarantees for arbitrary functions $\mathcal{L}$, gradient-based methods have been shown to reliably converge to (local) minima in practice.

If the rendering output is an image, i.e., $R(\boldsymbol{\theta}) = \mathbf{I}(\boldsymbol{\theta})$, as will be the case for most parts of this thesis, then from the chain rule, the gradient follows to be

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = \frac{\partial \mathcal{L}}{\partial \mathbf{I}} \frac{\partial \mathbf{I}}{\partial \boldsymbol{\theta}}. \quad (2.38)$$

The particular challenge is differentiating the image $\mathbf{I}$ with respect to scene parameters $\boldsymbol{\theta}$, which means quantifying how the scene parameters affect the image.

*Differentiable rendering*³ is concerned with computing such derivatives, i.e., differentiating the rendering output with respect to the parameters $\boldsymbol{\theta}$. More specifically, since problems usually involve millions of parameters (e.g., textures or vertex positions), the goal is to compute gradients in reverse-mode (see Section 2.3, *Reverse-mode*). Although it is possible to use automatic differentiation (see Section 2.3, *Automatic Differentiation*) to differentiate the implementation of a rendering algorithm, this requires some care: (1) automatic derivatives of numerical computations do not necessarily correspond to the actual derivatives of the underlying mathematical expression (Section 2.4.2); and (2)

³We refer interested readers to the survey by Kato et al. [2020] for a general introduction to the topic.

the more complex $R(\theta)$ becomes, the larger the computational graph stored for (tape-based) automatic differentiation. The potentially prohibitive memory requirements can be avoided using specialized differentiation algorithms (Section 2.4.3). Without loss of generality, we will often just consider a single scene parameter θ throughout the thesis.

2.4.1 Applications

The applications of differentiable rendering are too numerous to cover exhaustively. Instead, each of the main chapters will address the relevant background separately, so we will only briefly discuss the general context here. Broadly speaking, the applications in this thesis fall somewhere between computer graphics and 3D computer vision, many of which have the same general structure: given a reference image I_{ref} (or measurement), find a representation of the environment that gives rise to this observation.

With some rendering algorithm $R(\theta)$ at hand, this task may be viewed as *inverse rendering*, i.e., inverting the rendering process to obtain parameters for a virtual scene as a representation of the environment. However, R is generally not invertible, so instead, this is cast as a minimization problem of the form

$$\theta^* = \arg \min_{\theta \in \mathbb{R}^n} \|R(\theta) - I_{\text{ref}}\|_2^2. \quad (2.39)$$

Notice that this is a special case of Eq. (2.36), and the objective could really be any function that measures the difference between the rendering output and the reference (including neural network-based objectives [Johnson et al., 2016]). This approach to the above task is also known as *analysis-by-synthesis*.

It naturally generalizes to multiple references, and a recurring application in that regard will be multi-view 3D reconstruction: given multiple photographs of a scene (typically with known or estimated camera parameters), recover the scene geometry. With known camera parameters, one can render images from the reference perspectives, compare the rendered images to the reference photographs, and update a representation of the scene geometry (e.g., a triangle mesh) using gradient descent (also see Chapter 7). In general, it is common for some of the scene parameters to be fixed (e.g., known or estimated beforehand), and only a subset is optimized.

Apart from inverse applications, we will occasionally cover others that are not based on matching a given reference. For example, in Chapter 6, scene geometry is optimized to improve certain acoustic characteristics of a room (in that context, $R(\theta)$ generates an acoustic measurement from the scene).

Unfortunately, in most of the applications, we cannot hope to find a global minimum because the objectives are generally non-convex. Even worse, many of the local minima found using differentiable rendering are undesirable: for example, if the task is to reconstruct the surface of an object from real-world images, the renderings might agree with the reference images, but the recovered surface is physically implausible. The application-side drives a significant part of the research on better-suited objectives and robust optimization techniques (e.g., for the evolution of geometry [Luan et al., 2021; Nicolet et al., 2021])

Differentiable rendering comes in many flavors, from general differentiable renderers to application-specific systems. For the most part of the thesis, we will be concerned

with general, application-independent techniques, but we will always relate back to applications by evaluating our methods and algorithms in respective settings.

2.4.2 Discontinuities and Visibility

There is often a distinction made between physically-based differentiable renderers based on ray tracing with Monte Carlo methods (Section 2.1) [Nimier-David et al., 2019; Jakob et al., 2022b; Yan et al., 2020; Li et al., 2018b], and rasterization-based differentiable renderers with approximations (Section 2.2) [Ravi et al., 2020; Fuji Tsang et al., 2022; Laine et al., 2020; de La Gorce et al., 2008]. Regardless of whether one is interested in one or the other, a major challenge shared by both are *parameter-dependent discontinuities*⁴.

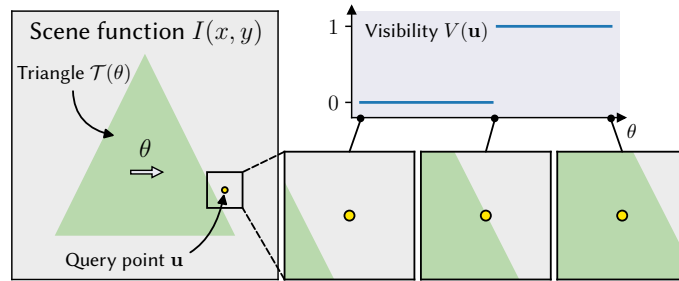
Discontinuities arise because several computations within a renderer involve discrete decisions. For example, a triangle either covers a point in the image or it does not, and a point in the scene is either visible to a point light source, or it is not. Special care is required when differentiating with respect to parameters that affect these decisions. Unfortunately, this case is common: many interesting applications involve optimization of geometry, for example, the vertices of a triangle mesh for 3D reconstruction – and the vertex positions directly affect occlusion relations in the scene.

Example: Primary Visibility To demonstrate why parametric discontinuities are problematic in differentiable rendering, we will use the case of *primary visibility*, in particular, the visibility of a single triangle to a pinhole camera. While in general multiple primitives may occlude each other, visibility in this simple case is entirely determined by coverage: a point in the image plane of the camera is either covered by the triangle, or it is not. Let $\theta \in \mathbb{R}$ be an offset to the triangle vertices leading to a horizontal translation in the image plane. The triangle visibility over the continuous image plane then is:

$$V(x, y, \theta) = \begin{cases} 1 & (x, y) \in \mathcal{T}(\theta) \\ 0 & \text{else,} \end{cases} \quad (2.40)$$

where $\mathcal{T}(\theta)$ denotes the points that lie within the projected triangle.

For the purpose of rendering, each point in the continuous image plane is assumed to have some value (e.g., radiance or a color) based on the visible primitive at the location, giving rise to the so-called *scene function* $I(x, y, \theta)$. Without loss of generality, we assign a constant color to the points covered by the triangle, while we use a background value for the rest. In practice, one may evaluate some shading model to determine the amount of light reflected towards the camera. Since the visibility is piecewise constant with discontinuities, the resulting scene function has the same discontinuities.



⁴Sometimes they are also called *moving* discontinuities. For example, in our primary visibility setting, the points of discontinuity, i.e., the silhouette points of the triangle, “move” over the image plane with θ .

Depending on the rendering approach, rasterization or physically-based rendering, an image is generated from the scene function in different ways.

Rasterization Rasterization effectively point-samples the scene function at discrete grid positions $\mathbf{u}_i = (x_i, y_i)$ in the image plane, each being a pixel in the output image $R(\theta) = (I(\mathbf{u}_i, \theta))_i$. If θ moves the triangle across the image plane, this variation is not reflected in the derivatives: since visibility is a discontinuous, piecewise constant function in θ , its derivative $\partial_\theta V(x, y, \theta)$ is zero almost everywhere, and so is the derivative of $I(x, y, \theta)$ for visibility changes by θ . In the above example with constant colors, this results in pixel derivatives $\partial_\theta I(\mathbf{u}_i, \theta)$ that are either zero or undefined, which makes it impossible to optimize the triangle position.

Various works have tackled this limitation for differentiable rasterization [de La Gorce et al., 2008; Kato et al., 2018; Liu et al., 2019a; Chen et al., 2019; Laine et al., 2020; Petersen et al., 2022; Cole et al., 2021; Pidhorskyi et al., 2025]. Common strategies include blending multiple triangles with soft visibility per pixel [Liu et al., 2019a; Chen et al., 2019; Petersen et al., 2022] or “reattaching” the value of point samples to the underlying geometry after rasterization [Laine et al., 2020] (see Fig. 2.5). While several techniques are only approximations, in practice, this often suffices to solve optimization problems involving visibility.

Physically-based Rendering In physically-based rendering, measurements are performed according to the measurement equation (2.3). So, for the setting above, we can assume that each pixel value effectively is a point-sample of an integrated scene function

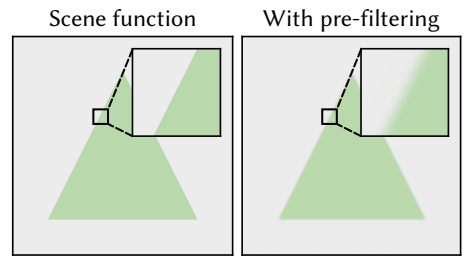
$$R(\theta) = (\tilde{I}(\mathbf{u}_i, \theta))_i \quad \tilde{I}(\mathbf{u}, \theta) = \int_I k_{\mathbf{u}}(\mathbf{p}) I(\mathbf{p}, \theta) dA(\mathbf{p}), \quad (2.41)$$

where $k_{\mathbf{u}}$ is a (smooth) pixel reconstruction filter, and the integral is over the image plane I . Although the scene function itself is discontinuous, this *pre-filtered* scene function is *continuous* over the image plane, and in particular, each pixel value varies continuously in θ . While the effect on a single triangle visually appears similar to the Smooth-then-Rasterize approach shown in Fig. 2.5, they are different: the rasterization approach smooths the *visibility* of triangles, requiring the blending of multiple primitives along a single view ray, whereas this approach smooths the *scene function*. In any case, the derivative $\partial_\theta \tilde{I}(\mathbf{u}_i, \theta)$ is non-zero for points affected by variation of the triangle.

So, what is the difficulty in this setting? Consider the derivative of the pre-filtered scene function with respect to θ :

$$\partial_\theta \tilde{I}(\mathbf{u}, \theta) = \partial_\theta \int_I k_{\mathbf{u}}(\mathbf{p}) I(\mathbf{p}, \theta) dA(\mathbf{p}). \quad (2.42)$$

To make the discussion more readable, we will represent the product of pixel reconstruction



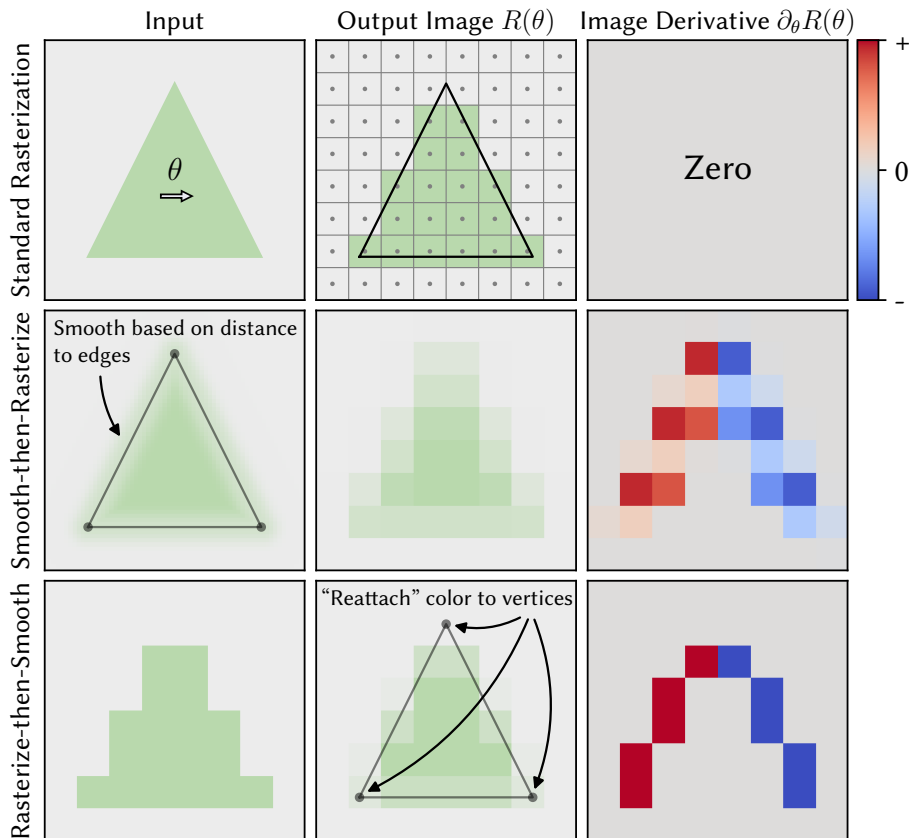


Figure 2.5: Rasterization and discontinuous visibility (see Section 2.4.2). A parameter θ translates a triangle horizontally over the camera image plane (top left). Differentiating the rasterization output with respect to θ results in zero derivatives because it point-samples a scene function that uses discontinuous, piecewise constant visibility (top row). Various approaches have been developed for differentiable rasterization. *Smooth-then-Rasterize* (e.g., [Liu et al., 2019a]) rasterizes *soft* triangles (i.e., with soft visibility), effectively blending multiple values per pixel (middle row). *Rasterize-then-Smooth* (e.g., [Laine et al., 2020]) “reattaches” the rasterization output to the geometry by expressing the values in terms of vertex positions, e.g., using pixel coverage (bottom row).

tion filter and scene function using a single function f , and drop the subscript $\mathbf{u}$:

$$= \partial_\theta \int_I f(\mathbf{p}, \theta) dA(\mathbf{p}). \quad (2.43)$$

Since the integrand f has parameter-dependent discontinuities due to visibility, one cannot simply “differentiate under the integral”, as this would ignore the effect of θ on the discontinuities. Instead, Zhang et al. [2019] have shown that such derivatives factor into two terms according to the Reynolds transport theorem [Reynolds et al., 1903]:

$$= \boxed{\int_{I \setminus \partial I(\theta)} \partial_\theta f(\mathbf{p}, \theta) dA(\mathbf{p})} + \boxed{\int_{\partial I(\theta)} \mathbf{n}^\top \partial_\theta \mathbf{p}_b \Delta f(\mathbf{p}_b, \theta) d\ell(\mathbf{p}_b)}, \quad (2.44)$$

an *interior* term and a *boundary* term. The interior integral is over the image and corresponds to “differentiating under the integral”. The boundary integral is over the discontinuity curves in image space ∂I , and, intuitively, represents the variation of $\tilde{I}$ due to “movement” of these discontinuities with θ . Here, each discontinuity point, or *boundary point*, $\mathbf{p}_b$ has an associated normal $\mathbf{n}$ that is perpendicular to the discontinuity curve, and

$$\Delta f(\mathbf{p}) = \lim_{\varepsilon \rightarrow 0^+} f(\mathbf{p} + \varepsilon \mathbf{n}) - \lim_{\varepsilon \rightarrow 0^-} f(\mathbf{p} + \varepsilon \mathbf{n}) \quad (2.45)$$

is the difference between the function values of f on either side of the discontinuity.

Unfortunately, this means that differentiating physically-based renderers is not as simple as automatically differentiating the numerical computations. To see why, recall that the integral in Eq. (2.41) is typically computed using Monte Carlo methods (see Section 2.1.3), where the estimator is a sum over discrete samples:

$$\tilde{I} \approx \tilde{I}_N = \frac{1}{N} \sum_{i=1}^N \frac{f(\mathbf{p}_i, \theta)}{p(\mathbf{p}_i)}. \quad (2.46)$$

However, the (automatic) derivatives of this estimator only estimate the interior term:

$$\partial_\theta \tilde{I}_N = \frac{1}{N} \sum_{i=1}^N \frac{\partial_\theta f(\mathbf{p}_i, \theta)}{p(\mathbf{p}_i)} \approx \int_{I \setminus \partial I(\theta)} \partial_\theta f(\mathbf{p}, \theta) dA(\mathbf{p}). \quad (2.47)$$

For the above primary visibility example, the interior term is zero because $\partial_\theta f$ is zero on $I \setminus \partial I(\theta)$, so automatic differentiation simply produces zero derivatives for the image.

Instead, one would need to explicitly compute the boundary term, which, when done using Monte Carlo estimation, means sampling the discontinuities. In the simplified setting above, this corresponds to integrating over the boundary of the triangle $\mathcal{T}(\theta)$ in image space. In general scenes, however, this is significantly more difficult because discontinuities arise at silhouettes, which are determined by the occlusion relation of primitives. Even worse, this affects not only primary visibility but also the integrals appearing in the rendering equation (2.4).

A significant part of the research on physically-based differentiable rendering is therefore concerned with efficiently handling visibility discontinuities. Directly estimating the boundary term by sampling discontinuities has been investigated for unidirectional path tracing [Li et al., 2018a; Soroka et al., 2025] and the path integral form [Zhang et al., 2020, 2023]. A fundamental observation, also discussed in Chapter 5 and Chapter 6, is that the “parameter-dependence” of discontinuities depends on the parameterization of the integration domain. *Reparameterization* techniques attempt to construct parameterizations where discontinuities are static [Loubet et al., 2019; Bangaru et al., 2020; Xu et al., 2023; Wu et al., 2025], which effectively moves all contributions into the interior term. The same approach works for various surface representations, for example, signed distance functions [Bangaru et al., 2022; Vicini et al., 2022]. For a more comprehensive and recent overview of the topic, interested readers are referred to the survey by Zeng et al. [2025a].

2.4.3 Path Replay Backpropagation

When the task is to differentiate physically-based renderers (Section 2.1), a natural choice would be automatic differentiation (Section 2.3) because even if parameter-dependent discontinuities are present, automatic differentiation can be used to compute the interior term of the derivative (Section 2.4.2). However, tape-based automatic differentiation requires immense amounts of memory to store the computational graph of complex light transport simulations. In practice, this limits its applicability to simple scenes, short light paths, and low sample counts.

Nimier-David et al. [2020] and Stam [2020] showed that the derivatives of light transport simulations can be computed using another form of transport simulation. Building on this idea, Vicini et al. [2021] proposed *path replay backpropagation* (PRB), a (reverse-mode) differentiation algorithm that computes derivatives using constant memory and linear time in path length. This approach makes it possible to differentiate complex simulations without storing large computational graphs spanning the full computation.

Path replay backpropagation is an important component in Chapter 6, where we extend the same algorithm to time-resolved energy transport. Our derivation is based on a recursive formulation introduced in the following. Besides supporting the derivation in Chapter 6, we believe it also provides another helpful perspective on the algorithm.

Preliminaries Assume without loss of generality that we want to differentiate an image with respect to a single parameter θ : One has to differentiate the radiance measured at the sensor, which in turn leads to differentiating the equations defining light transport (Section 2.1.2):

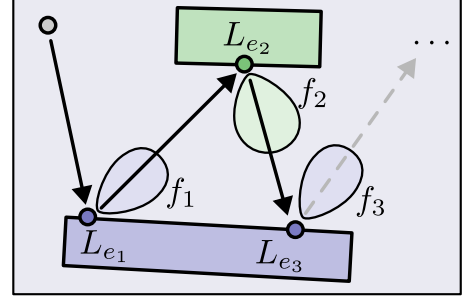
$$\partial_\theta L_o = \partial_\theta L_e + \int_{S^2} f \partial_\theta L_i + \partial_\theta f L_i d\omega_i^\perp, \quad (2.48)$$

where it is assumed, for now, that the integrand is continuous in θ .

The first part of the equation is similar to the standard rendering equation, but instead of radiance, *differential radiance* is collected. The second term inside the integral describes how θ affects the BSDF and therefore the contribution of all *remaining incoming radiance* to the derivative. Nimier-David et al. [2020] observe that, if one can

obtain an estimate of L_i (all remaining incoming radiance), at each point, the derivative can be computed similar to the radiance in forward rendering. They approximate L_i by a nested Monte Carlo simulation, leading to quadratic complexity, as two rays are spawned at every hit: one for collecting further differential radiance and one for the remaining incoming radiance. Path replay backpropagation [Vicini et al., 2021] avoids the nesting by computing L_i from the *total incoming radiance*, the radiance collected at the sensor, which can be precomputed in the forward simulation. In the following, we derive formulas for calculating L_i that capture the core idea.

Recursive Formula for the Total Incoming Radiance The total incoming radiance is usually approximated using Monte Carlo integration: given a simulation depth N , draw N direction samples $(\omega_{i_1}, \dots, \omega_{i_N})$ and trace the path consisting of multiple surface interactions (bounces). At each bounce the directions of the incoming and outgoing rays are known, so the emission L_{e_k} and BSDF value f_k can be computed (see inset).



The contribution of this sampled path to the total radiance is given by (division by the probability density function (pdf) omitted)

$$L_N = L_{e_1} + f_1 L_{e_2} + f_1 f_2 L_{e_3} + f_1 f_2 f_3 L_{e_4} + \dots \quad (2.49)$$

and can be computed using an iterative formula

$$\begin{aligned} \beta_0 &= 1, & \beta_k &= \beta_{k-1} f_k, \\ L_0 &= 0, & L_k &= L_{k-1} + \beta_{k-1} L_{e_k}, \end{aligned} \quad (2.50)$$

where β_k is the *path throughput*, defining the weight of the incoming radiance after k bounces. This shows that computing the total radiance is possible with constant memory (for β_k and L_k), independent of depth, simply by looping over all bounces.

Recursive Formula for the Derivative To differentiate the total radiance w.r.t. a scene parameter θ , we assume that the scene is static, leading to a continuous integrand in θ (the consequences of this rather strong assumption are discussed at the end of this section). Consequently, one can change the order of differentiation and integration. This allows computing the derivative by summing the derivatives of all Monte Carlo samples, each given by:

$$\begin{aligned} \partial_\theta L_N &= \partial_\theta L_{e_1} \\ &+ \partial_\theta f_1 L_{e_2} + f_1 \partial_\theta L_{e_2} \\ &+ \partial_\theta f_1 f_2 L_{e_3} + f_1 \partial_\theta f_2 L_{e_3} + f_1 f_2 \partial_\theta L_{e_3} + \dots \end{aligned} \quad (2.51)$$

Vicini et al. [2021] rearrange the derivative into “bounce-wise” terms

$$\begin{aligned} \partial_\theta L_N &= \partial_\theta L_{e_1} + \partial_\theta f_1 (L_{e_2} + f_2 L_{e_3} + \dots) \quad (\text{Bounce 1}) \\ &+ f_1 [\partial_\theta L_{e_2} + \partial_\theta f_2 (L_{e_3} + \dots)] \quad (\text{Bounce 2}) \\ &+ f_1 f_2 [\partial_\theta L_{e_3} + \partial_\theta f_3 (\dots)] \quad (\text{Bounce 3}) + \dots, \end{aligned} \quad (2.52)$$

revealing the analogy between Eq. (2.48) and the derivative of a single Monte Carlo sample. This leads to a recursive formula for the derivative:

$$\begin{aligned}\partial_\theta L_0 &= 0, \\ \partial_\theta L_k &= \partial_\theta L_{k-1} + \beta_{k-1} [\partial_\theta L_{e_k} + \partial_\theta f_k (L_N - L_k) / \beta_k].\end{aligned}\tag{2.53}$$

This allows computing $\partial_\theta L_N$ in a loop similar to L_N : at each bounce, add (1) the derivative of the current emitted radiance $\partial_\theta L_{e_k}$, and (2) the derivative of the current BSDF value $\partial_\theta f_k$, multiplied by the remaining incoming radiance $L_{k,N} / \beta_k = (L_N - L_k) / \beta_k$. Both terms are weighted with the previous path throughput β_{k-1} . For the latter term this multiplication cancels the division by β_k , by definition of β_{k-1} , resulting in a division by f_k only. The derivatives $\partial_\theta L_{e_k}$ and $\partial_\theta f_k$ can be evaluated locally using (reverse-mode) AD, without storing a computational graph that spans multiple loop iterations.

While the remaining incoming radiance $L_{k,N}$ is unknown a-priori, one can precompute L_N in a *primal pass* and then “replay” the same path in an *adjoint pass* to compute L_k and $L_{k,N} = L_N - L_k$, by iterating over the sequence of same intersections. In fact, instead of storing L_N and L_k in the adjoint pass, Vicini et al. [2021] store $L_{k,N}$ in a single variable that gets updated in each iteration

$$L_{0,N} = L_N, \quad L_{k,N} = L_{k-1,N} - \beta_{k-1} L_{e_k}.\tag{2.54}$$

Non-Static Scenes The static scene assumption underlying this derivation has two implications: first, the integrand is continuous in θ ; second, the position of each intersection point on a light path is independent of θ . When considering non-static scenes, discontinuities in the integrand can be handled using numerous existing techniques (Section 2.4.2). However, as we will show in Chapter 5, the second implication, i.e., path geometry moving in θ , leads to an additional, non-local term in the recursive formula for the derivatives. This term is not handled by existing implementations, such that, at the time of writing, no implementation simultaneously computes unbiased derivatives with (P)RB *and* handles discontinuous integrands. We will introduce an implementation that combines PRB with discontinuity handling in Chapter 5.

CHAPTER 3

Differentiable Shadow Mapping

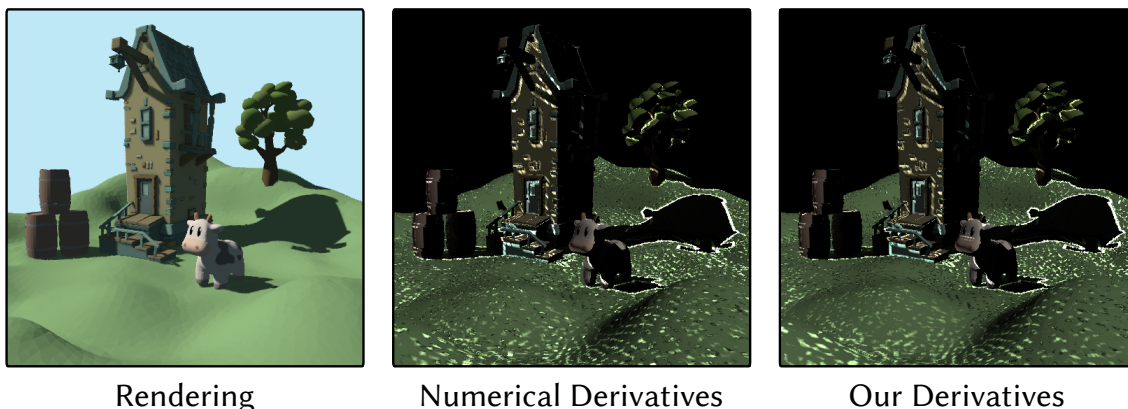


Figure 3.1: Complex scene (330k triangles) rendered in real time with differentiable shadow mapping. It shows self-shadowing of objects, shadowing between objects, and colored surfaces. Finite differences and our automatic derivatives w.r.t. movement of the light (note the non-zero derivatives at shadow boundaries).

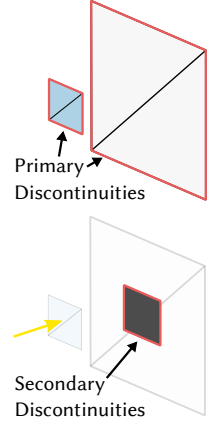
Many inverse rendering applications are concerned with recovering scene information solely from images captured in the real world. If this is to be solved using differentiable rendering, one seeks a renderer that can reproduce the various effects of physical light transport, as the goal is to match real-world images with renderings of a virtual scene. A natural choice are physically-based differentiable renderers [Li et al., 2018a; Nimier-David et al., 2019; Zhang et al., 2020; Jakob et al., 2022b] as they adhere to the equations of light transport (Section 2.1). However, as in forward rendering, the accuracy of these sophisticated systems comes at the price of efficiency.

The full picture shows that inverse rendering and other applications require differentiable renderers that strike a good balance between accuracy *and* efficiency, for example, when the renderer is only one of many components in a larger machine learning pipeline [Gao et al., 2022; Lattas et al., 2022]. Unsurprisingly, many techniques from real-time rendering have found their way into differentiable rendering, as highly realistic images can be efficiently generated by *approximating* light transport.

Despite the advances of modern real-time systems, and the potential to leverage additional information in optimizations, surprisingly little effort has been spent on approximating global effects like shadows or indirect illumination in differentiable rendering: The predominant approach for scenes with triangle meshes is differentiable rasterization with purely *local* shading models [Loper and Black, 2014; Chen et al., 2019, 2021]. While

this might lead to acceptable outcomes in some applications, ignoring effects like shadows in inverse settings can easily degrade solutions [Munkberg et al., 2022] and is generally only valid in very restrictive setups (e.g., with co-located camera and light [Zhang et al., 2022; Luan et al., 2021]).

In this chapter, we show how shadows can be efficiently generated in differentiable rendering of triangle meshes (Fig. 3.1). Our central observation is that one of the oldest and most widely used approximations, *shadow maps* [Williams, 1978] (Section 2.2.2), can be adapted to work in differentiable rendering frameworks. Our main idea is similar to the original one: exploit efficient rasterization from the light’s point of view. For differentiable shadows, this means that existing differentiable rasterizers handle discontinuities of primary visibility along primitive borders; we use this machinery to handle discontinuities of secondary visibility along shadow borders. In contrast to similar work [Lyu et al., 2021; Hasselgren et al., 2022; Gao et al., 2025], our approach (1) computes shadows directly from the triangle mesh instead of proxy geometry, (2) differentiates visibility with respect to movement of geometry, and (3) supports high-frequency shadows from point light sources. We demonstrate at several inverse graphics problems that *differentiable shadow mapping* is orders of magnitude faster than differentiable ray tracing, with similar accuracy – while differentiable rasterization without shadows often fails to converge.



3.1 Method

Given a point light source located at position $\mathbf{x}_l$ (possibly at infinity), our goal is to compute and differentiate the shadows it casts. Shadows are fully characterized by visibility: if the path between the light source and a point $\mathbf{x} \in \mathbb{R}^3$ in the scene is obstructed, the point is in shadow. Since visibility is entirely geometric, it only depends on the location of $\mathbf{x}$ and the scene geometry, which includes the pose of the light source:

$$\begin{array}{c} \text{dependency on scene geometry} \\ \downarrow \\ \text{Visibility } V(\mathbf{x}, \theta) \end{array} \quad (3.1)$$

We use the scene parameter θ from Section 2.4 to highlight this dependency and the desire to differentiate visibility not only with respect to $\mathbf{x}$ but also geometry.

The main difficulty with light visibility is its piecewise constant nature: it is either zero or one, so the derivative is zero almost everywhere, both for $\mathbf{x}$ and θ – the *shape* of shadows cast by a point light provides no additional cues to an inverse rendering optimization. Importantly, this is an artifact of (non-physical) point lights as the Dirac distribution collapses the integral that yields continuous visibility for (physical) area lights (see Section 2.2.1). Before we present our solution (Section 3.1.3), we first investigate the challenges with differentiating shadow mapping more closely (Section 3.1.1), and introduce the general technique underlying our approach (Section 3.1.2).

3.1.1 Discontinuities in Shadow Mapping

As shown in Section 2.2.2, the visibility in Eq. (3.1) can be computed using a shadow map f , which is a depth map from the perspective of the light:

$$V(\mathbf{x}, \theta) = \begin{cases} 0 & d(\mathbf{x}, \theta) > f(L(\mathbf{x}, \theta), \theta) \\ 1 & \text{else.} \end{cases} \quad (3.2)$$

This is exactly Eq. (2.18), only with an explicit dependency on scene parameters¹.

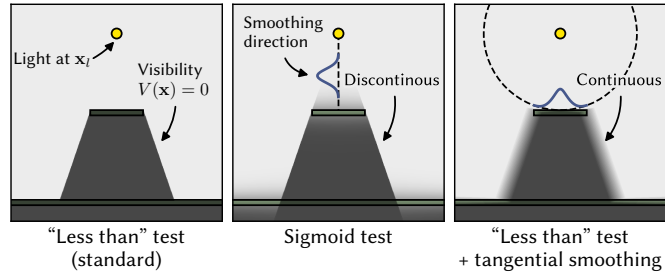
A common misconception in other constructions attempting to differentiate shadow mapping is that the main source of difficulties is the binary “less than” test: The comparison is replaced with a smooth test, typically using a sigmoid [Tiwary et al., 2022; Karunratanakul et al., 2023; Planche and Singh, 2021; Wang et al., 2023a],

$$V(\mathbf{x}, \theta) \approx 1 - \text{sigmoid}(\alpha(d(\mathbf{x}, \theta) - f(L(\mathbf{x}, \theta), \theta))), \quad (3.3)$$

with the expectation that this leads to useful shadow derivatives.

This is, however, not the case:

using a sigmoid test amounts to smoothing the visibility field $V(\mathbf{x})$ in *radial* direction, along light rays; but since the shadow map f is discontinuous at geometry silhouettes, any discontinuities in *tangential* direction are unaffected (see inset).



These tangential discontinuities are the *true* source of difficulties: the derivatives resulting from the sigmoid test remain blind to visibility changes across shadow boundaries and perturbations of the shadow outline by geometry. Therefore, we propose to consider a tangential smoothing of visibility, such that it continuously varies between zero and one across shadow boundaries².

3.1.2 An Inspiration from Real-Time Rendering

Smoothing (hard) shadow boundaries is commonly encountered in traditional real-time rendering as a way to (1) increase the realism, since most real-world shadows have some amount of softness, and (2) remedy aliasing at shadow boundaries resulting from the discretization of the shadow map (see [Hasenfratz et al., 2003] for an overview). While our goal is the same, the motivation is different: in differentiable rendering, smoothing is *not* optional, but a *necessity* for non-zero spatial and geometric shadow derivatives. Regardless, our idea is to reuse the same machinery.

¹Remember that $d(\mathbf{x}, \theta)$ is the distance and $L(\mathbf{x}, \theta)$ is the direction from the light to $\mathbf{x}$

²Some concurrent, sigmoid-based constructions [Karunratanakul et al., 2023; Wang et al., 2023a] also consider a smoothing of shadow boundaries, but their derivatives are incorrect and have the same limitations as non-smoothed versions (see Section 3.2).

Percentage Closer Filtering. Many approaches can be traced back to or interpreted as approximations of *percentage closer filtering* (PCF) [Reeves et al., 1987]: rather than comparing $d(\mathbf{x})$ to only $f(L(\mathbf{x}))$, consider a small neighborhood around $L(\mathbf{x})$ in the shadow map and return the *percentage* of the rays to which $\mathbf{x}$ is visible. In a slight generalization of the original formulation, we assume the neighborhood to be implicitly defined by a continuous (compactly supported) kernel $k_{L(\mathbf{x})}(\mathbf{u})$ satisfying $\int k_{L(\mathbf{x})}(\mathbf{u}) d\mathbf{u} = 1$. Then the visibility is

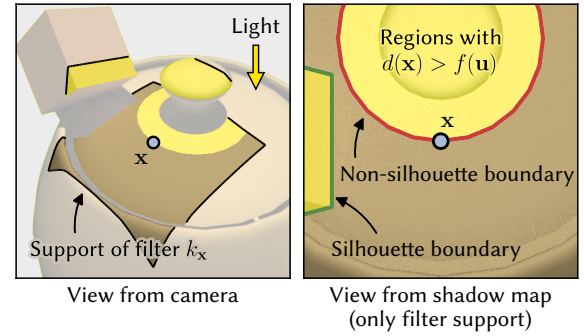
$$V_{\text{PCF}}(\mathbf{x}) = \int_{d(\mathbf{x}) \leq f(\mathbf{u})} k_{L(\mathbf{x})}(\mathbf{u}) d\mathbf{u}. \quad (3.4)$$

The visibility function is taking on values in $[0, 1]$, and, in particular, the boundaries of shadows are smoothly transitioning between the two extremes.

Derivatives of Percentage Closer Filtering. Despite being based on a binary test, differentiating the visibility function of percentage closer filtering yields the expected shadow derivatives. However, this must be done with care: simply applying automatic differentiation to the discretized integral (e.g., as in Karunratanakul et al. [2023]) misses the derivatives for shadow boundaries. Instead, the integral must be differentiated in its *continuous* form – one of the reasons we have not yet limited f to be a raster image. This is because the boundary of the integration domain $d(\mathbf{x}, \theta) \leq f(\mathbf{u}, \theta)$ depends on $\mathbf{x}$ and θ : when differentiating visibility with respect to either parameter, one must consider their effect on the integration domain, which involves an integral over its boundary [Reynolds et al., 1903]. Automatically differentiating the discretized integral, i.e., “Discretize-then-Differentiate” [Bangaru et al., 2021], is known to miss these boundary derivatives.

Conceptually, the problem is similar to visibility handling in differentiable rendering (Section 2.4.2), but with a key difference: while the integration domain boundaries in the former are simply object silhouettes, the boundaries in percentage closer filtering depend on how the points in the filter support geometrically relate to $\mathbf{x}$ and the light source. Those boundaries *can* include (subsets of) object silhouettes, but generally they are curves on the objects.

Even if one could integrate over these boundaries, percentage closer filtering is costly: not only the integral but also its derivative need to be evaluated for *every* point $\mathbf{x}$ visible to the camera. One would much rather *pre-filter* the shadow map instead of the comparison to $d(\mathbf{x})$. Yet, simply convolving f with k and then evaluating the smoothed shadow map is different – the result would take only the discrete values 0 and 1.



3.1.3 Differentiable Pre-Filtered Shadow Mapping

A practical solution to differentiable shadow mapping would generate soft shadow boundaries such that visibility can be differentiated (Section 3.1.1), but, at the same time, does not involve an integral for every point $\mathbf{x}$ that is costly to compute and even more costly to differentiate. While percentage closer filtering [Reeves et al., 1987] does not meet the

second criterion (see Section 3.1.2), its *approximations* do: Several techniques pre-warp f and/or pre-filter not only f but also simple functions $g(f)$, such as $g(f) = f^k$ or $g = \exp$ [Annen et al., 2007, 2008; Donnelly and Lauritzen, 2006; Peters and Klein, 2015]:

$$f_i : \mathcal{U} \mapsto \mathbb{R}, \quad f_i(\mathbf{u}) = \int_{\mathcal{U}} k_{\mathbf{u}}(\mathbf{p}) g_i(f(\mathbf{p})) d\mathbf{p}, \quad (3.5)$$

where f_i , like the shadow map, is defined over the space of light rays $\mathcal{U}$ (see Section 2.2.2). Then, an approximation to Eq. (3.4) is built from the set of pre-computed functions:

$$V_{\text{PCF}}(\mathbf{x}) \approx V_{\text{Approx}}(\mathbf{x}, f_1(L(\mathbf{x})), f_2(L(\mathbf{x})), \dots). \quad (3.6)$$

This structure is so useful in forward rendering because the integral is independent of $\mathbf{x}$, allowing it to be evaluated using hardware texture filtering; and V_{Approx} is typically a simple function. The structure is equally advantageous in differentiable rendering: under the mild assumption that $k_{\mathbf{u}}$ and g_i are continuous, the only discontinuities in the integrand relevant for differentiation are those of the shadow map f at object silhouettes. This effectively reduces the problem to visibility handling (Section 2.4.2).

Discretize and Smooth with Differentiable Rasterizers

Like the shadow map f , the pre-filtered maps f_i are not evaluated at query-time but precomputed by rendering from the perspective of the light. In practice, this means the functions f_i will be approximated by discrete versions $\mathbf{f}_i$. The main challenge is to ensure that the derivatives $\partial_{\theta}\mathbf{f}_i$ account for changes of geometry, including object silhouettes.

Similar in spirit to the original shadow map idea [Williams, 1978], we can use existing rasterizers. While the original motivation was efficiency, and this is also true in our case, a central point is that significant effort has been spent on visibility handling when rendering from the perspective of the viewer [Laine et al., 2020; Liu et al., 2019a; Loper and Black, 2014; Kato et al., 2018; Chen et al., 2019, 2021; de La Gorce et al., 2008]. We can readily exploit any existing solution and render from the perspective of the light.

This works because the output of differentiable rasterizers is, effectively, a smooth point-sampled version of the function in screen space: the output values vary continuously with the scene geometry. This means, $\mathbf{f}_i$ can be generated by rendering $g_i(f)$ with a differentiable rasterizer, optionally followed by a (discrete) convolution with any filter kernel. Since pre-filtered shadow mapping *requires* smoothing to attenuate aliasing, the smoothing by differentiable rasterizers is complementary in this setting.

Variance Shadow Maps

While our approach works with any pre-filtering technique, we demonstrate it at the example of *variance shadow maps* (VSM) [Donnelly and Lauritzen, 2006]. The approximation is built by taking the view that the local shadow map may be considered a probability distribution. This makes sense because it is irrelevant what points are in the integration domain in Eq. (3.4) as long as the measure of the integration domain is correct.

Pre-filtering is used to generate the first and second moments of the local distribution

$$m_1(\mathbf{u}) = \int_{\mathcal{U}} k_{\mathbf{u}}(\mathbf{p}) f(\mathbf{p}) d\mathbf{p} \quad m_2(\mathbf{u}) = \int_{\mathcal{U}} k_{\mathbf{u}}(\mathbf{p}) f(\mathbf{p})^2 d\mathbf{p}, \quad (3.7)$$

which means that $g_1(f) = f$ and $g_2(f) = f^2$ in the context of Eq. (3.5). Given a point $\mathbf{x}$, the mean and variance of the local distribution in the shadow map can be computed as

$$\mu_{\mathbf{x}} = m_1(L(\mathbf{x})) \quad \sigma_{\mathbf{x}}^2 = m_2(L(\mathbf{x})) - \mu_{\mathbf{x}}^2. \quad (3.8)$$

The observation is that $V_{\text{PCF}}(\mathbf{x})$ is a measure, and since the shadow map f is positive, the best bound for this measure is given by the *one-sided Chebyshev* or *Cantelli* inequality

$$V_{\text{PCF}}(\mathbf{x}) \leq \frac{\sigma_{\mathbf{x}}^2}{\sigma_{\mathbf{x}}^2 + (d(\mathbf{x}) - \mu_{\mathbf{x}})^2}, \quad (3.9)$$

Notice that, it may happen that $d(\mathbf{x}) < \mu_{\mathbf{x}}$ because $\mu_{\mathbf{x}}$ is a filtered version of f . Since in this case the approximation of the visibility function would decrease as $d(\mathbf{x})$ comes closer to the light source, Donnelly and Lauritzen [2006] suggest the following, slightly modified visibility function:

$$V_{\text{VSM}}(\mathbf{x}) = \begin{cases} \frac{\sigma_{\mathbf{x}}^2}{\sigma_{\mathbf{x}}^2 + (d(\mathbf{x}) - \mu_{\mathbf{x}})^2}, & d(\mathbf{x}) > \mu_{\mathbf{x}} \\ 1 & \text{else.} \end{cases} \quad (3.10)$$

The resulting shadows are observed to be very similar to percentage closer filtering.

Derivatives Our goal is to differentiate the visibility function of variance shadow mapping with respect to the scene geometry θ and the evaluation point $\mathbf{x}$.

First, notice that the elements in V_{VSM} depending on the shadow map f are $\mu_{\mathbf{x}}$ and $\sigma_{\mathbf{x}}^2$. Let $\tilde{f}$ and $\tilde{f}^2$ represent the smoothed and discretized versions of f and f^2 generated by a differentiable rasterizer, with $\partial_{\theta}\tilde{f}$ and $\partial_{\theta}\tilde{f}^2$ being their derivatives for θ . The moments are convolutions that, in the discrete case, can be expressed as the scalar products

$$\mathbf{m}_1 = \mathbf{G}\tilde{f} \quad \mathbf{m}_2 = \mathbf{G}\tilde{f}^2 \quad (3.11)$$

where $\mathbf{G}$ is a Toeplitz matrix representing the filter kernel. The derivatives are then

$$\partial_{\theta}\mathbf{m}_1 = \frac{\partial\mathbf{m}_1}{\partial\tilde{f}}\partial_{\theta}\tilde{f} = \mathbf{G}\partial_{\theta}\tilde{f} \quad \partial_{\theta}\mathbf{m}_2 = \frac{\partial\mathbf{m}_2}{\partial\tilde{f}^2}\partial_{\theta}\tilde{f}^2 = \mathbf{G}\partial_{\theta}\tilde{f}^2. \quad (3.12)$$

The functions m_i ($i \in \{1, 2\}$) are approximated using the discrete moments $\mathbf{m}_i$:

$$m_i(L(\mathbf{x})) \approx \mathbf{w}(L(\mathbf{x}))^{\top} \mathbf{m}_i. \quad (3.13)$$

where $\mathbf{w}(\mathbf{u})$ ($\|\mathbf{w}\|_1 = 1$) are non-negative weights assigned to each point sample (e.g. interpolation weights). From this, the derivatives of the mean $\mu_{\mathbf{x}}$ directly follow to be:

$$\partial_{\theta}\mu_{\mathbf{x}} = \mathbf{w}^{\top} \partial_{\theta}\mathbf{m}_1 + \mathbf{m}_1^{\top} (\partial_{\mathbf{u}}\mathbf{w} \partial_{\theta}L) \quad \partial_{\mathbf{x}}\mu_{\mathbf{x}} = \mathbf{m}_1^{\top} (\partial_{\mathbf{u}}\mathbf{w} \partial_{\mathbf{x}}L). \quad (3.14)$$

$L(\mathbf{x})$ is a linear projective transformation that is straightforward to differentiate (see Section 2.2.2) but notice that the choice of weights matters: if $\mathbf{w}(\mathbf{u})$ represents nearest neighbor interpolation (i.e., indexing with integer coordinates), we get $\partial_{\mathbf{u}}\mathbf{w} = \mathbf{0}$, in which case the derivative misses important variations of visibility with the light position and

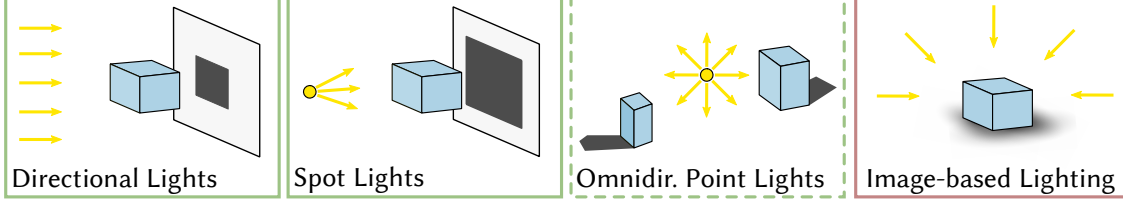


Figure 3.2: We implement directional lights and spot lights in our framework, and have experimental support for omnidirectional point lights. Image-based lighting (IBL) is not supported, but the illumination from some environment maps can be sufficiently well approximated by a collection of point lights.

the evaluation point. Therefore, we have $\mathbf{w}(\mathbf{u})$ represent bilinear interpolation weights, such that $\partial_{\mathbf{u}}\mathbf{w} \neq \mathbf{0}$. This is reasonable because the convex combination of moments behaves just like another filter. The derivatives for σ^2 can be found in a similar way:

$$\partial_{\theta}\sigma_{\mathbf{x}}^2 = \mathbf{w}^{\top}\partial_{\theta}\mathbf{m}_2 + \mathbf{m}_2^{\top}(\partial_{\mathbf{u}}\mathbf{w}\partial_{\theta}L) - 2\mu_{\mathbf{x}}\partial_{\theta}\mu_{\mathbf{x}} \quad \partial_{\mathbf{x}}\sigma_{\mathbf{x}}^2 = \mathbf{m}_2^{\top}(\partial_{\mathbf{u}}\mathbf{w}\partial_{\mathbf{x}}L) - 2\mu_{\mathbf{x}}\partial_{\mathbf{x}}\mu_{\mathbf{x}}. \quad (3.15)$$

The distance $d(\mathbf{x})$ is independent of the shadow map. Its variation with θ stems from possible modification of the light source position, which can be easily expressed.

Lastly, V_{VSM} is a piecewise rational function of these variables, with the boundary between the two pieces at $d = \mu_{\mathbf{x}}$ depending on both d and $\mu_{\mathbf{x}}$. This requires no special treatment, however, because it is a continuously differentiable function across $d = \mu_{\mathbf{x}}$:

$$\frac{\partial V_{\text{VSM}}}{\partial \mu_{\mathbf{x}}} = -\frac{\partial V_{\text{VSM}}}{\partial d} = \begin{cases} 2\frac{\sigma_{\mathbf{x}}^2(d(\mathbf{x})-\mu_{\mathbf{x}})}{(\sigma_{\mathbf{x}}^2+(d(\mathbf{x})-\mu_{\mathbf{x}})^2)^2} & d(\mathbf{x}) > \mu_{\mathbf{x}} \\ 0 & \text{else} \end{cases}. \quad (3.16)$$

Thus, $\partial_{\theta}V_{\text{VSM}}$ and $\partial_{\mathbf{x}}V_{\text{VSM}}$ can be directly computed using the chain rule.

3.2 Implementation and Parameter Exploration

Our reference implementation is in Python³ and builds on `NVDIFFRAST`, the differentiable rasterization primitives by Laine et al. [2020], for rendering and smoothing the shadow map. The derivatives $\partial_{\theta}\tilde{\mathbf{f}}$ and $\partial_{\theta}\tilde{\mathbf{f}}^2$ (see Section 3.1.3) are not explicitly computed. Rather, we use reverse-mode differentiation to obtain the gradients of an objective function. The relevant derivatives are computed automatically [Paszke et al., 2019]. The implementation currently supports directional lights and spot lights (a single shadow map per light), and has experimental support for omnidirectional point lights (six shadow maps per light). Area lights or environment lighting would have to be approximated by point light sources – we leave a more accurate treatment for future work. Fig. 3.2 illustrates the light sources supported in our current implementation. For all experiments – except those explicitly stating otherwise – we use a shadow map resolution of 256×256 .

³An open-source implementation is available at <https://github.com/mworchel/differentiable-shadow-mapping>

By building on an existing differentiable rasterizer and automatic differentiation, our code becomes rather straightforward. The pseudo-code below is provided for a single light source – keep in mind that the methods have to be executed for each light. The pre-computation of the pre-filtered depth images $\mathbf{m}_1$ and $\mathbf{m}_2$ is described in Alg. 1 below. Here, we assume that the `RENDERSCENE` function returns the smoothed discrete depth image $\tilde{\mathbf{f}}$ as well as $\tilde{\mathbf{f}}^2$. The modular design of `NVDIFFRAST` implements this in two steps, first providing the discrete values $\mathbf{f}$ and $\mathbf{f}^2 = \mathbf{f} \otimes \mathbf{f}$, which are then smoothed using the `antialias` function. We implement the filtering as a convolution with a kernel k of configurable size (e.g. a Gaussian). Once the discrete smoothed shadow maps have been computed, the visibility for a point $\mathbf{x} \in \mathcal{X}$ can be computed with Alg. 2.

Algorithm 1 Render a variance shadow map for a light source

Inputs: scene geometry $\mathcal{X}$, light source position $\mathbf{p}$, filter kernel k

Output: a variance shadow map $(\mathbf{m}_1, \mathbf{m}_2)$

```

function RENDERSHADOWMAP( $\mathcal{X}, \mathbf{p}, k$ )
     $\tilde{\mathbf{f}}, \tilde{\mathbf{f}}^2 \leftarrow \text{RENDERSCENE}(\mathcal{X}, \mathbf{p})$ 
     $\mathbf{m}_1 \leftarrow \text{FILTER}(\tilde{\mathbf{f}}, k)$ 
     $\mathbf{m}_2 \leftarrow \text{FILTER}(\tilde{\mathbf{f}}^2, k)$ 
    return  $\mathbf{m}_1, \mathbf{m}_2$ 

```

Algorithm 2 Compute the visibility using variance shadow mapping

Inputs: a point in space $\mathbf{x}$, light source position $\mathbf{p}$, variance shadow map $(\mathbf{m}_1, \mathbf{m}_2)$

Output: the visibility $V_{\text{VSM}}(\mathbf{x})$

```

function COMPUTEVISIBILITY( $\mathbf{x}, \mathbf{p}, \mathbf{m}_1, \mathbf{m}_2$ )
     $\mathbf{u} \leftarrow L(\mathbf{x}, \mathbf{p})$ 
     $m_1 \leftarrow \text{SAMPLEBILINEAR}(\mathbf{m}_1, \mathbf{u})$ 
     $m_2 \leftarrow \text{SAMPLEBILINEAR}(\mathbf{m}_2, \mathbf{u})$ 
     $d \leftarrow d(\mathbf{x}, \mathbf{p}), \quad \mu \leftarrow m_1, \quad \sigma^2 \leftarrow m_2 - m_1^2$ 
    if  $d \leq \mu$  then return 1
    else return  $\sigma^2 / (\sigma^2 + (d - \mu)^2)$ 

```

Optimization without Smoothed Shadow Map

We test our hypothesis that the smoothing of the differentiable rasterizer, giving $\tilde{\mathbf{f}}$ and $\tilde{\mathbf{f}}^2$, is essential for geometry derivatives. This is facilitated by the modularity of `NVDIFFRAST` that allows disabling the smoothing, thus working only with the non-smoothed but point-sampled versions $\mathbf{f}$ and $\mathbf{f}^2$.

Our setup is as simple as possible: a single square occluder casts a single shadow on a single square receiver. We optimize the position of one displaced vertex of the occluder to reproduce the shadow in a reference configuration (see Fig. 3.3). The optimization minimizes the mean squared error to the reference shadow image.

We find that the target configuration is not attained when using the discontinuous discrete shadow map for generating the visibility (i.e., without using `NVDIFFRAST`'s `antialias` function). Importantly, this result is independent of the kernel k used for

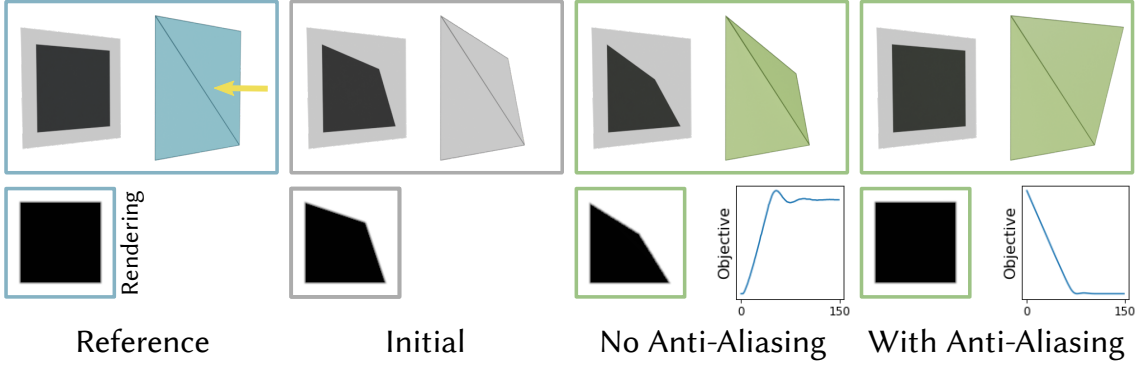
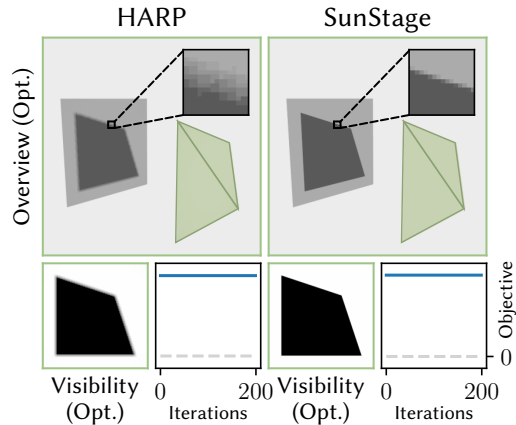


Figure 3.3: Pre-filtering of shadow maps must include a technique for handling visibility discontinuities to make shadow mapping differentiable. Most differentiable renderers implement such a technique: we use anti-aliasing of NVDIFFRAST [Laine et al., 2020] and show the convergence behavior when optimizing a single vertex on the occluder plane.

pre-filtering to generate the moments $\mathbf{m}_1$ and $\mathbf{m}_2$. In other words, filtering $\mathbf{f}$ and $\mathbf{f}^2$ cannot recover the connection between the scene parameters and the shadow boundaries. Using the smooth functions (i.e., applying antialias to $\mathbf{f}$ and $\mathbf{f}^2$) shows the expected convergence across variations of all other parameters.

Visibility Derivatives in Concurrent Work

In concurrent work, Wang et al. [2023a] (“SunStage”) and Karunratanakul et al. [2023] (“HARP”) propose differentiable shadow mapping for facial capture and hand reconstruction, respectively. Both constructions use a sigmoid-based visibility test, giving as a reason that this ensures “smooth gradients”, and Karunratanakul et al. [2023] explicitly note that the shadow mapping implementation is differentiable with respect to geometry. However, as we have discussed in Section 3.1.1, a sigmoid test amounts to radially smoothing visibility: the resulting visibility remains discontinuous across shadow boundaries, and the derivatives with respect to scene geometry do not reflect the variation of the shadow outline. Interestingly, in their implementation, *both* methods also use another form of smoothing, which, visually, leads to soft shadow boundaries: Karunratanakul et al. [2023] use percentage closer filtering [Reeves et al., 1987] and Wang et al. [2023a] spatially filter visibility in the image space of the camera using a box filter. While this produces soft shadow boundaries in the generated image, the visibility derivatives for geometry remain zero (the minimal optimization task shown in Fig. 3.3 does not converge, see inset). This is because neither implementation differentiates the integral *expression* underlying the filtering; instead, both automatically differentiate the discretized integral. This corresponds to “Discretize-then-Differentiate” [Bangaru et al., 2021], which is



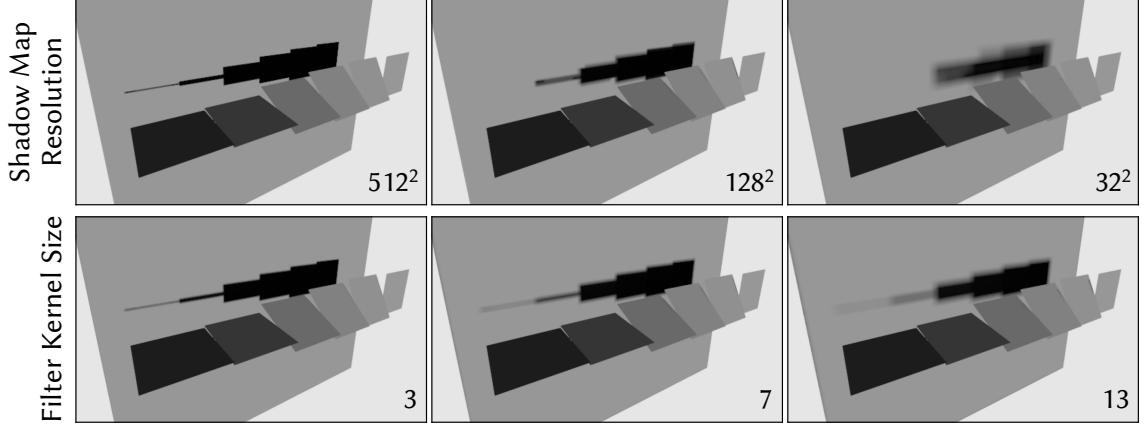


Figure 3.4: Effect of shadow map resolution (top row) and filter size k (bottom row) on the shadows cast by differently slanted planes. The filter size is its radius in pixels.

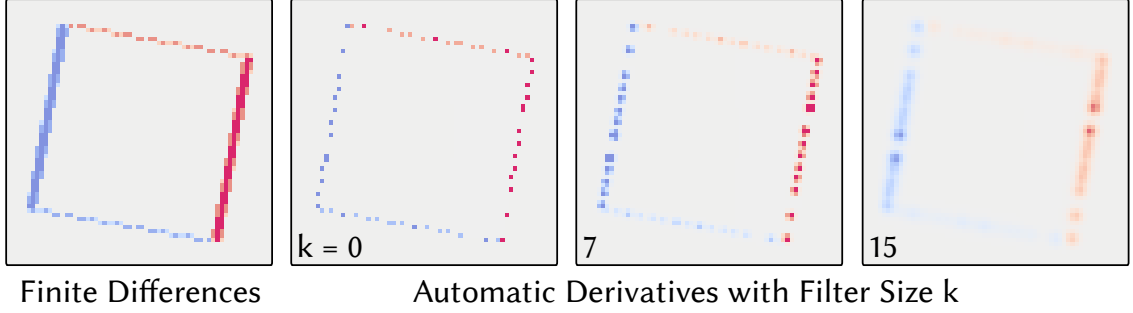


Figure 3.5: Derivative of the visibility $\partial_t V_{SM}$, where t is the horizontal position of a highly tessellated occluder over a planar receiver, for filters with varying size.

known to produce incorrect derivatives if the integrand contains parameter-dependent discontinuities. Here, this is the case because the visibility discontinuities in the integrand (i.e., shadow outlines) depend on the scene geometry. Unfortunately, it may be difficult to correctly differentiate either approach: for percentage closer filtering, one would need to sample boundaries that depend on the evaluation point (see Section 3.1.2), and for image-space filtering, one would need to sample arbitrary cast shadow boundaries in image space, which appears similarly challenging.

Shadow Map Resolution and Filter Kernel

The shadow map resolution and the filter kernel size are key parameters as they directly affect the quality of the shadow approximation (see Fig. 3.4). Instead of investigating the *forward* pass of shadow map rendering, we focus on the *backward* pass, because it influences the gradient computation and any optimization based on it.

Consider a scene similar to the one above, now with a highly tessellated and slightly rotated occluder. High tessellation (relative to shadow map pixels) results in sparse shadow map derivatives for occluder geometry because the anti-aliasing misses primitive boundaries. This might occur for other differentiable rasterizers relying on explicit boundary detection. The filter applied to recover the moments (Eq. (3.11)) is also ap-

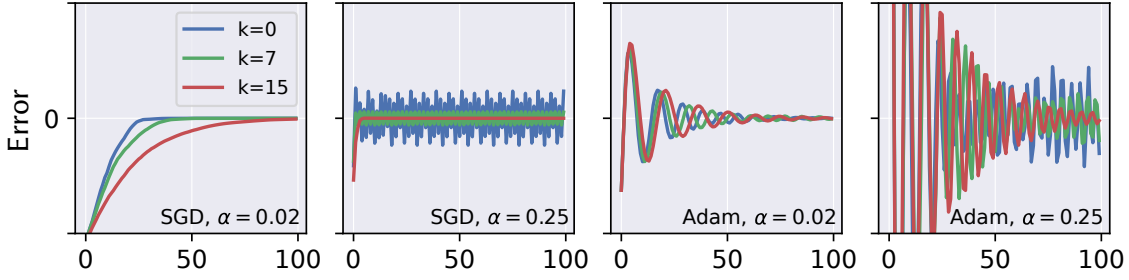


Figure 3.6: Convergence plots different filter kernel sizes k . We displace the occluder shown in Fig. 3.5 horizontally, and the goal of the optimization is to recover the original position, only by using the shadow images. Stochastic gradient descent (left) and ADAM [Kingma and Ba, 2017] (right) with small and large step sizes α . The y -axis shows the signed error of the horizontal position.

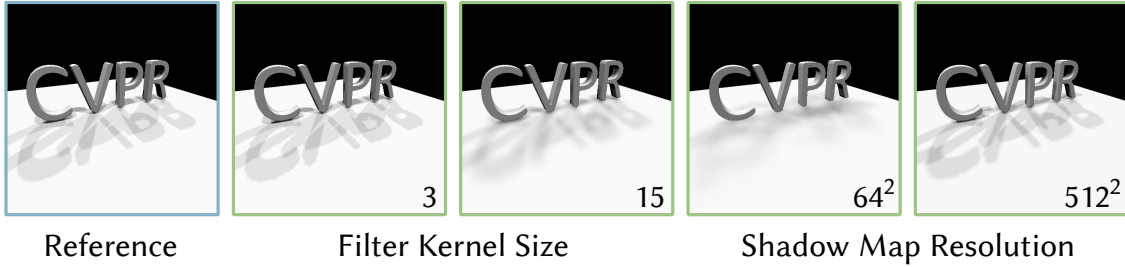


Figure 3.7: Light direction estimation with four lights, varying filter kernel sizes and shadow map resolutions.

plied to the derivatives of the shadow map (Eq. (3.12)), so it affects the derivative of the visibility function (see Fig. 3.5). While the filter kernel size is merely a visual effect in the forward pass, it can indeed affect the optimization: Fig. 3.6 shows that larger filter kernels increase the robustness of gradient descent, both for stochastic gradient descent and momentum-based variants. However, larger kernels also increase the runtime.

Low shadow map resolution may also lead to occluders being missed completely by the visibility test (see Fig. 3.4, top row). These issues can be alleviated by increasing the shadow map resolution; again at the cost of runtime.

In the following, we consider a more complex setting to test different shadow map resolutions and filter kernel sizes: an object is placed on a rectangular floor plane and illuminated by n directional lights (Fig. 3.7). Given a single reference image I_{ref} , we wish to recover the n light directions $\{l_i\}$, minimizing the mean squared image error.

We run the optimization for different objects and varying number of lights n . For each configuration of filter size or shadow map resolution, we repeat the optimization 150 times with randomized target and initial conditions (the randomization is deterministic, so all configurations use same set of conditions), and average the results. We report qualitative results in Fig. 3.7 and quantitative results in Fig. 3.8. As expected, increasing the shadow map resolution or the filter kernel size increases the runtime. The optimization behaves robustly with high accuracy and consistent error bounds across resolutions and filter sizes. Again, the accuracy improves slightly with larger filter size, albeit this

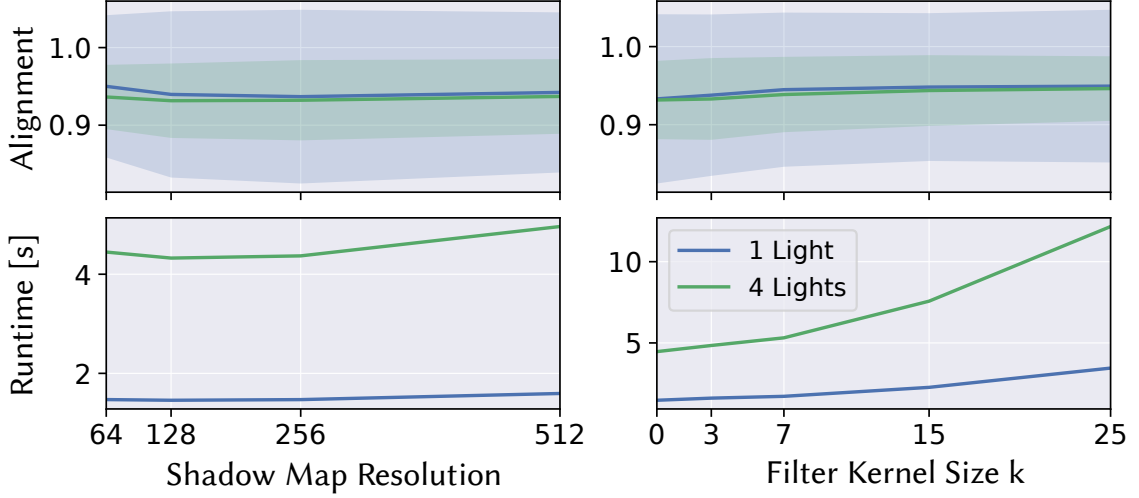


Figure 3.8: Quantitative results for the parameter exploration, using a light direction estimation experiment. The y -axis in the top row is the average alignment of predicted light directions to the reference light directions, measured by the dot product after greedily matching the two sets.

improvement seems marginal considering the error bounds. Lowering the shadow map resolution has a similar effect because the effective filter footprint increases if its size stays constant. We observe that the optimization for one light can get stuck in local minima where the shadows do not align but the shade of the floor plane matches that in the reference image. For multiple lights, shadows dominate the image, so this scenario is less likely; this could explain the different error bounds. Conversely, recovering multiple lights is more difficult, hence the lower average accuracy. These challenges are specific to the task and, in general, we observe that our method works robustly with different sets of parameters.

3.3 Applications

We run all experiments on a workstation with an NVIDIA 1070 Ti GPU with 8 GB VRAM.

3.3.1 Monocular Pose Estimation

Estimating the orientation and position of an object from images is a classic computer vision task. The problem becomes easier for sequences of images [Lepetit and Fua, 2005] or if priors apply, such as in pose estimation of human body parts [Sarafianos et al., 2016; de La Gorce et al., 2008]. Given only a single (monocular) image makes retrieving the rigid transformation of a known object ill-posed. Shafer and Kanade [Shafer and Kanade, 1983] noticed that the shadow of an object in the image provides viable cues for reducing the ambiguity. We consider a simplified version of the 6D problem, allowing only 2 translational and one rotational degree of freedom. We place an object in front of a receiver plane and a directional light orthogonal to the plane. In this setting, we recover

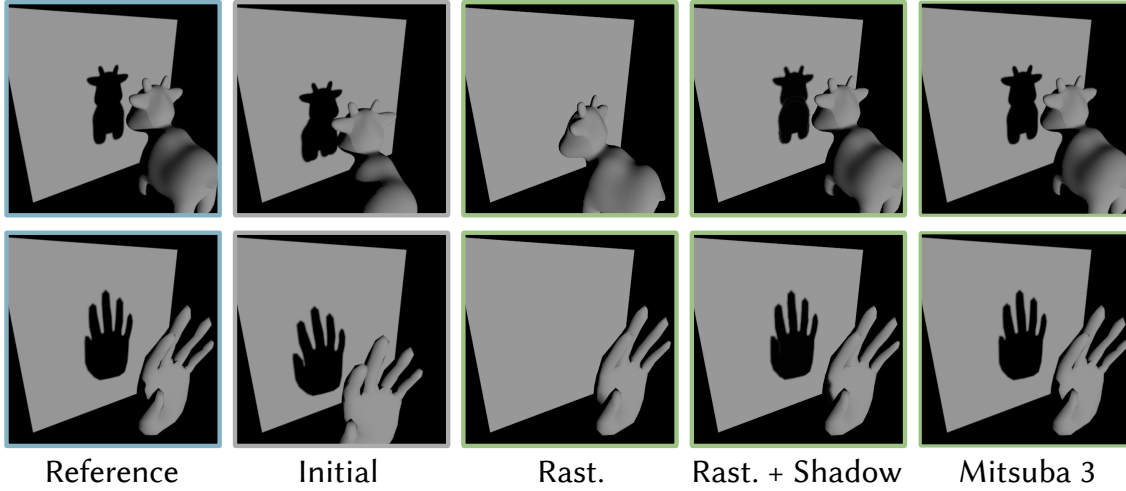
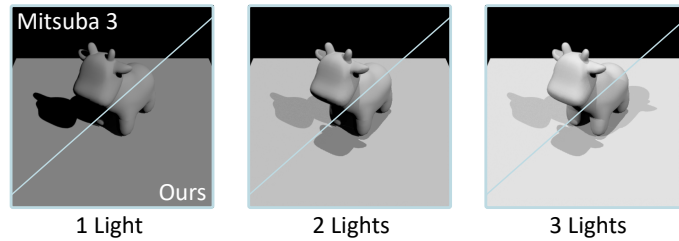


Figure 3.9: Pose estimation results with path-traced references. In most cases, shadows provide viable cues for convergence (top row) but occasionally local shading is sufficient (bottom row).

the parameters $\mathbf{t} = (x, y)$ (translation parallel to the plane) and φ (rotation around the object-centric “up” axis), given a reference image I_{ref} .

We perform tests across a variety of common 3D meshes. The reference images are generated using the differentiable path tracer MITSUBA 3 [Jakob et al., 2022b] in direct illumination mode. We verify that our lo-



cal shading model can generate images similar to the reference images (see inset). The results are generated by averaging over several random starting conditions for each object. This experiment is repeated for different camera image resolutions (128^2 , 512^2 , 1024^2).

We report quantitative results in Table 3.1 and qualitative results in Fig. 3.9. MITSUBA 3 recovers the rotation and translation most accurately, which *could* be attributed to its accurate gradient computation, using multiple sub-pixel samples. However, it should be expected that the results are biased towards the method used for generating the reference images. We show that this is the case by using our renderer as a reference for one experiment (see Table 3.1) and finding our method to yield the most accurate results in this case. Across all experiments, not using shadows at all and relying purely on the shading of the object surface performs worst, yet still manages to converge to reasonable solutions in some instances.

Unsurprisingly, rasterization is magnitudes faster than path tracing, both with and without shadows. Shadow mapping roughly doubles the runtime in rasterization, however, the pose estimation accuracy is significantly increased: our method consistently yields accuracy similar or close to path tracing. Modern GPUs are equipped with hardware units that significantly increase ray tracing performance. We include scores for a system with such a GPU (NVIDIA RTX 3080 Ti). It improves the path tracing runtime

Table 3.1: Quantitative results for the pose estimation experiments for different scenes and image resolutions. We measure the rotation error $\Delta\phi$, the translation error Δt , the total runtime, and runtime per iteration, averaged over ten runs. We include results for a GPU with ray-tracing cores (“RT”) and for a setting where we use our renderer as reference (“Our Ref.”). Best scores in **bold**, second best scores underlined.

	MITSUBA 3 [Jakob et al., 2022b]			Rasterizer + Shadows (Ours)			Rasterizer		
	$\downarrow \Delta\phi [^\circ]$	$\downarrow \Delta t$	$\downarrow \text{time [s] (/it)}$	$\downarrow \Delta\phi [^\circ]$	$\downarrow \Delta t$	$\downarrow \text{time [s] (/it)}$	$\downarrow \Delta\phi [^\circ]$	$\downarrow \Delta t$	$\downarrow \text{time [s] (/it)}$
128 × 128									
Bunny	0.31 0.05		54.13 (0.361)	<u>0.69</u>	<u>0.27</u>	<u>2.27 (0.015)</u>	3.04	5.56	1.06 (0.007)
Dragon	2.78 <u>4.01</u>		52.46 (0.349)	<u>3.32</u>	3.94	<u>2.09 (0.014)</u>	8.01	14.31	1.03 (0.007)
Hand	1.44 <u>5.24</u>		53.17 (0.354)	<u>3.66</u>	4.02	<u>2.08 (0.014)</u>	6.80	12.42	1.03 (0.007)
Spot	0.02 0.11		58.19 (0.388)	<u>7.40</u>	<u>12.83</u>	<u>2.11 (0.014)</u>	12.47	44.70	1.05 (0.007)
512 × 512									
Bunny	0.23 0.10		325.29 (2.168)	0.33	<u>0.22</u>	<u>3.76 (0.024)</u>	<u>0.31</u>	0.26	1.58 (0.009)
Dragon	2.86 3.91		306.46 (2.043)	<u>3.32</u>	<u>4.53</u>	<u>3.64 (0.023)</u>	8.02	11.63	1.54 (0.009)
Hand	1.41 5.18		317.40 (2.116)	<u>1.68</u>	<u>5.36</u>	<u>3.66 (0.023)</u>	1.85	5.66	1.51 (0.009)
Spot	0.03 0.02		381.80 (2.545)	<u>0.05</u>	<u>0.05</u>	<u>3.76 (0.023)</u>	5.80	23.15	1.77 (0.010)
Spot (RT)	0.03 0.02		53.26 (0.354)	<u>0.05</u>	<u>0.05</u>	<u>1.93 (0.012)</u>	6.92	21.52	1.05 (0.006)
Spot (Our Ref.)	<u>0.05</u> <u>0.12</u>		384.24 (2.561)	0.02 0.06		<u>3.60 (0.022)</u>	5.83	23.12	1.71 (0.010)
1024 × 1024									
Bunny	0.22 0.12		1245.60 (8.304)	<u>0.31</u>	<u>0.21</u>	<u>7.72 (0.047)</u>	0.33	0.25	3.71 (0.021)
Dragon	2.86 3.87		1151.74 (7.678)	<u>3.22</u>	<u>4.48</u>	<u>7.53 (0.046)</u>	7.85	11.48	3.18 (0.018)
Hand	1.42 5.38		1200.74 (8.005)	<u>1.61</u>	<u>5.43</u>	<u>7.72 (0.048)</u>	1.76	5.66	3.13 (0.018)
Spot	0.03 0.03		1441.56 (9.610)	0.03	<u>0.04</u>	<u>7.64 (0.046)</u>	<u>2.58</u>	3.93	3.19 (0.017)

but, at least in this example, rasterization is still 27× faster.

3.3.2 Face Reconstruction from Profile Shadows

While the variation within certain classes of objects is governed by pose, such as in articulated objects, deformations are best described by morphable models. A prominent example for morphable models is face geometry [Egger et al., 2020]. Our idea is to reconstruct face geometry from one (or more) shadows of the face. We base the experiment on the publicly available morphable model assembled by Li et al. [2020] (other models would work as well).

The first experiment uses synthetic reference shadows that are generated by rendering a known state of the morphable model. This allows comparing the optimization result not just based on images but also geometry (see Fig. 3.10). We find that using more shadows only slightly increases the runtime (the time is not linear in the number of light sources, because we have one base pass and multiple shadow passes) but significantly improves the accuracy (see Table 3.2).

We have also performed reconstructions of face geometry from “real” silhouettes, created manually by artist Charles Burns using scissors and black paper [Burns, 2011] (the process is shown here [Alive Network, 2016]). We ignore regions appearing to be

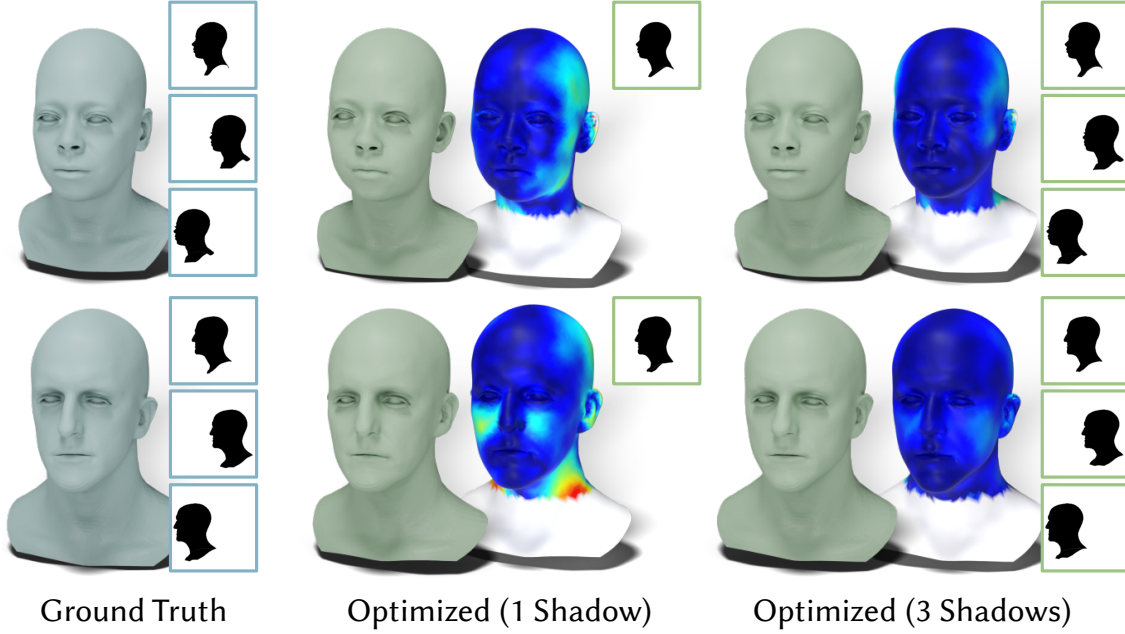


Figure 3.10: Face reconstruction from synthetic shadows.

Table 3.2: Quantitative results for face reconstruction from shadows, using artificial head geometry generated with a morphable face model. Best scores in **bold**.

Face ID	1 Shadow			3 Shadows		
	$\downarrow D_{\text{mean}}$	$\downarrow D_{\text{Hausdorff}}$	$\downarrow \text{time [s] (/it)}$	$\downarrow D_{\text{mean}}$	$\downarrow D_{\text{Hausdorff}}$	$\downarrow \text{time [s] (/it)}$
1	0.011	0.114	18.86 (0.047)	0.006	0.066	27.11 (0.068)
6	0.012	0.108	19.00 (0.048)	0.010	0.106	27.32 (0.068)
8	0.014	0.129	19.36 (0.048)	0.007	0.054	27.37 (0.068)
131	0.014	0.069	18.09 (0.045)	0.008	0.046	27.46 (0.069)
mean	0.013	0.105	18.83 (0.047)	0.008	0.068	27.32 (0.068)

(facial) hair or torso as those are not captured by the morphable model we use. While the lack of ground truth makes it difficult to quantify the resulting geometry, visually they appear to be reasonable approximations (Fig. 3.11).

3.3.3 Shadow Art

Shadow Art [Mitra and Pauly, 2009; Hsiao et al., 2018] is the design of a 3D object based only on the shadows it casts. When solving this design problem computationally, it is commonly simplified by assuming that the shadows are cast to planar surfaces orthogonal to directional lights, reducing the shadow images to silhouettes. In the traditional approach [Mitra and Pauly, 2009], the object is represented on a volumetric grid and voxel lines outside the shadow regions are removed – similar in spirit to visual hulls [Laurenzini, 1994] or space carving [Kutulakos and Seitz, 2000]. Sadekar et al. [2022] perform

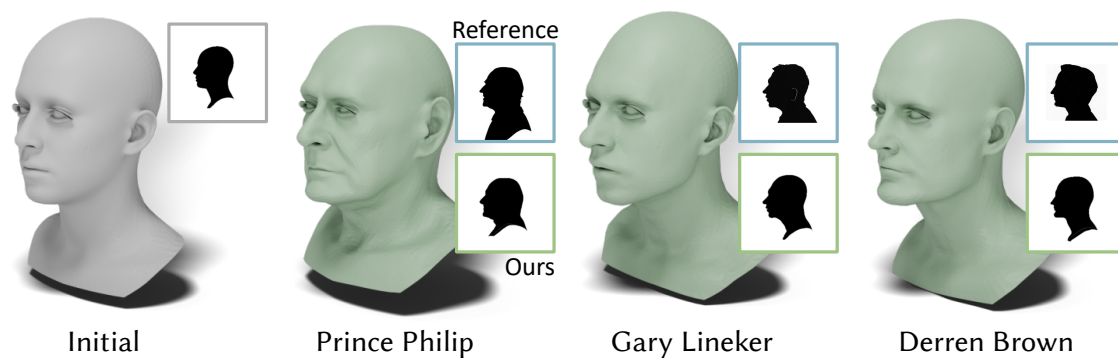


Figure 3.11: Face reconstruction from real silhouettes of public figures. We masked out hair, facial hair, and the base.

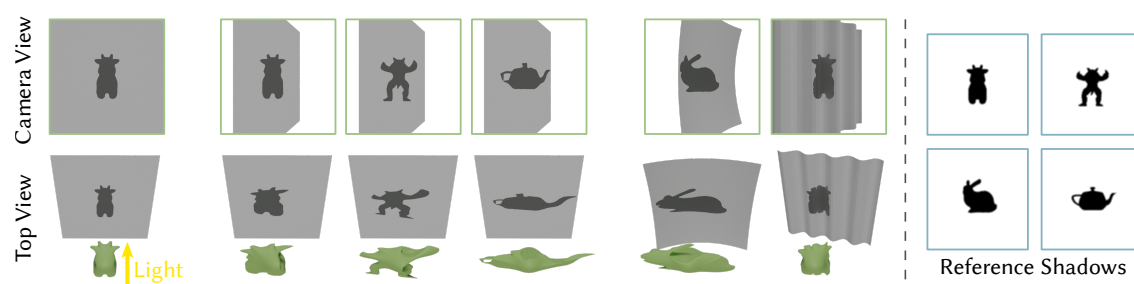


Figure 3.12: Shadow art with one view. Our method can be used for simple scenes with co-located light and camera (left), for more complex settings with perspective cameras that observe the shadow receiver from any direction (middle), and for complex receiver geometry (right).

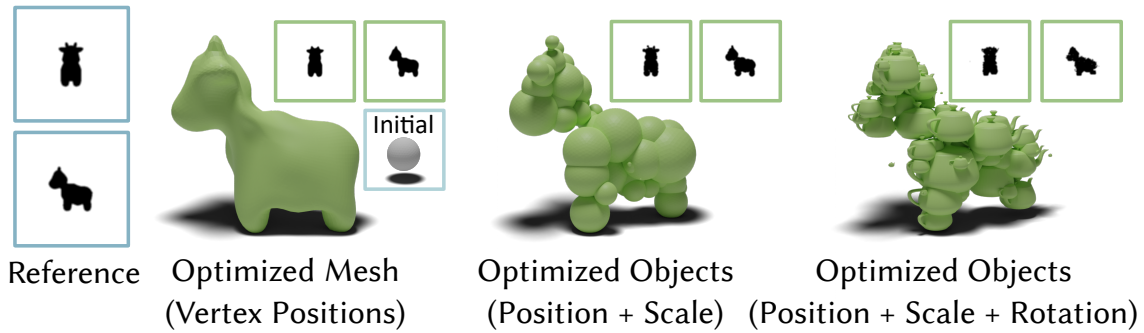


Figure 3.13: Shadow art with two views. Our method can be used in any setting compatible with differentiable rendering, for example optimizing vertex positions of a mesh (left) or optimizing the parameters of multiple geometric objects (middle and right).

Shadow Art with differentiable rendering of meshes, but the limitation of co-located cameras and lights remains.

In our framework, lights and cameras can be placed arbitrarily. Also, curved shadow receivers work out-of-the-box (Fig. 3.12). The only (minor) constraints we impose are gradient pre-conditioning [Nicolet et al., 2021] and a smoothness term between the normals of neighboring faces [Luan et al., 2021] (Chapter 7) to avoid arbitrary vertex movements during optimization.

A single triangle mesh is only one possible representation of the generated geometry. The combination of automatic differentiation and gradient descent allows optimizing arbitrary scene parameters. For example, we can also represent the object as a collection of shapes (see Fig. 3.13). Beyond our reference implementation based on triangle meshes, the same approach readily works with any geometry representation as long as it can be differentially rendered (e.g., signed distance functions [Vicini et al., 2022; Bangaru et al., 2022] or spheres [Lassner and Zollhöfer, 2021]).

3.3.4 Interactive Modeling from Shadows

Our rasterization-based implementation generates shadows and – most importantly – the gradients of the objective function in real time. We demonstrate this by turning the shadow art setting, which is usually concerned with static reference shadows, into a “shadow modeling” setting, where a user can interactively paint the target shadows and an appropriate deformation is applied to the initial geometry in real time (see Fig. 3.14). Note that using co-located camera and light is a design decision since painting silhouettes feels intuitive – the method is not limited to this setup.

3.4 Conclusion

In this chapter, we have shown that shadow mapping, one of the most common real-time algorithms for rendering shadows, can be made differentiable. Contrary to popular belief, we have demonstrated that the binary test in shadow mapping is not the critical factor, but rather visibility discontinuities at object silhouettes. Similar to the original

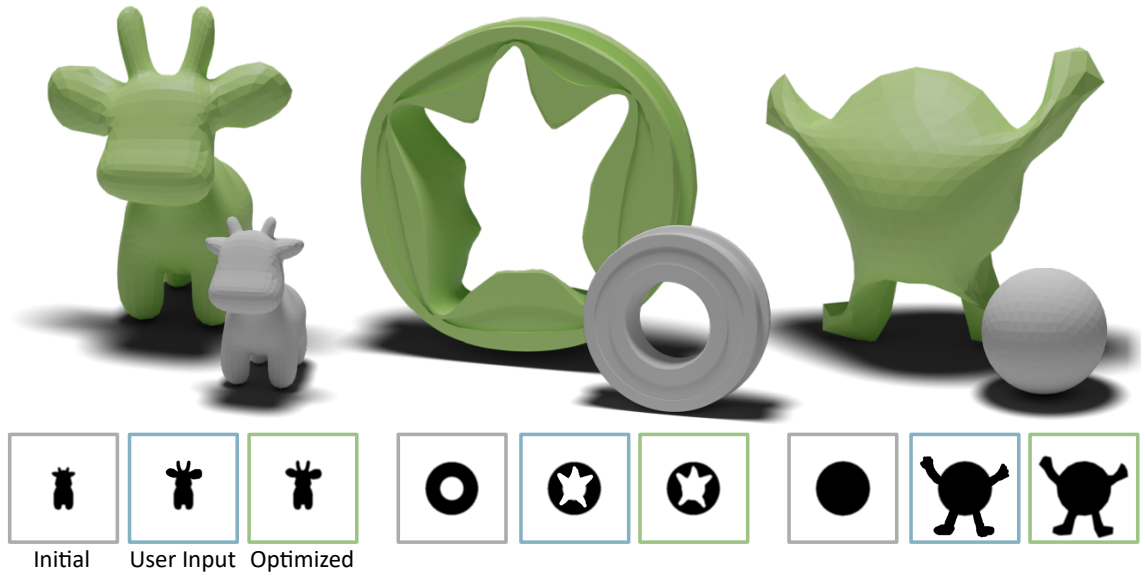
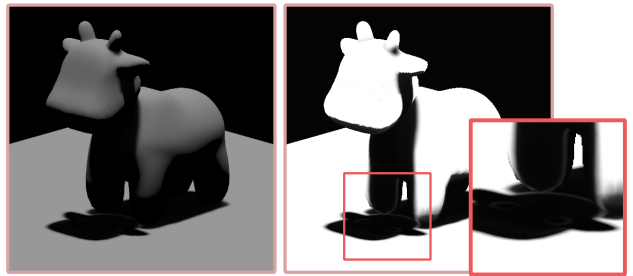


Figure 3.14: Shadow art from one view, with co-located camera and light is used as an interactive modeling tool. Users can modify the shadow of the reference object (gray) and an appropriate deformation is applied to the mesh in real-time (green).

shadow mapping idea, our solution uses existing rasterizers to render from the perspective of the light: rendering pre-filtered shadow maps with *differentiable* rasterizers results in shadow derivatives that properly track their variation with scene geometry.

Our method takes a step in closing the gap between efficient, rasterization-based differentiable renderers and physically-based differentiable renderers. While our approach is only approximate, the level of realism will suffice for various problems (e.g. shadows cast by the sun). We believe that having the ability to render and differentiate images with shadows magnitudes faster than Monte Carlo methods provides a new incentive for widespread use in applications, particularly ones based on machine learning.

Limitations So far, we have considered only point light sources instead of area lights, and focusing on direct shadows neglects other effects of global illumination. It is also well known that Variance Shadow Maps [Donnelly and Lauritzen, 2006] – our particular choice of pre-filtered shadow maps – suffer from *light leaking* in regions of high depth variance (see inset). While we did not observe an immediate impact in the experiments in this chapter, it might be desirable to reduce light leaking by using heuristics [Nguyen, 2007] or implementing other pre-filtered shadow maps [Peters and Klein, 2015; Annen et al., 2007, 2008] in our framework. In Chapter 4, we further investigate this direction and show that Moment Shadow Maps [Peters and Klein, 2015] can indeed mitigate adversarial effects of light leaking in optimizations. Independent of the illumination effects, we have so far only



considered triangle meshes for the representation of geometry.

Possible Generalizations In addition to shadows, various effects of global illumination have been recreated by efficient real-time approximations. Approaches very similar to ours may readily work with ambient occlusion [Miller, 1994], reflections and refractions [Heidrich and Seidel, 1999], or indirect illumination [McGuire et al., 2017].

Other geometry representations that are popular in the vision community, such as signed distance fields [Jiang et al., 2020] or NeRFs [Mildenhall et al., 2020], should work with shadow maps as well – although it is not clear if the same efficiency can be achieved, as the fast computation is mainly achieved by rasterizing triangle primitives.

CHAPTER 4

Differentiable Moment Bounds

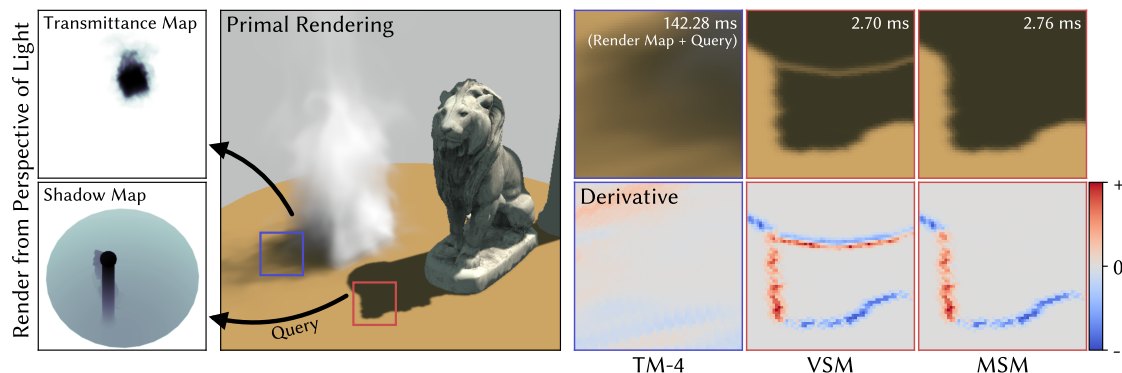


Figure 4.1: We prove that moment-based bounds are differentiable. This enables various applications of approximate methods from real-time graphics in differentiable rendering: we show differentiable transmittance mapping (“TM- n ”) akin to volumetric shadow mapping and differentiable shadow mapping for triangle meshes. The suggested moment-based techniques (e.g. **Moment Shadow Mapping**), are more principled, more accurate, and similarly efficient as existing approaches (e.g. **Variance Shadow Mapping**). The (forward) derivatives show the effect of moving the directional light.

In Chapter 3, we have shown that shadows can be efficiently generated and differentiated using the framework of pre-filtered shadow mapping. This also came with a realization: arbitrary real-time rendering algorithms do not produce useful derivatives in the general case, and even when they do, automatic differentiation is not guaranteed to give the right answer. This means that, even though forward rendering systems have recently been differentiated using differentiable shading languages [Bangaru et al., 2023] or with stochastic gradient estimates [Deliot et al., 2024], the burden of carefully examining the underlying theory for potential pitfalls (piecewise definitions, integrals with discontinuous integrand, etc.) remains. While it is easy to verify that many techniques are free of such issues (e.g., approximations for image-based lighting [Munkberg et al., 2022]), for other techniques this is less obvious. This naturally leads to the question: *which* rendering approximations *are* differentiable?

In this chapter, we seek an answer to the question for a family of approximations that build on the *theory of moments* to bound *measures*, which, in the simplest case, can be understood as non-negative, non-decreasing functions. Various problems in graphics can be cast in this framework: Donnelly and Lauritzen [2006] introduced moment bounds to graphics for shadow mapping, with extensions explored by Salvi [2008], and finally introduced in the general form by Peters and Klein [2015] with Moment Shadow Mapping.

Further, moment-based approximations are used for volume transmittance [Peters et al., 2016] and order-independent transparency [Münstermann et al., 2018; Sharpe, 2018].

The central algorithm underlying most of these techniques computes bounds from a finite set of power moments and is derived from the conditions in the classical literature [Akhiezer and Kreĭn, 1962; Akhiezer, 1965; Kreĭn and Nudel'man, 1977]. Due to its complexity in comparison to other rendering techniques, potential pitfalls for differentiation are less obvious. More so, because it is known that the representation has singularities – those could easily represent points of discontinuity. To our knowledge, we are the first to prove that the lower and upper bounds, as stated by Tari [2005] for the infinite case, are continuously differentiable functions in all parameters: many moment-based approximations can be readily combined with differentiable rendering. The proof generalizes the result from Chapter 3, where we have shown that the visibility function of variance shadow mapping is continuously differentiable (Section 3.1.3).

While the representation is differentiable, the aforementioned singularities have another adversarial effect in practice: in their vicinity, evaluating the bound becomes numerically unstable. Our study of differentiability reveals the behavior of the bound and its derivatives near singularities – we use these insights to derive an efficient and robust implementation of differentiable moment bounds.

Our results not only justify using moment-based approximations for differentiable rendering, but we also show that the new techniques improve over existing ones in inverse settings: For differentiable shadow mapping (Section 4.4.1), we use Moment Shadow Mapping [Peters and Klein, 2015] and show that it is an efficient and more accurate approach than Variance Shadow Mapping [Donnelly and Lauritzen, 2006], as used in Chapter 3. For differentiable transmittance in volume rendering (Section 4.4.2), we use classical work on moment-based transmittance estimation [Münstermann et al., 2018; Peters et al., 2016] and show that even the lowest quality approximation is as fast as existing ad-hoc approximations while being significantly more accurate.

4.1 Background: Bounding Measures with Moments

Given a non-negative function $f(z)$ on $\mathbb{R}$ with $\int_{-\infty}^{\infty} f(z) dz > 0$. We are interested in the integral

$$F(x) = \int_{-\infty}^x f(z) dz, \quad (4.1)$$

which is non-negative and non-decreasing. For example, we may encounter such integrals in volume rendering, as part of the transmittance, where f is the attenuation coefficient along a ray

$$T(x) = \exp(-F(x)) = \exp\left(-\int_0^x f(z) dz\right), \quad (4.2)$$

and the lower limit is adjusted to 0, assuming zero attenuation in the negative ray direction. Now, consider the following task: from f derive a compact representation, i.e., a small set of scalar values, from which the measure $F(x)$ can be approximated. A practical

motivation could be efficiency, if the transmittance is repeatedly evaluated at different points, given that the approximation can be evaluated faster than the integral.

This task may be approached using different classical techniques, including the representation of F or f in a (polynomial) function space. For many techniques, it is unclear how the essential properties of f (non-negative) or F (non-decreasing) could be preserved: transmittance ranges from 0 to 1, but approximations could yield values beyond 1 (approximation of F is negative) or *decrease* along the ray. This causes (visual) artifacts in the best case but generally leads to undefined behavior as assumptions about T do not hold.

A suitable tool for studying and approximating integrals of non-negative functions is the theory of moments. Consider a non-decreasing measure $\mu(x)$ ¹ and the Riemann-Stieltjes integral

$$m_i = \int_{-\infty}^{\infty} z^i d\mu(z), \quad 0 \leq i \leq n. \quad (4.3)$$

Characterizing, and in particular, recovering μ from the *power moments* m_i is known as the *moment problem*. Specifically, the instance considering a finite set of moments and the real line is known as the truncated Hamburger moment problem. The measure $\mu(x)$ corresponds to $F(x)$ from above but this form is more general, as it includes measures that do not admit a representation with a function f , such as the step function.

In general, a measure cannot be uniquely determined from a finite set of moments because different measures can generate the same moments. Yet, it is possible to generate lower and upper *bounds* for the set of measures that are representations² of the given moments:

$$\begin{aligned} L(x) &= \inf \left\{ \mu(x) \mid m_i = \int_{-\infty}^{\infty} z^i d\mu(z) \right\} && \text{(Lower bound)} \\ U(x) &= \sup \left\{ \mu(x) \mid m_i = \int_{-\infty}^{\infty} z^i d\mu(z) \right\}. && \text{(Upper bound)} \end{aligned} \quad (4.4)$$

Quite surprisingly, these bounds can be efficiently computed under mild assumptions on the moments [Tari et al., 2005], and in graphics, they have been used in efficient methods for shadow mapping [Peters and Klein, 2015; Peters et al., 2016] and order-independent transparency [Münstermann et al., 2018]. In this thesis, we only consider even orders $2n$, i.e., an odd number of moments $m_0, m_1, \dots, m_{2n}$ for $n > 0$.

4.1.1 A Geometric Perspective and General Observations

We think it is helpful to introduce an intuitive, geometric perspective on moment-based bounds. Consider the following curve in $\mathbb{R}^{2n+1}$

$$\mathbf{u}(t) = (1 \quad t \quad t^2 \quad \dots \quad t^{2n})^\top, \quad -\infty < t < \infty, \quad (4.5)$$

in the affine hyperplane $(1 \ 0 \ 0 \ \dots \ 0)^\top \mathbf{x} = 1$. Using $\mathbf{u}(t)$, the integral in Eq. (4.3) can be re-written in vector notation as

$$\mathbf{m} = \int_{-\infty}^{\infty} \mathbf{u}(z) d\mu(z), \quad (4.6)$$

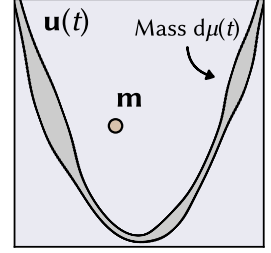
¹This definition of “measure” follows the conventions by Kreĭn and Nudel’man [1977].

²A measure μ is a “representation” of a moment vector $\mathbf{m}$ if it fulfills Eq. (4.6).

so the moment sequence is a point in $\mathbb{R}^{2n+1}$

$$\mathbf{m} = (m_0 \ m_1 \ \dots \ m_{2n})^\top. \quad (4.7)$$

If μ is viewed as a measure of physical mass distributed along the curve, normalized such that the total mass is $\int_{-\infty}^{\infty} d\mu(z) = m_0 = 1$, then the point $\mathbf{m}$ is simply the center of mass. More generally, the set of moments to all non-decreasing measures is the convex hull of the curve $\mathbf{u}(t)$ [Riesz, 1911]. The inset visualizes a mass distribution on the curve of order two $\mathbf{u}(t) = (1 \ t \ t^2)^\top$. If the requirement of normalized distributions is dropped, the set of moments is the *cone* formed by the rays from the origin through the convex hull of the curve – the *conic hull* of $\mathbf{u}$ [Kreĭn and Nudel'man, 1977, Theorem 3.4, p.15].



A central observation in the theory of moments is that integrals of polynomials are exactly expressible in the moments $\mathbf{m}$ for any measure μ that is a representation of them. Given a polynomial $p(x) = \sum_{i=0}^k a_i x^i$, with $k \leq 2n$, its integral is

$$\int_{-\infty}^{\infty} p(x) d\mu(x) = \int_{-\infty}^{\infty} \sum_{i=0}^k a_i x^i d\mu(x) = \sum_{i=0}^k a_i m_i. \quad (4.8)$$

For a moment sequence $\mathbf{m} = (m_0, \dots, m_{2n})$ and a vector of polynomial coefficients $\mathbf{a} = (a_0, \dots, a_{2n})$, $a_{j>k} = 0$ the operator

$$\mathfrak{S}_{\mathbf{m}}\{p\} = \mathbf{a}^\top \mathbf{m} \quad (4.9)$$

provides the exact integral of p for *any* μ being a representation of $\mathbf{m}$ [Kreĭn and Nudel'man, 1977, p.58], and leads to a key property:

Definition 1. A moment sequence $\mathbf{m} = (m_0, \dots, m_{2n})$ is called strictly positive if from $p(x) \geq 0$ ($p \not\equiv 0$) it follows that $\mathfrak{S}_{\mathbf{m}}\{p\} > 0$.

Strict positivity has an intuitive interpretation: the polynomial coefficients represent a vector $\mathbf{a} \in \mathbb{R}^{2n+1}$ and the condition $p(x) = \sum_{i=0}^k a_i x^i = \mathbf{a}^\top \mathbf{u}(x) \geq 0$ is fulfilled if the curve $\mathbf{u}$ lies in the closed upper half-space of the hyperplane $\mathbf{a}^\top \mathbf{x} = 0$. In other words, it is fulfilled if $\mathbf{a}$ defines a support hyperplane for the conic hull of the curve. The condition $\mathfrak{S}_{\mathbf{m}}\{p\} > 0$ is equivalent to $\mathbf{a}^\top \mathbf{m} > 0$, which demands that $\mathbf{m}$ does not lie on the support hyperplane: a strictly positive moment sequence lies *inside* the conic hull of $\mathbf{u}(t)$.

Strictly positive moments $\mathbf{m}$ are important because they can be represented as conical combinations of points on the moment curve:

Theorem 1 ([Akhiezer and Kreĭn, 1962, Theorem 3a, p.8]). *If $\mathbf{m}$ is strictly positive, there exist infinitely many canonical representations*

$$\mathbf{m} = \sum_{i=0}^n w_i \mathbf{u}(x_i),$$

with $w_i > 0$ the weights and $x_i \in (-\infty, \infty)$ the roots ($x_i \neq x_j$).

We will later show that it is natural to identify the x_i as the zeros of a polynomial, hence the name. The importance is that any point $\mathbf{m} \in \mathbb{R}^{2n+1}$ strictly inside the conic hull can be represented by the (strictly) conical combination of $n+1$ points on the curve. Note that $n+1$ is roughly *half* the embedding dimensions of the curve. Applied to the physical example above, the theorem states that there are infinitely many ways of concentrating mass in at most $n+1$ points on the curve, which result in the same center of mass. The infinite set is a one-parameter family:

Theorem 2 ([Akhiezer and Kreĭn, 1962, Theorem 3c, p.8]). *If $\mathbf{m}$ is strictly positive, for any value $\eta \in \mathbb{R} \setminus A$, there is a unique canonical representation of $\mathbf{m}$ where η is one of the $n+1$ roots.*

A is a small, finite set that depends on the moments and we delay the discussion to Section 4.2.1. Geometrically, given a point on the curve $\mathbf{u}(\eta)$, n unique points $\{\mathbf{u}(x_i)\}_1^n$ on the curve can be found so that their weighted combination yields $\mathbf{m}$.

The unique canonical representation is associated with a piecewise constant measure $\mu_\eta(x)$, increasing in the discrete set of the $n+1$ points, where $d\mu_\eta(x)$ consists of $n+1$ Dirac distributions placed at η and the $\{x_i\}$:

$$\int_{-\infty}^y \mathbf{u}(x) d\mu_\eta(x) = \int_{-\infty}^y \sum_{\substack{i=0 \\ x_i < y}}^n \delta(x - x_i) w_i \mathbf{u}(x) dx = \sum_{\substack{i=0 \\ x_i < y}}^n w_i \mathbf{u}(x_i). \quad (4.10)$$

The measure μ_η is important because it defines a bound on the set of all measures consistent with a given set of moments:

Theorem 3 ([Kreĭn and Nudel'man, 1977, Theorem 3.1, IV.§3, p.125], *Chebyshev-Markov inequalities*). *Let μ be any measure with the same moments as μ_η , then*

$$\begin{aligned} \int_{-\infty}^{\eta-0} d\mu(x) &\geq \int_{-\infty}^{\eta-0} d\mu_\eta(x) \\ \int_{-\infty}^{\eta+0} d\mu(x) &\leq \int_{-\infty}^{\eta+0} d\mu_\eta(x) \end{aligned}$$

Remark 1. *The notation follows Kreĭn and Nudel'man [1977, p.15]: the first inequality considers the mass in the interval $(-\infty, \eta)$ and the second inequality the mass in the interval $(-\infty, \eta]$.*

With the discrete representation from Eq. (4.10), this yields an explicit expression for the bounds [Tari, 2005, Corollary 2.7]:

$$\begin{aligned} \int_{-\infty}^{\eta-0} d\mu(x) &\geq \sum_{\substack{i=1 \\ x_i < \eta}}^n w_i = L(\eta) \\ \int_{-\infty}^{\eta+0} d\mu(x) &\leq w_0 + \sum_{\substack{i=1 \\ x_i < \eta}}^n w_i = U(\eta) \end{aligned} \quad (4.11)$$

Computing the bounds from Eq. (4.4) therefore boils down to computing the unique canonical representation consisting of the weights $\{w_i\}$ and the roots $\{x_i\}$ given η .

4.1.2 Computing the Unique Canonical Representation

If η is identified with x_0 , the unique canonical representation is the solution of a system of $2n + 1$ equations (c.f. Theorem 1)

$$m_k = \sum_{i=0}^n w_i x_i^k, \quad k = 0, \dots, 2n \quad (4.12)$$

in $2n+1$ unknowns: the $n+1$ weights and the n points $x_1, \dots, x_n$ (because $\eta = x_0$ is given). Maybe surprisingly, the computation of the points and weights can be separated, and it is possible to solve for the points x_i not knowing the weights w_i . Once all x_i are given, solving for the weights is just a linear system of equations. The equations in this section are provided without proof, and details can be found in the work by Tari [2005].

The main tool is the *Hankel matrix* of the moments

$$\mathbf{H} = \begin{bmatrix} m_0 & m_1 & \dots & m_n \\ m_1 & m_2 & & \vdots \\ \vdots & & \ddots & \\ m_n & \dots & & m_{2n} \end{bmatrix}, \quad (4.13)$$

which also indicates strict positivity of $\mathbf{m}$:

Theorem 4 ([Akhiezer and Kreĭn, 1962, Theorem 1, p.3]). *For a moment sequence to be strictly positive, it is necessary and sufficient that the Hankel matrix is positive definite.*

With the shorthand notation

$$\mathbf{x} \triangleq (1 \quad x \quad \dots \quad x^n)^\top, \quad (4.14)$$

the $\{x_i\}$ can be computed as solutions to

$$\mathbf{x}_i^\top \mathbf{H}^{-1} \mathbf{x}_0 = \mathbf{x}_i^\top \mathbf{H}^{-1} \boldsymbol{\eta} = 0, \quad i > 0. \quad (4.15)$$

This relates the unknown x_i to the known quantities $\boldsymbol{\eta}$ and $\mathbf{H}$: the points $\{x_i\}$ are the roots of the so-called *kernel polynomial*

$$K(x) = \mathbf{x}^\top \mathbf{H}^{-1} \boldsymbol{\eta} = \sum_{k=0}^n x^k (\mathbf{H}^{-1} \boldsymbol{\eta})_k. \quad (4.16)$$

Therefore, finding the n points x_i of the unique canonical representation given $x_0 = \eta$ amounts to finding the roots of the n -degree kernel polynomial. If $\mathbf{m}$ is strictly positive and $\eta \notin A$, the roots are real and simple, i.e., K has a set of n distinct roots (Corollary 4).

With all points x_i , the weights can be either computed by solving the (reduced) Vandermonde system in Eq. (4.12) or individually as

$$w_i = \frac{1}{\mathbf{x}_i^\top \mathbf{H}^{-1} \mathbf{x}_i}. \quad (4.17)$$

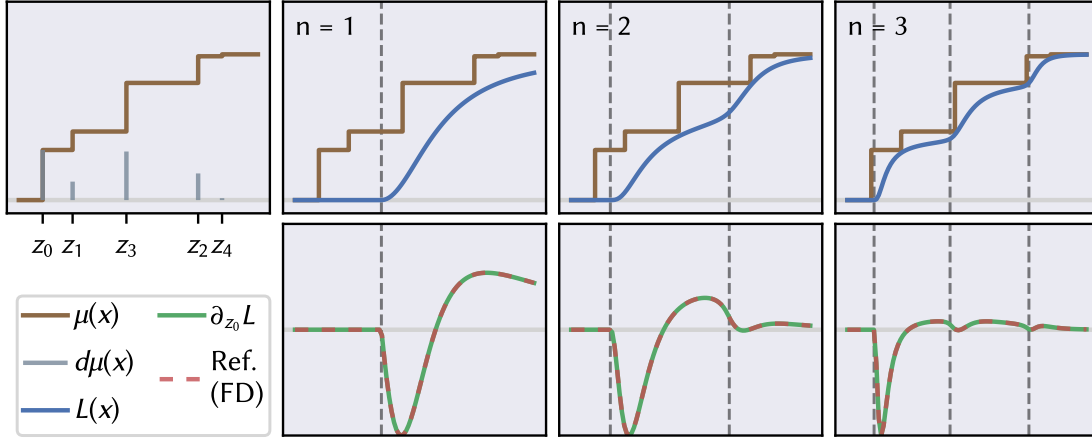


Figure 4.2: A discrete measure μ with five points of increase $z_0, \dots, z_4 \in \mathbb{R}$. Top row: the lower bound L for different degrees, corresponding to 3, 5, and 7 moments, respectively. Bottom row: the derivative of the lower bound for the point z_0 , which affects the moments. The bound and its derivatives are continuous. The dashed vertical lines mark singularity points of the bound.

4.2 Differentiating Moment-based Bounds

The approach detailed in Section 4.1 gives rise to two functions that, depending on an evaluation point $\eta \in \mathbb{R}$ and a sequence of moments $\mathbf{m} \in \mathbb{R}^{2n+1}$, compute an upper and a lower bound on all measures μ that are representations of the moments. When viewed as a function in all parameters, the lower bound takes the form (c.f. Eq. (4.11))

$$L(\eta; \mathbf{m}) = \sum_{\substack{i=1 \\ x_i(\eta; \mathbf{m}) < \eta}}^n w_i(\eta; \mathbf{m}), \quad (4.18)$$

where the w_i are the weights and the x_i are the points of the unique canonical representation. The upper bound includes the weight of η

$$U(\eta; \mathbf{m}) = w_0(\eta; \mathbf{m}) + L(\eta; \mathbf{m}) \quad (4.19)$$

If one intends to use the lower or upper bounds in a differentiable setting, one may ask if these functions are differentiable in $\mathbf{m}$ and η . It turns out that the answer to this question is difficult to find in the given form, in particular, because the *number of terms* in the sum in L varies depending on *both*, $\mathbf{m}$ and η .

In Chapter 3, we have investigated the visibility function of Variance Shadow Mapping [Donnelly and Lauritzen, 2006] and shown that it is continuously differentiable. It turns out that this function, based on Chebyshev's inequality, is a special case of the lower bound for probability measures and $\mathbf{m} = (1, m_1, m_2)$. We now generalize this result to sequences with an arbitrary number of moments (see the example in Fig. 4.2).

The proof can be broken down into the following high-level parts:

1. Section 4.2.1: we partition the space $\mathbb{R} \times \mathbb{R}^{2n+1}$ into regions where L is defined and regions where it is not, and show that L is trivially differentiable in the regions forming its domain. A key observation is that the number of terms in the bound, i.e., the number of summed weights (c.f. Eq. (4.18)), remains constant inside the valid regions.
2. Section 4.2.2: we then analyze the behavior of the bound as one approaches the boundary between two regions. The main insight is that at each transition between distinct regions only *one* weight is included or excluded from the sum in L .
3. Section 4.2.3: finally, we show that not only the weight but also its derivative with respect to $\mathbf{m}$ and η vanish: the limit exists, so the singularities at region boundaries are removable, which makes L (and U) continuously differentiable in their domain and across the singular points.

Since the position of η among the other roots is relevant in the following, we assume, for this section, that all roots, including η , are ordered from lowest to highest, i.e., $x_0 < \dots < x_j (= \eta) < \dots < x_n$.

4.2.1 Partitioning the Domain

The main difficulty with the bound L and its derivative is the varying number of summed weights (c.f. Eq. (4.18)): including a weight is a *binary* decision that can lead to discontinuities in the function or its derivative. The central idea is to first dissect the space $\mathbb{R} \times \mathbb{R}^{2n+1}$ into regions with a constant number of terms. Inside of these regions, differentiability can be easily investigated in terms of the w_i .

Regions with Constant Number of Terms

According to Theorem 1, canonical representations, in particular the unique one that defines the bound L , only exist for strictly positive moments $\mathbf{m}$. Moreover, Theorem 2 indicates that there is the additional restriction of $\eta \notin A$: A contains the n simple roots of a special degree- n polynomial $P_n(x; \mathbf{m})$. This characterizes the *domain* of L :

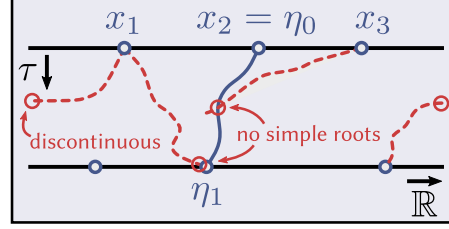
$$\Omega = \{(\eta, \mathbf{m}) \in \mathbb{R} \times \mathbb{R}^{2n+1} : \text{positive}(\mathbf{m}) \wedge P_n(\eta; \mathbf{m}) \neq 0\}. \quad (4.20)$$

At any given point $(\eta_0, \mathbf{m}_0) \in \Omega$ the bound L will be a sum over some number of terms, so, under which conditions does the number change if one moves to *another* point $(\eta_1, \mathbf{m}_1) \in \Omega$, on a continuously differentiable path? The answer is our first result:

Theorem 5. *If the number of terms in the bounding functions differs between two points $(\eta_0, \mathbf{m}_0) \in \Omega$ and $(\eta_1, \mathbf{m}_1) \in \Omega$, any continuously differentiable path connecting the points crosses at least one point $(\eta', \mathbf{m}') \in \mathbb{R} \times \mathbb{R}^{2n+1}$ where either η' is a root of $P_n(x; \mathbf{m}')$ or the sequence of moments $\mathbf{m}'$ is not strictly positive.*

The proof can be found in Appendix A.3, Section A.3.3. It exploits that the number of terms changes if and only if one of the $\{x_i\}$ changes its place with η on the real line. The points $\{x_i\}$ and η are the roots of an orthogonal polynomial $\psi_{n+1}(x; \eta, \mathbf{m})$,

which has $n + 1$ real and simple roots in the domain Ω . Since the roots are continuously differentiable functions in $\mathbf{m}$ and η , the $\{x_i\}$ and η can only change their order outside of the domain Ω . The inset visualizes the theorem and shows the evolution of the roots on a continuously differentiable path $\gamma(\tau) \in \mathbb{R} \times \mathbb{R}^{2n+1}$ from a point $(\eta_0, \mathbf{m}_0) \in \Omega$ to a point $(\eta_1, \mathbf{m}_1) \in \Omega$. Red trajectories cannot be taken in Ω .



Geometrically, strict positivity can be violated on a path when moments $\mathbf{m}$ cross through $\mathbf{u}(\eta)$ on the boundary of the moment curve (more generally boundary regions on the cone) or due to numerical inaccuracy. But, strict positivity can be enforced by *biasing* [Peters and Klein, 2015]: the moments $\mathbf{m}$ are slightly “pulled” inside of the conic hull, away from its boundary.

Proposition 1. *If $\alpha \in (0, 1]$ and $\mathbf{m}^\star \in \mathbb{R}^{2n+1}$ is a bias vector such that $(1 - \alpha)\mathbf{m} + \alpha\mathbf{m}^\star$ is strictly positive, the number of terms in the bounding functions changes only in points where η is a root of the polynomial $P_n(x; (1 - \alpha)\mathbf{m} + \alpha\mathbf{m}^\star)$.*

Therefore, one can see the domain Ω as a partition of regions where η is not a root of P_n , and inside of each region, the number of terms in L remains constant.

Differentiable Weights

Each region of the domain Ω has a fixed number of terms, so, inside of it, the derivative of L simply reduces to the sum of derivatives of the weights w_i , and we observe:

Theorem 6. *For $(\eta, \mathbf{m}) \in \Omega$ the weights $w_0, \dots, w_n$ associated with all points of the unique canonical representation are continuously differentiable in $\mathbf{m}$ and η .*

See Appendix A.3, Section A.3.4 for the (short) proof; it follows:

Corollary 1. *The bounding functions $L(\eta, \mathbf{m})$ and $U(\eta, \mathbf{m})$ are continuously differentiable in their domain Ω .*

4.2.2 Approaching Singularities

Although having established differentiability on the domain of L in Section 4.2.1, the result does not rule out discontinuities at points in

$$\Lambda = \{(\eta, \mathbf{m}) \in \mathbb{R} \times \mathbb{R}^{2n+1} : \text{positive}(\mathbf{m}) \wedge P_n(\eta; \mathbf{m}) = 0\}, \quad (4.21)$$

which is the set of points with strictly positive $\mathbf{m}$ but η is a root of P_n . Recall that the bounds are undefined at these points. In the following, we investigate the behavior of the $\{x_i\}$ as a point $(\eta, \mathbf{m}) \in \Omega$ in the domain of L approaches such *singularities*.

The polynomial $P_n(x, \mathbf{m})$ is part of a sequence of polynomials $\{P_i\}$, which are pairwise orthogonal under measures that are representations of $\mathbf{m}$. As an orthogonal polynomial, P_n has n real and simple roots $\{y_i\}_1^n$: at a point $(\eta, \mathbf{m}) \in \Omega$, η must lay between at most two roots of P_n , in either of the intervals $(-\infty, y_1)$, (y_j, y_{j+1}) , or (y_n, ∞) .

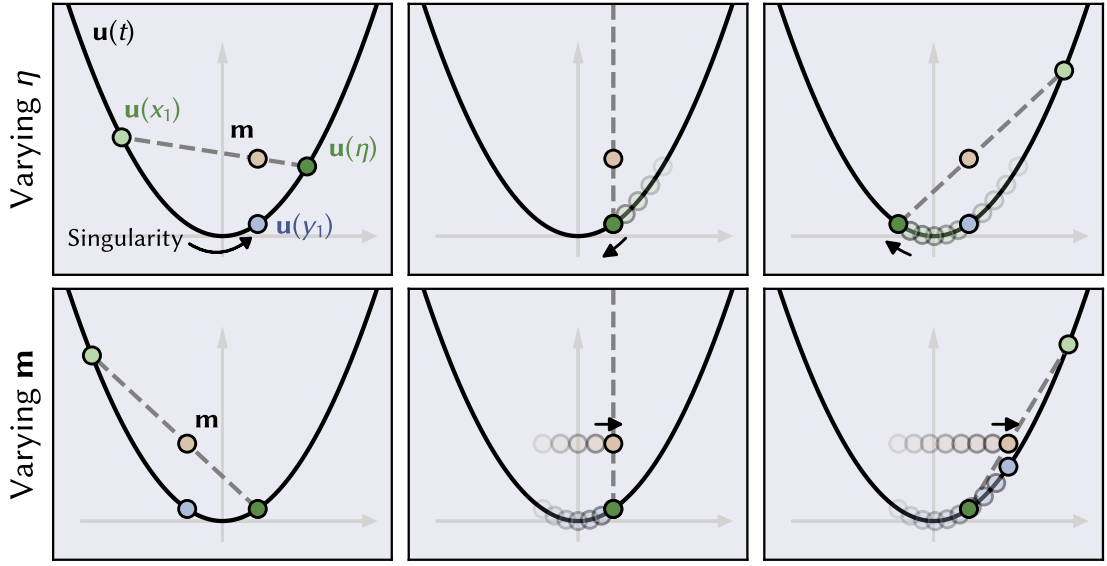
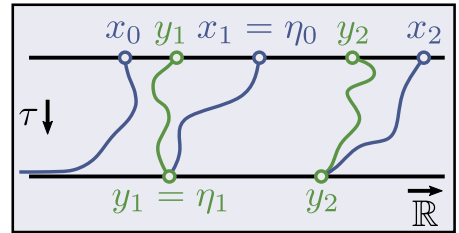


Figure 4.3: Geometric relationship between the points of the canonical representation η and x_1 for a moment sequence $\mathbf{m} = (1, m_1, m_2)$ and the singularity point y_1 , a root of P_n . The first dimension is omitted. First row: η approaches y_i . Second row: y_i approaches η as the moments are varied.

Geometrically, one can think of the roots as n distinct points $\mathbf{u}(y_i)$ on the moment curve, with $\mathbf{u}(\eta)$ between (at most) two of them. The situation $P_n(\eta; \mathbf{m}) = 0$ can now arise in different ways: (1) the point $\mathbf{u}(\eta)$ moves towards one of the two roots on the curve, (2) one of the two points moves towards $\mathbf{u}(\eta)$ on the curve, or (3) a combination of the two. Fig. 4.3 visualizes the behavior of the roots:

Theorem 7. *If $x_j(= \eta) < x_n$ approaches y_{j+1} on a continuously differentiable path, then $x_n \rightarrow \infty$. If $x_j(= \eta) > x_0$ approaches y_j on a continuously differentiable path, then $x_0 \rightarrow -\infty$.*

The proof is found in Appendix A.3, Section A.3.5. It exploits that the points $\{x_i\}_0^n$ (including η) strictly *interlace* with the roots $\{y_i\}_1^n$ of P_n in the domain Ω . At singularities, n of the points $\{x_i\}$ collapse to the n roots $\{y_i\}$ of P_n (see inset). As the $\{x_i\}$ and the roots $\{y_i\}$ are continuously differentiable functions in $(\eta, \mathbf{m})$, a path $\gamma(\tau)$ from a point $(\eta_0, \mathbf{m}_0) \in \Omega$ to a singularity point $(\eta_1, \mathbf{m}_1) \in \Lambda$ is reversible, so each $\{y_i\}$ at the singularity corresponds exactly to one $\{x_i\}$. This leaves one of the outer points x_0 or x_n ($\neq \eta$) without a corresponding root at the singularity, and it must go to $\pm\infty$. Since the theorem considers both directions, we can state the following:



Corollary 2. *If $x_j(= \eta) > y_j$ follows a continuously differentiable path through y_j such that after the singularity $x_{j-1}(= \eta) < y_j$, then the point x_0 goes to $-\infty$ and reappears as x_n from ∞ . The other direction of x_j approaching y_{j+1} behaves analogously.*

Therefore, when crossing a singularity, exactly *one* root x_i changes its place with η , so for the sum in L (c.f. Eq. (4.18)) the following holds:

Corollary 3. *As η follows a continuously differentiable path through a singularity, only the weight associated with x_0 or x_n is included in or excluded from the sum in the bounding functions.*

The last observation is essential as it allows studying only *one* of the weights and its derivatives to determine the limit behavior for the bounding functions L and U .

4.2.3 Weight (Derivative) at Singularities

In Section 4.2.1 we have shown that L is continuously differentiable on its domain Ω but the domain is a collection of distinct regions, connected by singularities at which L is undefined, not ruling out discontinuous behavior across singularities. We have then observed in Section 4.2.2 that when singularities are crossed, exactly one of the points $x_k \in \{x_i\}$ switches places with η , and therefore only its weight w_k is included or excluded from the sum in L (recall Eq. (4.18)). It remains to investigate how the weight and its derivative, and therefore L and its derivative, behave when a singularity is approached, which is equivalent to studying their limit as $x_k \rightarrow \pm\infty$ (c.f. Theorem 7).

The possible outcomes are apparent: if the weight w_k , in the limit, goes to zero, the singularities in L are removable. If additionally the derivatives $\frac{\partial w_k}{\partial \eta}$ and $\frac{\partial w_k}{\partial \mathbf{m}}$ go to zero, the singularities in the derivatives $\frac{\partial L}{\partial \eta}$ and $\frac{\partial L}{\partial \mathbf{m}}$ are removable. Otherwise, L or its derivatives will be discontinuous. We make the following two observations (proven in Appendix A.3, Sections A.3.6 and A.3.7):

Lemma 1. *If x_k is the point that goes to $\pm\infty$, then the associated weight w_k goes to zero.*

Lemma 2. *If x_k is the point that goes to $\pm\infty$, then the associated derivatives of the weight $\frac{\partial w_k}{\partial \eta}$ and $\frac{\partial w_k}{\partial \mathbf{m}}$ go to zero.*

The proofs use the fact that both the weight and its derivatives are rational functions in x_k , where the denominator has a (much) higher degree than the numerator, so they vanish as $x_k \rightarrow \pm\infty$. We conclude this section with our main result:

Theorem 8. *The bounding functions $L(\eta; \mathbf{m})$ and $U(\eta; \mathbf{m})$ are continuously differentiable on their domains Ω and the singularities are removable so that L and U are continuously differentiable over the set $\Omega \cup \Lambda \subset \mathbb{R} \times \mathbb{R}^{2n+1}$, with all strictly positive moments.*

4.3 Implementation

Algorithm 3 computes the moment bounds (the *primal* algorithm) and loosely follows Münstermann et al. [2018, Algorithm 1], where β blends between the lower and upper bound. We have not yet materialized all operations (e.g., root finding) because, as will be shown, the choice significantly affects the stability of the primal algorithm and its derivative, which is calculated by the *adjoint* algorithm.

Algorithm 3 Computing a moment-based bound (primal algorithm)

Inputs: moments $\mathbf{m} = (m_0, \dots, m_{2n})$, evaluation point η , overestimation weight $\beta \in [0, 1]$, bias $\alpha \in [0, 1]$

Output: bound $b = (1 - \beta)L + \beta U$

```

 $\mathbf{m} \leftarrow (1 - \alpha)\mathbf{m} + \alpha\mathbf{m}^\star$  ▷ Biasing (Prop. 1)
 $\mathbf{H} \leftarrow \text{hankel\_matrix}(\mathbf{m})$  ▷ Eq. (4.13)
 $\mathbf{c} \leftarrow \text{cholesky\_solve}(\mathbf{H}, (1, \eta, \eta^2, \dots, \eta^n)^\top)$  ▷ Eq. (4.16)
 $x_1, \dots, x_n \leftarrow \text{find\_polynomial\_roots}(\mathbf{c})$  ▷ Eq. (4.15)
 $\bar{\mathbf{m}} \leftarrow (m_0, \dots, m_n)$ 
 $w_0, \dots, w_n \leftarrow \text{vandermonde\_solve}(\eta, x_1, \dots, x_n, \bar{\mathbf{m}})$  ▷ Eq. (4.12)
 $b \leftarrow \beta w_0$ 
for  $i = 1, \dots, n$  do
  if  $x_i < \eta$  then
     $b \leftarrow b + w_i$ 
return  $b$ 

```

Automatic Differentiation Algorithm 3 can be readily implemented in an automatic differentiation (AD) framework like PyTorch [Paszke et al., 2019]: a Cholesky solver is available, the roots can be determined in closed-form or, for high degrees, as the eigenvalues of the companion matrix (as in numpy [Harris et al., 2020]), and a generic linear solver can be used for the Vandermonde system.

While this straightforward implementation is a useful reference, it is numerically unstable and inefficient, both in terms of memory consumption and runtime. Improving the stability requires careful consideration of the numerically critical parts (Section 4.3.1). The derived primal and adjoint algorithms can be efficiently implemented by fusing all operations, avoiding the construction of a computation graph as in automatic differentiation (Section 4.3.2).

4.3.1 Explicit Algorithm and Reverse-Mode Differentiation

In this section, we will gradually materialize Algorithm 3, one operation at a time, and derive expressions for reverse-mode derivatives. This leads to concrete primal and adjoint algorithms. We will use the notation δa to denote the adjoint of a variable a , i.e., the derivative $\frac{\partial \mathcal{L}}{\partial a}$ w.r.t. a scalar function $\mathcal{L}(a)$, which could be the loss function in a gradient-based optimization. We assume adjoints with the same “shape” as the primal variable, i.e., if $\mathbf{b} \in \mathbb{R}^m$ then $\delta \mathbf{b} \in \mathbb{R}^m$.

Solving for the Kernel Polynomial Since the Hankel matrix $\mathbf{H}$ of the biased moment vector is positive-definite and symmetric,

$$\mathbf{H}\mathbf{c} = \boldsymbol{\eta} \tag{4.22}$$

(c.f. Eq. (4.16)) can be solved stable by performing the Cholesky decomposition of $\mathbf{H}$, followed by forward and backward substitution.

The output of this operation is the vector $\mathbf{c}$, so, in reverse-mode differentiation, the adjoint operation uses the input $\delta \mathbf{c}$ to compute $\delta \mathbf{H}$ and $\delta \boldsymbol{\eta}$. For a linear system $\mathbf{A}\mathbf{x} =$

$\mathbf{b}$ with invertible $\mathbf{A}$, one can derive the equalities defining the adjoints [Strang, 2007, Section 8.7]

$$\begin{aligned}\delta\mathbf{b} &= \mathbf{A}^{-\top} \delta\mathbf{x} \\ \delta\mathbf{A} &= -\delta\mathbf{b} \mathbf{x}^\top,\end{aligned}\tag{4.23}$$

which lead to explicit expressions for the outputs

$$\begin{aligned}\delta\boldsymbol{\eta} &= \mathbf{H}^{-1} \delta\mathbf{c} \\ \delta\mathbf{H} &= -\delta\boldsymbol{\eta} \mathbf{c}^\top,\end{aligned}\tag{4.24}$$

since $\mathbf{H}$ is symmetric. Therefore, $\delta\boldsymbol{\eta}$ is obtained by solving another linear system involving the Hankel matrix, with the same Cholesky decomposition, and the adjoint $\delta\mathbf{H}$ is simply an outer product.

Finding the Kernel Roots By the structure of the problem, and critical for our proofs in Section 4.2, we know that one of the roots $\{x_i\}_1^n$ of the kernel polynomial is unbounded when the query point η approaches a singularity, as its leading coefficient c_n vanishes. Therefore, besides efficiency, numerical stability and adequate precision in 32-bit arithmetic are desired properties.

The cases $n = 1$ (3 moments) and $n = 2$ (5 moments) lead to kernel polynomials of degree $d = 1$ and $d = 2$, respectively, whose roots can be found using closed-form expressions. For the quadratic equation, we have implemented a robust solver following Kahan [2004]. Although closed-form expressions exist for degrees 3 and 4, we opt for a fast and more accurate Newton-style approach for degrees > 2 [Yuksel, 2022]. Notably, vanishing c_n is naturally handled in our implementation by returning one root with value `inf` instead of branching to a different version of the algorithm or trying to explicitly avoid this case by comparing η to the singularity points (the roots of P_n , Section 4.2.1). All subsequent operations gracefully handle roots at infinity by returning the limit value.

If $\{x_i\}_{i=1}^n$ are the roots of the kernel polynomial $K(\mathbf{c}, x)$ with coefficients $\mathbf{c}$, the adjoint operation computes $\delta\mathbf{c}$, given $\{\delta x_i\}$:

$$\delta\mathbf{c} = \sum_{i=1}^n \frac{\partial x_i}{\partial \mathbf{c}} \delta x_i.\tag{4.25}$$

The implicit function theorem relates the roots to the coefficients

$$\frac{\partial x_i}{\partial \mathbf{c}} = - \frac{\begin{pmatrix} 1 & x_i & \dots & x_i^n \end{pmatrix}^\top}{\frac{\partial K}{\partial x}(\mathbf{c}, x_i)}\tag{4.26}$$

This expression is undefined for a root at infinity but our proofs in Section A.3.7 show that the product

$$\underbrace{\frac{\partial x_i}{\partial \mathbf{c}} \frac{\partial w_i}{\partial x_i} \delta w_i}_{:= \delta x_i}\tag{4.27}$$

goes to zero when approaching singularities. Since the result becomes less reliable with large x_i , we test if the quadratic factor in the denominator of the derivatives $\frac{\partial w_i}{\partial \eta}$ (Eq. (A.16)) and $\frac{\partial w_i}{\partial \mathbf{m}}$ (Eq. (A.21)) overflows and if so, set the contribution of x_i to $\delta\mathbf{c}$ to zero.

Solving the Vandermonde System Instead of solving the Vandermonde system with a generic approach, we use the iterative algorithm by Björck and Pereyra [1970] as we found it to be efficient, stable, and simple to implement. The algorithm is derived from interpolation with divided differences, which is also the suggested method by Münstermann et al. [2018]. Additionally, we observe that it gracefully handles inputs close to or exactly at infinity: if a point x_i is at infinity with value `inf`, the associated solution w_i becomes zero. This corresponds exactly to the expected limit behavior (Section 4.2.3). The stability of the algorithm generally depends on the order of the inputs $\{\eta, x_1, \dots, x_n\}$ [Higham, 1987]. We found that ordering the roots ascending by absolute value drastically improves the precision (this is effectively pivoting), and only then the limit value is obtained.

The inputs to the algorithm are the roots $x_0(= \eta), x_1, \dots, x_n$ and the first $n+1$ entries of the moment vector $\mathbf{m}$, which we denote by $\bar{\mathbf{m}} = (m_0, \dots, m_n)^\top$. The adjoint of the partial moment vector is obtained by re-using the equalities for the linear system (Eq. (4.23)):

$$\delta \bar{\mathbf{m}} = \mathbf{V}^{-\top} \delta \mathbf{w}, \quad (4.28)$$

where $\mathbf{w} = (w_0, w_1, \dots, w_n)^\top$ is the vector of weights and its adjoint $\delta \mathbf{w}$ is an input. We obtain $\delta \bar{\mathbf{m}}$ with an algorithm for solving the *dual* Vandermonde system by Björck and Pereyra [1970]. Intuitively, the solution is a polynomial interpolating the values $\{\delta w_i\}$ at points $\{x_i\}$. If a root x_i is at infinity, its value δw_i does not contribute to the interpolation (this can be seen by expressing the derivative in Eq. (A.18) with the Vandermonde matrix), so the resulting polynomial has one less degree, i.e., $\delta \bar{\mathbf{m}}_n = 0$. This is exactly the output of the algorithm for an x_i at infinity.

The same equalities provide an expression for the adjoint matrix

$$\delta \mathbf{V} = -\delta \bar{\mathbf{m}} \mathbf{w}^\top, \quad (4.29)$$

which leads to the root adjoints

$$\delta x_i = \sum_{j=0}^n \sum_{k=0}^n \delta \mathbf{V}_{j,k} \left[\frac{\partial \mathbf{V}}{\partial x_i} \right]_{j,k} = -\frac{\partial}{\partial x_i} \left(\sum_{j=0}^n \delta \bar{\mathbf{m}}_j w_j x_i^j \right), \quad (4.30)$$

This is simply the derivative of a polynomial with coefficients $\delta \bar{\mathbf{m}} w_i$ evaluated at x_i . For x_i at infinity, it may be undefined in floating point arithmetic and this case is explicitly handled in the adjoint root finding (see discussion around Eq. (4.27)).

Remaining Operations The moment biasing, Hankel matrix construction, and weight summation require no special considerations.

4.3.2 Fused Implementation

Implementing the AD version or the explicit algorithms (Section 4.3.1) in a framework like PyTorch can be disappointing performance-wise: each operation invokes a dedicated kernel, with the launch overhead quickly accumulating for complex computation graphs, and more importantly, all intermediate results are materialized in memory, which not

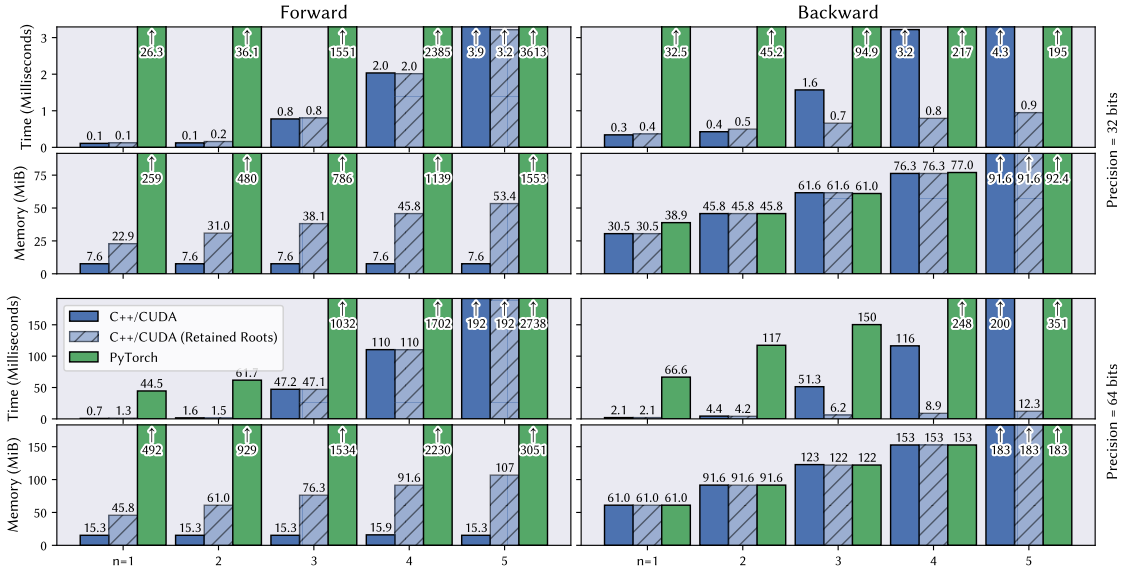


Figure 4.4: Runtime and memory requirements of forward and backward implementations of moment-based bounds. The benchmark uses 2 million randomly sampled points ($\eta, \mathbf{m}$). All measurements were performed on a workstation with an Intel i7-13700 CPU and an NVIDIA RTX 3090 GPU (24 GB VRAM). The PyTorch implementation is based on automatic differentiation applied to Algorithm 3 (Section 4.3, Paragraph *Automatic Differentiation*), while the C++/CUDA implementation follows Section 4.3.1. The *Retained Roots* mode stores the roots of the kernel polynomial between the forward and backward pass.

only leads to excessive storage requirements but also results in expensive reads from (GPU) memory. Just-in-time (JIT) compilation can mitigate these issues but comes at the cost of losing control over the generated kernels and might not lead to the expected improvements: for example, we have JIT-compiled the AD implementation (Section 4.3, *Automatic Differentiation*) using PyTorch 2.x’s `compile` feature, measuring either negligible improvements ($n = 1, 2$) or even worse runtime ($n > 2$), because some operations are currently unsupported (e.g. finding eigenvalues of a matrix).

Instead, we have implemented the primal and adjoint algorithms from Section 4.3.1 entirely in C++, only depending on a small subset of the standard library. The floating point type and n are known at compile time, leading to the generation of highly optimized code. We instantiate and test variants for 32-bit float and 64-bit double, and $n = 1, \dots, 5$. GPUs are targeted using CUDA, such that all operations of the primal algorithm are compiled into a single CUDA kernel, and similarly, a single kernel is compiled for the adjoint algorithm. Global memory accesses are often the bottleneck of a GPU implementation, so we ensure that intermediate values can reside in registers. Specifically, we avoid register spilling to (local) memory by only statically indexing into arrays, for example during the root finding procedure [Peters, 2023].

The C++/CUDA implementation is exposed to Python using nanobind [Jakob, 2022], so it readily works with array programming frameworks like PyTorch, numpy, or JAX.

Accuracy Verification We first verify the accuracy of the C++ implementation by comparing it to the PyTorch (AD) implementation for 1 million random samples. The average accuracy in the computed roots and weights is comparable (Table A.1, Appendix A.2). The gradients of the bound computed by the adjoint procedure are verified by comparing them to finite differences. The C++ implementation produces errors almost equal to the PyTorch AD implementation, differing in the 3rd to 4th digit (Table A.1). We exclude samples with η near a singularity, so this merely serves as a sanity check and the limit behavior is studied separately (Paragraph *Stability Analysis*)

Performance We compare the runtime and memory consumption of the C++/CUDA implementation to the PyTorch-based AD implementation, on the GPU, using 2 million randomly sampled moments $\{\mathbf{m}_i\}$ and points $\{\eta_i\}$ for varying n (Fig. 4.4). Across all tests, the custom CUDA implementation is considerably faster, both for the primal (*forward*) and adjoint (*backward*) computations as well as in 32-bit and in 64-bit precision. The over $1000\times$ increase in runtime ($n = 3$, 32 bits) is related to root finding using the companion matrix for $n > 2$, because PyTorch seemingly does not solve for eigenvalues on the GPU (the CPU runtime is similar). Surprisingly, the corresponding function is also faster for 64-bit input, which explains why the 64-bit primal algorithm for $n > 2$ is faster than the 32-bit variant. We have no conclusive explanation but this behavior is consistent across different machines, operating systems, and PyTorch versions.

Regardless, the variants $n = 1$ and $n = 2$ use closed-form expressions for the roots, so they are unaffected by this anomaly. In 32-bit, the CUDA implementation is up to three orders of magnitude faster, and in 64-bit up to two orders. The default variant of the CUDA implementation recomputes *all* intermediate values from the forward pass in the backward pass, which makes it at least as expensive as the primal algorithm. Still, the runtime is much lower than the AD graph traversal. The observed performance benefit in an application will certainly be lower because a differentiable rendering pipeline consists of various costly operations and the GPU might not be saturated by the number of computed bounds. If the algorithm is used for shadow mapping, 2 million bounds correspond roughly to rendering a 1k image with two light sources. Even for $50\times$ larger input (~ 100 Million bounds), for $n \leq 2$, the 32-bit primal algorithm runs in less than 5 milliseconds and the adjoint algorithm in under 20 milliseconds (Fig. 4.5, left column). The double-precision runtime is significantly worse and roughly aligns with the theoretical performance of the tested GPU ($64\times$ higher FP32 throughput).

The memory footprint of the CUDA implementation is very predictable as storage from global memory is only allocated for the output. If N is the number of bounds to compute, the primal algorithm generates an array of size N and the adjoint algorithm an array of size $N(2n + 2)$ (for $2n + 1$ moment adjoints and the adjoint of η), therefore the memory requirements are linear (Fig. 4.5, right column, mind the log-scaled x-axes). Specifically, in contrast to the PyTorch AD implementation, no computation graph is constructed during the primal algorithm, which leads to a *significantly* smaller memory footprint (Fig. 4.4). For the adjoint computation, the AD implementation also only allocates memory for the outputs.

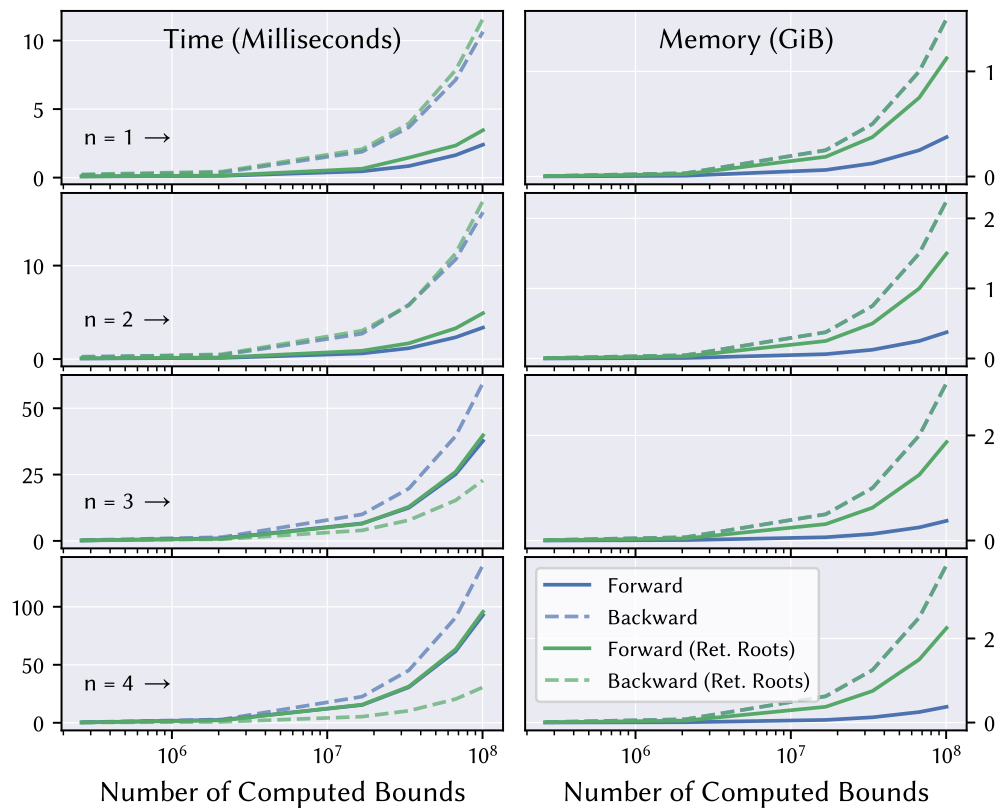


Figure 4.5: Runtime and memory requirements of the 32-bit C++/CUDA implementation with increasing number of computed bounds. The rows correspond to orders $n = 1, 2, 3, 4$, from top to bottom. Mind the log-scaled x-axis.

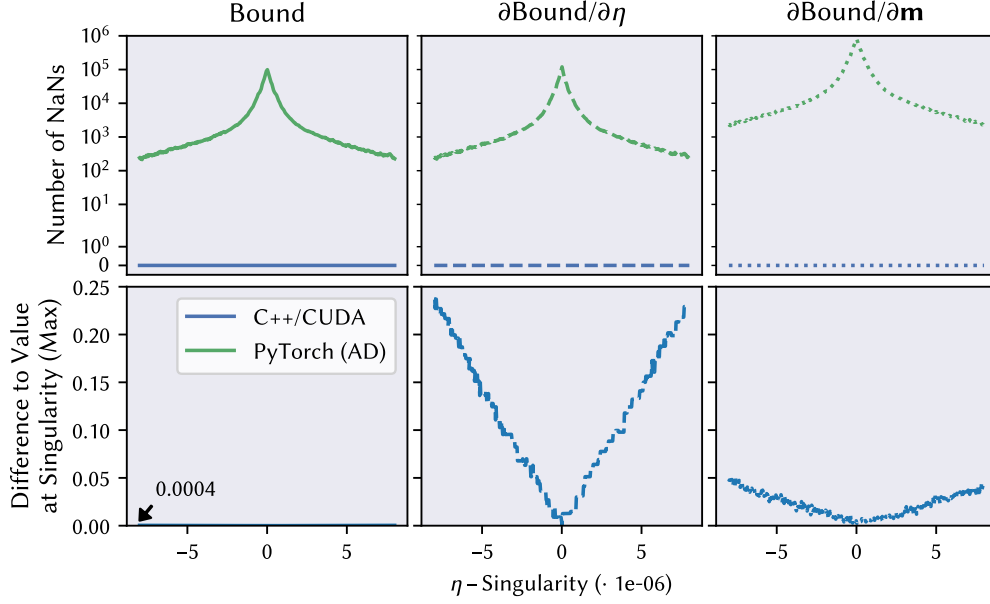


Figure 4.6: Evaluating the bound near singularities can be numerically unstable, leading to undefined output in the forward and backward computation (top). The implementation of an algorithm that considers the properties of the bound at singularities (“C++/CUDA”, Section 4.3.1) is more robust than the naive implementation based on automatic differentiation (AD).

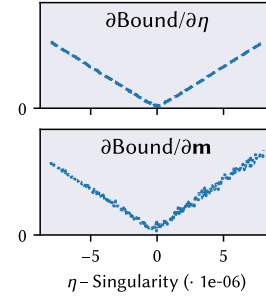
Retaining Roots The default C++/CUDA implementation outputs only the bounds from the primal kernel. In the adjoint kernel, the required quantities (e.g., the Cholesky factorization and the roots) are recomputed by re-running the primal algorithm. For $n > 2$, root finding is an iterative procedure (Section 4.3.1), which can become the dominating operation during the adjoint phase, in terms of runtime. We have therefore implemented a “retained roots” variant that returns the roots from the primal computation and re-uses them during the adjoint computation. The variant has a similar computation time for the primal part and an increased memory complexity, which is still significantly less than AD, but a considerably lower runtime for the adjoint part for $n > 2$ (Figures 4.4 and 4.5).

Stability Analysis The bounding functions are undefined at a finite set of points, where η takes the value of one of the roots of the polynomial P_n (and this is not a pathological case, as shown in Section 4.4.1). As η approaches a singular point, one root x_i tends to infinity, which can compromise the accuracy of the bound and its derivatives, specifically in 32-bit precision.

We use the following procedure to study the implementations near the singularities: for moments $\mathbf{m}$, we generate points η_k inside an ϵ -neighborhood ($\epsilon = 8\text{e-}6$) around each singularity y_j ($j = 1, \dots, n$), and compute the bound and its derivatives. We also evaluate the bound and the derivatives *exactly* at the singular point (up to floating-point precision). At each η_k , two traits are desirable : (1) the bound and the derivatives remain finite since undefined values can severely compromise a gradient-based optimization, and (2) their values tend towards the value at the singularity.

For $n = 1, \dots, 5$ we randomly generate moments, then compute the number of NaNs and the (absolute) difference to the value at the singularity (Fig. 4.6). Results are for 32-bit precision and the difference is reported as the *maximum* across all n and moment vectors. The PyTorch (AD) implementation generates almost no usable values near a singularity, which, to a large part, is due to failures in its root-finding procedure. The C++ implementation generates no NaN output since its operations are carefully designed around the limit case (Section 4.3.1). This is also reflected by the values tending to the value at the singularity (Fig. 4.6, second row).

We found the higher-order powers in the computation of δx_i (Eq. (4.30)) and the product in Eq. (4.27) particularly troublesome: without the measurements for $n = 5$, the derivatives tend even more clearly to the singularity value (inset). Unexpectedly, the 64-bit precision variant behaves *less* robust near singularities, which is likely due to the reliance on overflow semantics. We consider the 64-bit implementation experimental and leave a deeper investigation to future work.



4.4 Applications

4.4.1 Differentiable Shadow Mapping

In rendering, shadows from point lights (including those at infinity) are often approximated with shadow mapping [Williams, 1978] (see Section 2.2.2), and, as shown in Chapter 3, pre-filtered shadow mapping is a suitable framework for differentiable rendering. The general idea behind pre-filtering techniques based on moments [Peters and Klein, 2015; Donnelly and Lauritzen, 2006] is to obtain a set of moment maps by filtering the shadow map f with a compact kernel:

$$m_i(\mathbf{u}) = \int_{\mathcal{U}} k_{\mathbf{u}}(\mathbf{p}) f(\mathbf{p})^i d\mathbf{p}, \quad 0 \leq i \leq 2n, \quad (4.31)$$

where the kernel fulfills $\int k_{\mathbf{u}}(\mathbf{p}) = 1$, so $m_0 = 1$ (c.f. Eq. (3.7)). If $d(\mathbf{x})$ is the distance of a point $\mathbf{x}$ to the light and $\mathbf{u}_{\mathbf{x}}$ is its position in the shadow map, then the visibility is³

$$V_{\mathbf{m}}(x) = L(d(\mathbf{x}), m_1(\mathbf{u}_{\mathbf{x}}), m_2(\mathbf{u}_{\mathbf{x}}), \dots), \quad (4.32)$$

which is a lower bound to the result of percentage closer filtering [Reeves et al., 1987].

An observation in Section 3.1.3 was that the visibility of Variance Shadow Mapping (VSM) by Donnelly and Lauritzen [2006], based on two power moments $(1, m_1, m_2)$, is differentiable. In this chapter, we proved that L is differentiable for an arbitrary number of moments, motivating more general techniques like Moment Shadow Mapping (MSM) [Peters and Klein, 2015] with four power moments $(1, m_1, m_2, m_3, m_4)$.

³In this context, L represents the lower bound from Eq. (4.4) and not the linear projective transformation $L(\mathbf{x})$ introduced with shadow mapping (Section 2.2.2). The latter is here represented by $\mathbf{u}_{\mathbf{x}}$.

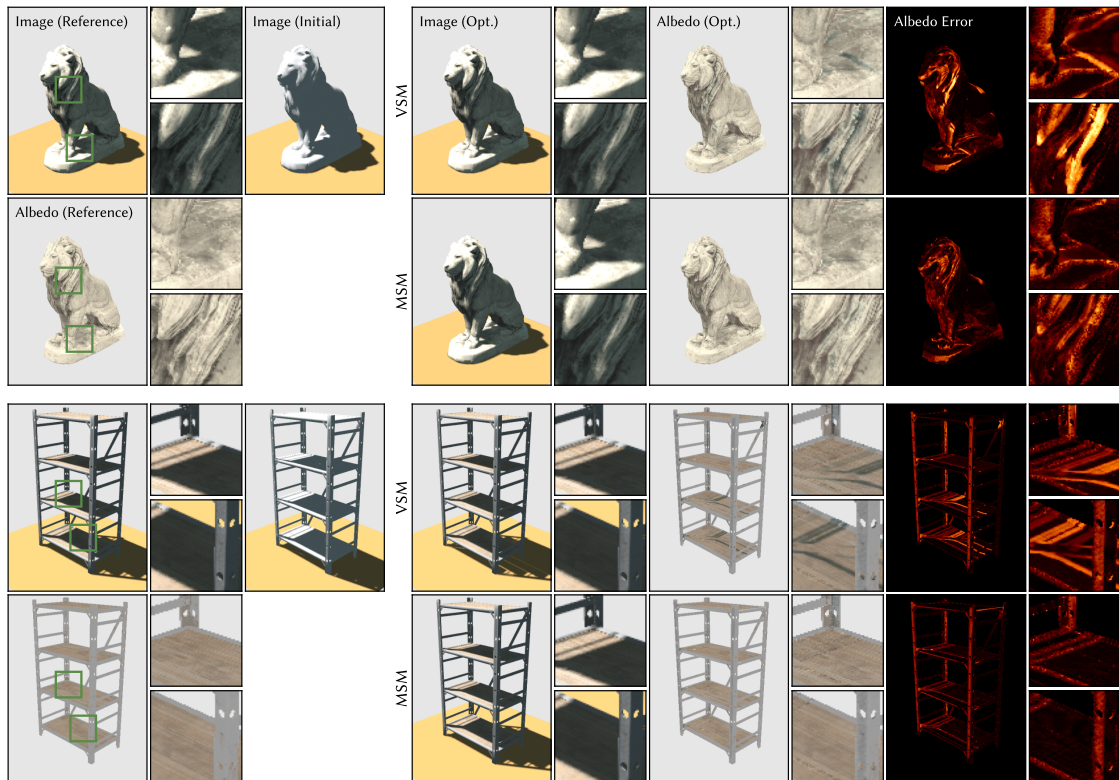
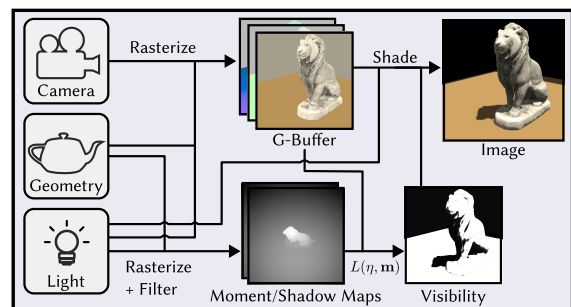


Figure 4.7: Optimizing the light position and spatially-varying albedo of an object from a single reference image. Both, Variance Shadow Mapping (VSM) and Moment Shadow Mapping (MSM) are based on the theory of moments, but Moment Shadow Mapping uses more moments, which leads to a more accurate estimation of visibility. This is crucial if the appearance of an object is decomposed because otherwise, remains of the shadow are baked into the albedo (see albedo optimized with VSM). The target reference image is generated using percentage closer filtering [Reeves et al., 1987].

We have implemented a rasterization-based differentiable renderer with shadow mapping (inset): the geometry is first rasterized, producing the G-buffer, a collection of attribute images needed for shading (e.g. diffuse albedo and normals). Shadow maps are rendered and used to compute the surface visibility. All data is then combined to the final image. Gradients flow along the shown edges.



Light Leaking in Appearance Decomposition

Variance Shadow Mapping suffers from light leaking: if the depth has high variance in the filter support, the computed bound is not sharp and visibility is overestimated. The adverse effect of light leaking is more significant in *inverse* settings: in forward render-

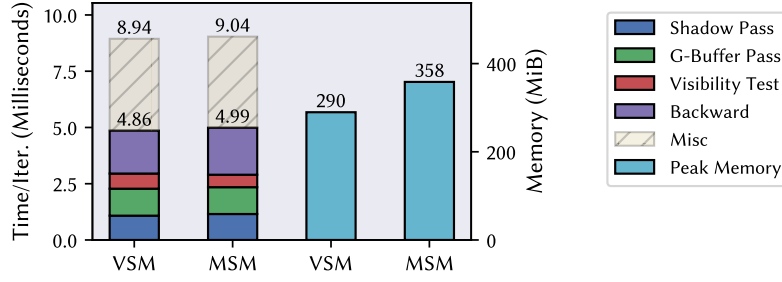


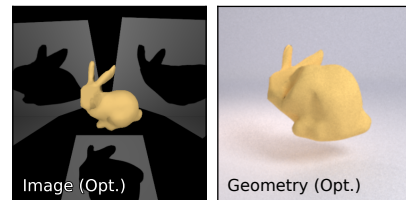
Figure 4.8: Runtime of Variance Shadow Mapping (VSM) and Moment Shadow Mapping (MSM) for the appearance decomposition task. Both methods are similarly efficient, but MSM requires slightly more memory. “Misc” includes the remaining operations, such as anti-aliasing and evaluating the objective.

ing, subtle artifacts might stay unnoticed by human observers, but with differentiable rendering, images are often compared to a reference that potentially originates from the physical world. Moment Shadow Mapping strongly reduces light leaking by providing a sharper bound to the visibility (Fig. 4.1).

Faithful estimation of visibility is critical when the *appearance* of an object is decomposed into the *material* and the *illumination*, e.g., as a post-processing step of a (neural) 3D reconstruction pipeline. Jointly optimizing the light direction and albedo texture from a single reference image yields significantly more accurate results with Moment Shadow Mapping (Fig. 4.7): while VSM and MSM obtain the correct light direction, the VSM albedo texture shows traces of the shadow because it must account for areas with reduced shadow intensity. Our MSM implementation not only generates higher quality textures, but is as fast as the VSM implementation from Chapter 3 (Fig. 4.8). The C++/CUDA implementation (Section 4.3.2) is critical to matching the performance: a PyTorch (AD) implementation of MSM is observed to have twice the VSM runtime, and, more importantly, does not converge due to NaN outputs (Section 4.4.1).

Comparison to Ray Traced Shadows

Instead of approximating visibility, it can be accurately determined using ray tracing, and, more generally, realistic images can be synthesized with systems for physically-based, differentiable rendering. The impact on runtime is significant: an approximate, differentiable renderer with Moment Shadow Mapping finishes an image-based reconstruction task in seconds, while the physically-based system MIT-SUBA 3 [Jakob et al., 2022b] requires minutes (Fig. 4.9). Importantly, the visibility gradients generated by MSM lead to a successful reconstruction: if the visibility is detached from the AD graph, the reconstruction fails (see inset). With significant contribution from indirect illumination, approximating only shadows would not be sufficient, and additional techniques are required.



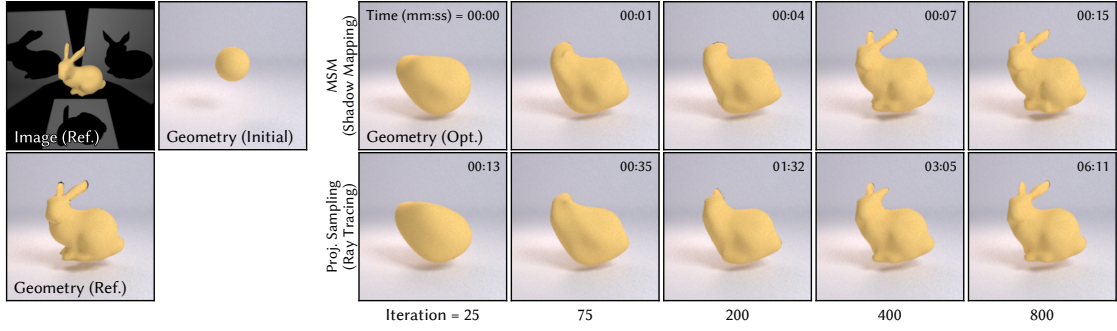


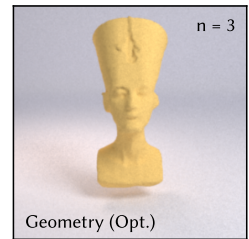
Figure 4.9: 3D mesh reconstruction from a single image of an object illuminated by three spotlights. The differentiable, moment-based shadow mapping technique (MSM) converges significantly faster than the ray tracing approach based on projective sampling [Zhang et al., 2023], with similar reconstruction results. The reference image is rendered using ray tracing and serves as a reference for both methods. The reference image does not include indirect illumination. The optimization uses gradient preconditioning [Nicolet et al., 2021] and a rotation-equivariant instance of ADAM [Kingma and Ba, 2017; Ling et al., 2022].

Numerical Considerations

The PyTorch AD implementation, in contrast to the C++/CUDA implementation, generates undefined outputs and gradients if the query point is close to a singularity (Section 4.3.2, Paragraph *Stability Analysis*). In forward rendering, this merely leads to visual artifacts, but in inverse rendering, this can corrupt a (potentially long-running) optimization. Fig. 4.10 shows that this is not a pathological case: almost all surfaces directly visible to a light will result in queries close to singularities. The reason is that the moments are generated by filtering over a compact region in the shadow map (Eq. (4.31)). Except at discontinuities, the depth values in the filter support will be correlated and only subtend a small section of the moment curve with concentrated singularities. The depth values of visible points will be close to the depths in the filter support, so fall within that same segment of the moment curve and near the singularities. For this reason, we were unable to achieve a stable convergence using the PyTorch AD implementation.

Moment Biasing A central feature of Moment Shadow Mapping is the ability to *exactly* reconstruct the visibility if the filter support region contains no more than two surfaces (perpendicular to the light direction) [Peters and Klein, 2015]. For such cases, the moments are slightly biased to ensure that the Hankel matrix is non-singular. We use the bias vectors by Münstermann et al. [2018] and test the recommended bias for $n = 2$ ($\alpha = 5 \cdot 10^{-7}$), but singular matrices are still occasionally encountered. We use a slightly higher bias (as a rule of thumb, at least twice the recommended value).

Some experiments run successfully with the bias recommended by Münstermann et al. [2018, Table 1, Supplementary Material] (e.g., Section 4.4.1), so it is worth testing different values to strike the right balance between accuracy and robustness in an application. With stable



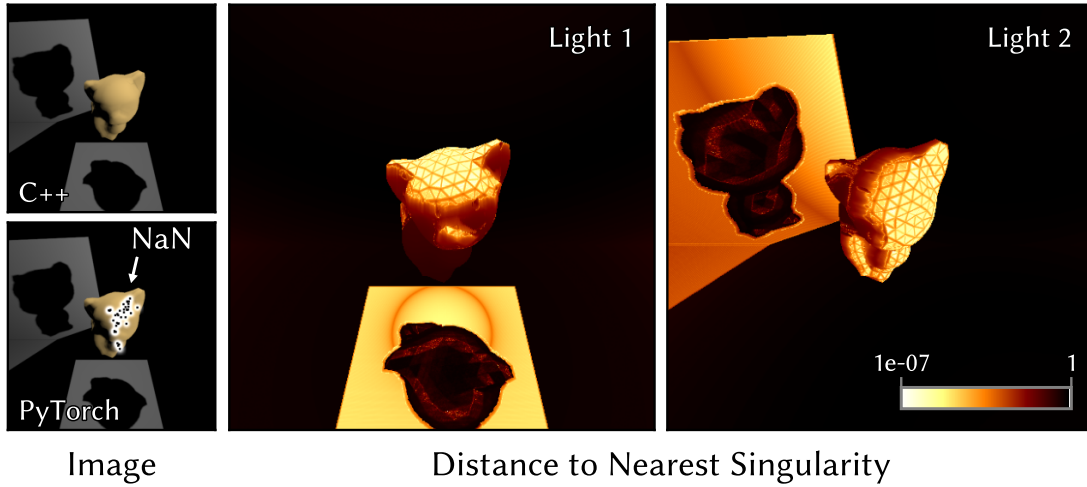


Figure 4.10: Moment shadow mapping uses bounding functions that are undefined at certain points, close to which evaluating the bound and its derivatives becomes numerically challenging. Even in a simple scene, query points are close to these singularities, which leads to undefined output in naive implementations (“PyTorch”), motivating more robust approaches (“C++”).

biasing, the number of moments can be increased from 4 ($n = 2$) to 6 ($n = 3$) and further, still resulting in successful 3D reconstructions (see inset).

Quality of Rasterization Derivatives.

The quality of the approximated visibility decreases with the shadow map resolution. Low-resolution shadow maps also harm the inverse setting (Fig. 4.11). First, they cannot reproduce fine structures in the target shadow, and second, they are limited by the differentiable rasterizer: the implementation uses `NVDIFFRAST` [Laine et al., 2020], which differentiates discontinuous coverage by anti-aliasing silhouettes. If pixels are larger than (projected) triangles, the silhouette detection can fail and coverage is not differentiated (Fig. 4.12). As a result, shadow boundaries carry no gradients w.r.t. the geometry.

We have made the same observation in Chapter 3, Section 3.2, but, until now, have not yet shown the adverse effect in a reconstruction task. However, this is not a conceptual issue of differentiable shadow mapping, and it can likely be mitigated by using other approaches to differentiable rasterization [Liu et al., 2019a; Pidhorskyi et al., 2025].

4.4.2 Differentiable Visibility for Volume Rendering

Volume rendering [Kajiya and Von Herzen, 1984] is commonly used in computer vision for volumetric scene representations, such as (neural) radiance fields [Mildenhall et al., 2020]. If $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$ is a camera ray, the image formation model assigns to it a color

$$\int_0^\infty \mathbf{c}(\mathbf{r}(t)) T(t) \sigma(\mathbf{r}(t)) dt, \quad (4.33)$$

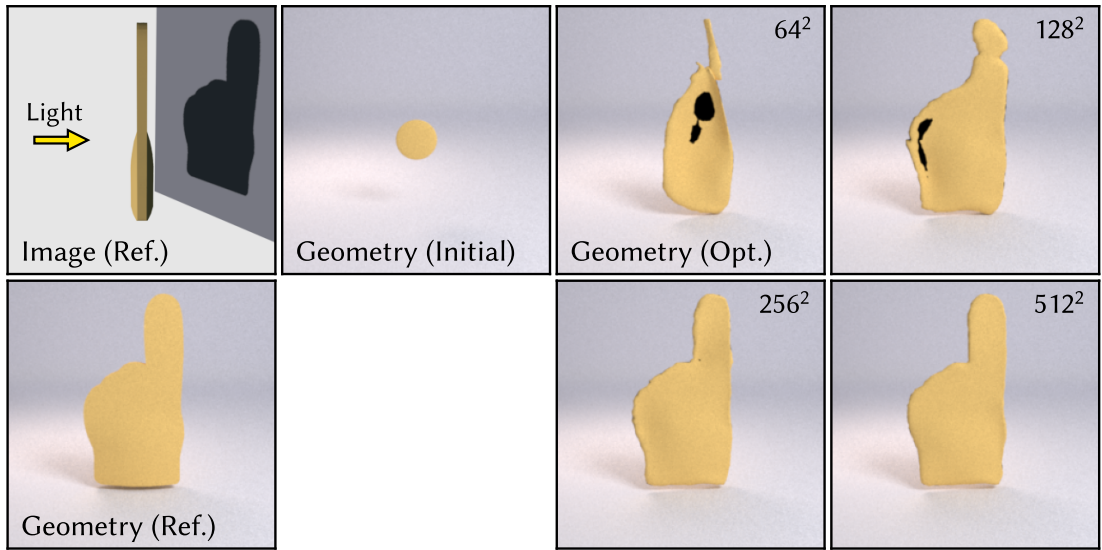


Figure 4.11: An optimization with shadow mapping can fail if the underlying differentiable renderer misses the relation between changes in geometry to changes in the shadow map (here caused by low shadow map resolution).

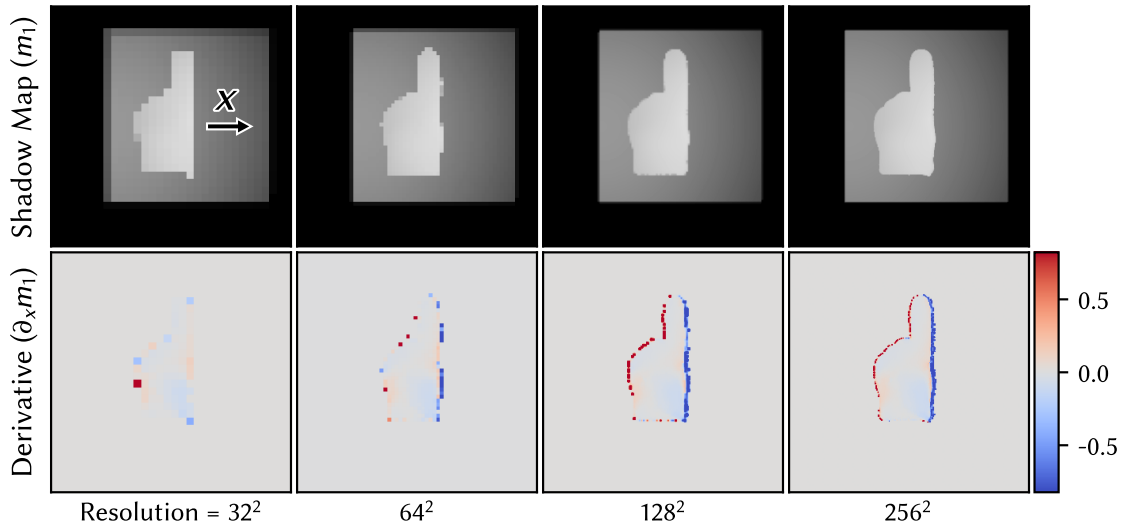


Figure 4.12: Differentiable shadow mapping can fail at capturing the changes of the shadow map with respect to movement of the geometry, depending on the underlying differentiable renderer. In this specific case, the renderer relies on silhouette detection, which fails for low-resolution shadow maps.

where $\mathbf{c}$ is a color field, σ is a density field and T is the transmittance. Intuitively, the density σ measures the probability (per unit length) of absorbing or scattering a photon from its path and transmittance

$$T(t) = \exp \left(- \int_0^t \sigma(\mathbf{r}(z)) \, dz \right) \quad (4.34)$$

is the fraction of radiance being transported from 0 to t .

When volume rendering is combined with a physically-based image formation model, e.g., to decompose the appearance into material and illumination or to use shadows as additional reconstruction cues, one must compute the transmittance from a light source to each point on a primary ray. Numerically estimating the transmittance can lead to quadratic complexity, which, even for small images, quickly becomes infeasible, especially when storing a computation graph for reverse-mode differentiation. Several ways of *approximating* the transmittance have been proposed, one of which is applying shadow mapping by rendering and storing the expected termination depth from the perspective of a light

$$f(\mathbf{u}) = \int_0^\infty t T(t) \sigma(\mathbf{r}(t)) \, dt, \quad (4.35)$$

where $\mathbf{u}$ is a position in shadow map space, and then using soft, ad-hoc extensions of the binary shadow test to make it differentiable [Liang et al., 2022; Tiwary et al., 2022]. While the approximation is efficient and induces dense gradients in a volume, it is difficult to achieve shadows with reasonable intensity (Section 4.4.2) and the approach is not volume-based, i.e., it does not properly handle empty space, which is desirable when combining multiple volumes or different scene representations (Section 4.4.2).

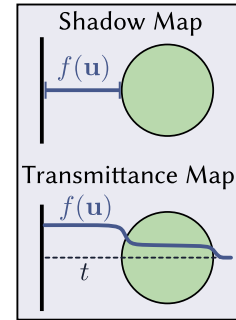
Transmittance Mapping Inspired by classical work in real-time graphics [Peters et al., 2016; Jansen and Bavoil, 2010; Lokovic and Veach, 2000; Delalandre et al., 2011] we propose a principled approach to transmittance approximation for differentiable volume rendering. Similar to how a shadow map stores the minimum depth along a ray emanating from the light source, a *transmittance map* stores a compact representation of the transmittance *function* along a ray (see inset).

Following Münstermann et al. [2018], we define absorbance as

$$A(t) = -\ln(T(t)) = \int_0^t \sigma(\mathbf{r}(z)) \, dz. \quad (4.36)$$

A is non-decreasing and non-negative as long as σ , the volume density, is non-negative. This allows approximating A using moments (Section 4.1): a transmittance map is built by computing the moments

$$f_i(\mathbf{u}) = m_i = \int_0^\infty z^i \sigma(\mathbf{r}(z)) \, dz, \quad 0 \leq i \leq 2n. \quad (4.37)$$



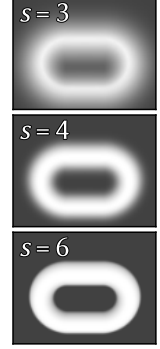
The representation is compact because it only stores $2n + 1$ values in each pixel (“TM- n ” will refer to order- $2n$ transmittance mapping, i.e., a transmittance map with moments $m_0, \dots, m_{2n}$). The bounding functions from Eq. (4.4) then approximate the transmittance:

$$T(t) \approx \exp \left(- (1 - \beta)L(t) - \beta U(t) \right), \quad (4.38)$$

where $\beta \in [0, 1]$ blends between the bounds [Münstermann et al., 2018]. The approximation quality increases with the order n and transmittance maps are filterable, i.e., the approximation still “works” as expected when they are filtered or sampled from with (bilinear) interpolation. Using the alternative form from Appendix A.1, following Peters et al. [2016], transmittance maps can also be easily integrated into existing volume rendering frameworks, e.g. by Li et al. [2023].

Implicit Surfaces and Volumes Differentiable visibility in volumes might be of particular interest to the 3D reconstruction community because the properties of volume rendering, namely trivial differentiability and dense gradients, are often exploited for reconstruction using implicit surfaces [Wang et al., 2021; Oechsle et al., 2021; Yariv et al., 2021]: constructions like NeuS [Wang et al., 2021] use differentiable volume rendering for an indicator or signed distance function $g(\mathbf{x})$ transformed into a volume density⁴

$$\sigma(\mathbf{r}(t)) = \phi_s(g(\mathbf{r}(t))). \quad (4.39)$$



The parameter s controls the spread of the bell-shaped function ϕ_s and therefore the opacity of the induced density field (inset). We use this construction throughout the section, so some signed distance function induces the shown volumes.

Visibility Gradients for the Density

We first verify, using simple, analytic shapes, that the gradients computed from the transmittance approximation affect the density field and the underlying implicit surface. Fig. 4.13 shows that a perturbation of geometry is reflected by the derivatives, both for directly visible surface points and the cast shadow computed using transmittance mapping.

The quality of the approximation and its gradients is compared to shadow mapping using a simple task: recover the pose of an object, i.e., the position and orientation, from only a *single* image. Shadows can be important cues for this task [Shafer and Kanade, 1983; Liu et al., 2023] and it might be approached by volume rendering implicit surfaces [Qu et al., 2023]. The image formation model simply accumulates the visibility V along the primary rays

$$\int_0^\infty V(\mathbf{r}(t)) T(t) \sigma(\mathbf{r}(t)) dt, \quad (4.40)$$

where V is the (approximate) transmittance from the light source to the point $\mathbf{r}(t)$. Even in this simple setting, differentiating shadow rays is already infeasible due to the excessive memory requirements.

⁴A direction-dependent normalization factor is omitted for brevity.

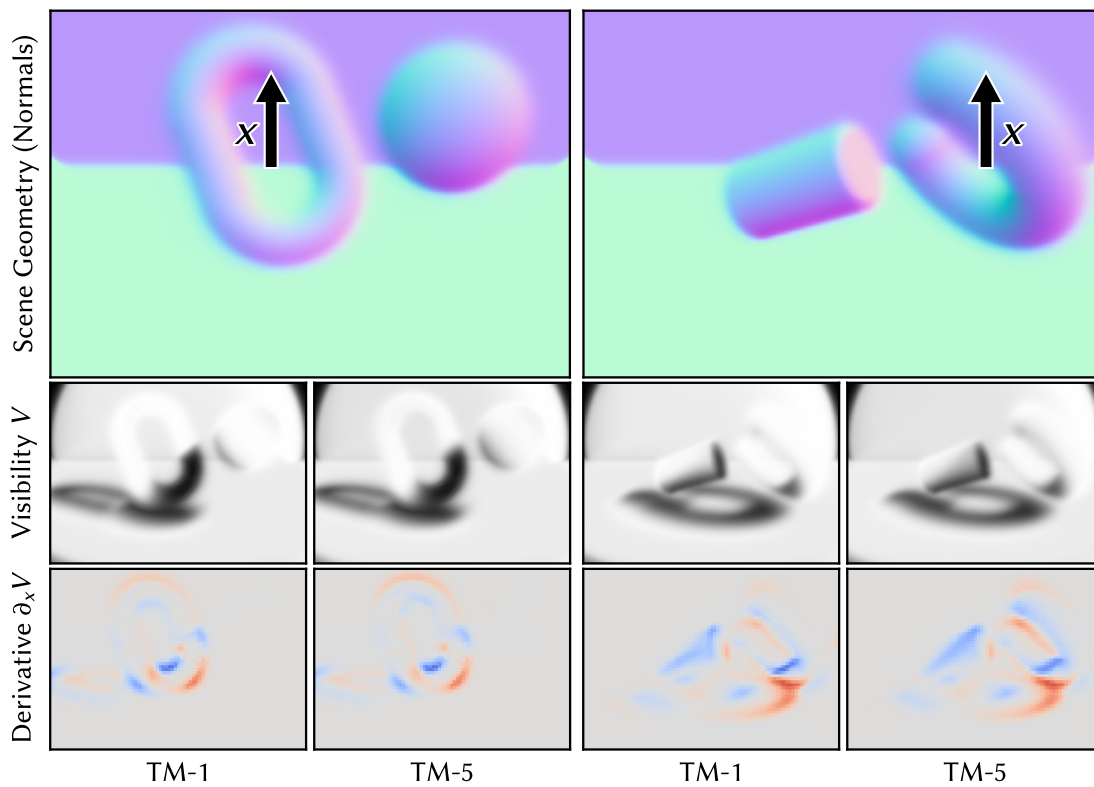


Figure 4.13: The quality of the approximation of transmittance increases with the number of moments. TM- n denotes an order n with $2n + 1$ moments. The approximation is differentiable, and the variation of the geometry with a parameter x is correctly reflected in the derivatives.

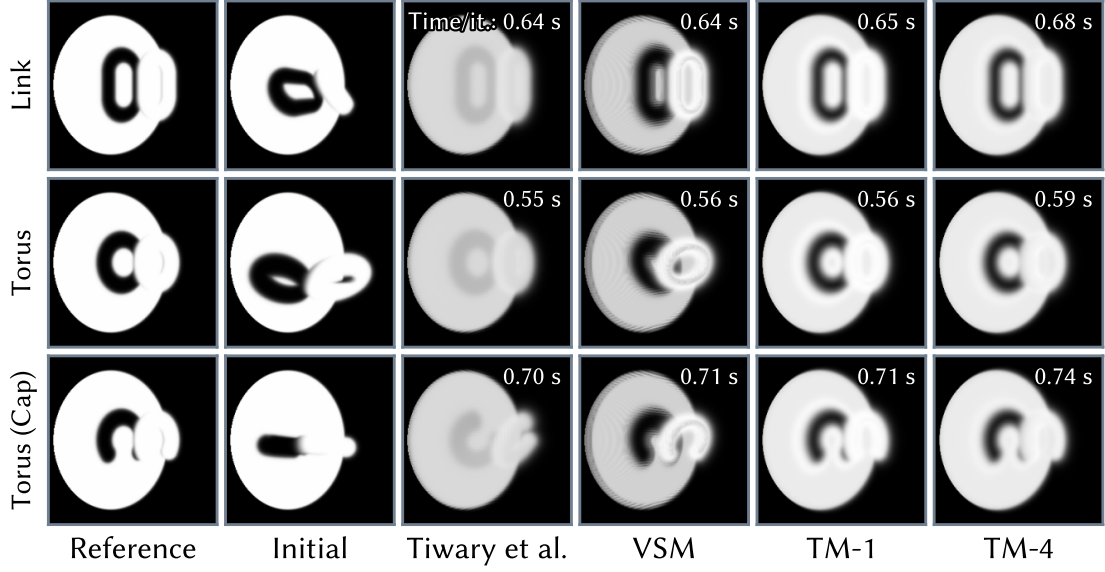


Figure 4.14: Optimizing the pose of simple, analytic shapes with differentiable volume rendering. The approximation of shadow mapping ([Tiwary et al., 2022] and VSM) is not accurate and often fails to find the pose, while the more principled moment-based (TM- n) approach succeeds. The reference is rendered at a slightly higher density ($s = 5$) than used in the optimization ($s = 4$) to mimic a more “realistic” reference.

We compare against the shadow mapping variants by Tiwary et al. [2022], which has a sigmoid-based soft visibility test, and Variance Shadow Mapping (VSM) by Donnelly and Lauritzen [2006]⁵. Fig. 4.14 shows that the shadow mapping visibility is far from the shadow ray reference, leading to inaccuracy in the estimated pose.

We measure the translation error Δt , the Hausdorff distance D_H , both scaled by factor 10, and the runtime per iteration t , as the median over multiple runs, averaged over different shapes (inset table). Transmittance maps consistently outperform shadow mapping in the geometric error measures. The difference between the transmittance map orders is marginal but could become noticeable for multiple (articulated) objects. Most importantly, transmittance mapping is more accurate and at the same time equally efficient.

	$\downarrow \Delta t \downarrow D_H \downarrow t [s]$		
Tiwary et al.	0.50	0.33	0.58
VSM	0.94	0.67	0.58
TM-1	0.09	0.08	<u>0.59</u>
TM-2	0.07	0.08	<u>0.59</u>
TM-3	<u>0.08</u>	<u>0.09</u>	0.60
TM-4	<u>0.08</u>	<u>0.09</u>	0.61
TM-5	0.07	0.08	0.63

Visibility Through Empty Space

The issue arising from shadow mapping not being a principled method for approximating transmittance is best shown by testing a trivial property of transmittance: if the density in a volume is zero, the transmittance is one, because no light is absorbed. The test setting is inspired by shadow art [Mitra and Pauly, 2009]: planes (triangle meshes)

⁵While we have previously used VSM only with differentiable rasterization [Worchel and Alexa, 2023b], it is included here because it has a more principled visibility test than the heuristics.

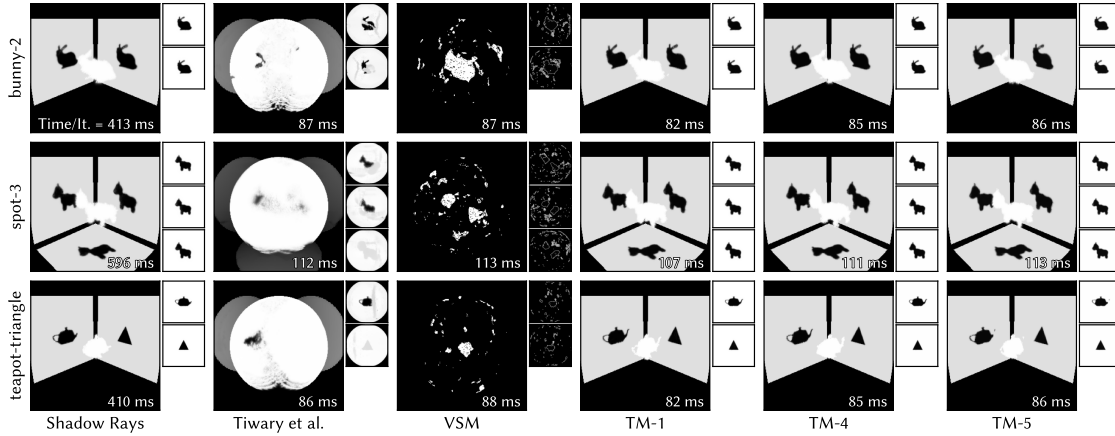


Figure 4.15: Performing shadow art by using differentiable volume rendering. The shadow receiver planes are triangle meshes and a signed distance function, defined on a grid, is optimized such that its cast shadows resemble target shadows on the planes. Methods based on shadow mapping (by Tiwary et al. [2022] and Variance Shadow Mapping (VSM) [Donnelly and Lauritzen, 2006]), do not handle empty space because they are not volume-based. Using a moment-based approach to approximate the transmittance (TM- n) yields results close to the shadow ray reference, with the same runtime.

are placed around a grid representing an SDF. The SDF and the density-controlling variable s are optimized to cast given target shadows on the planes, by using an objective function that compares renderings of the cast shadows to the targets. The topological flexibility of the SDF is beneficial for this task and volume rendering ensures dense gradients. To maintain a valid SDF, the function is redistanced by solving the Eikonal equation in each iteration [Detrixhe et al., 2013; Vicini et al., 2022].

Shadow rays do not have quadratic complexity in this setting because the receiver is rasterized, not volume-rendered, so they can serve as a reference. While transmittance maps generate results close to the shadow ray solution, shadow mapping fails (Fig. 4.15): since the receiver planes are not volumes, they are not considered in volume rendering, so, whenever a light ray misses the implicit surface, the expected terminated depth will be zero (compare Eq. 4.35). Shadow mapping behaves as if an object directly in front of the light source obstructs the view. Therefore, the unobstructed parts of the receiver plane will appear in shadow, and the obstructed parts will, too. Since the shadow intensity in the method by Tiwary et al. [2022] depends on the distance between the occluder and receiver some areas in shadow appear grayish. The optimization behaves contrary to the expectation and recreates the bright parts of the target image using the volume (the sphere emerges) while the shadow is recreated by generating free space. Transmittance maps handle free space correctly by reporting constant 1 transmittance, while being about $4\times$ faster than shadow rays – as fast as shadow mapping.

3D Reconstruction from Low-Density Volumes

Shadow mapping does not necessarily fail and has been successfully used in differentiable volume rendering: if s is high, the volume density spikes at the surface, and vol-

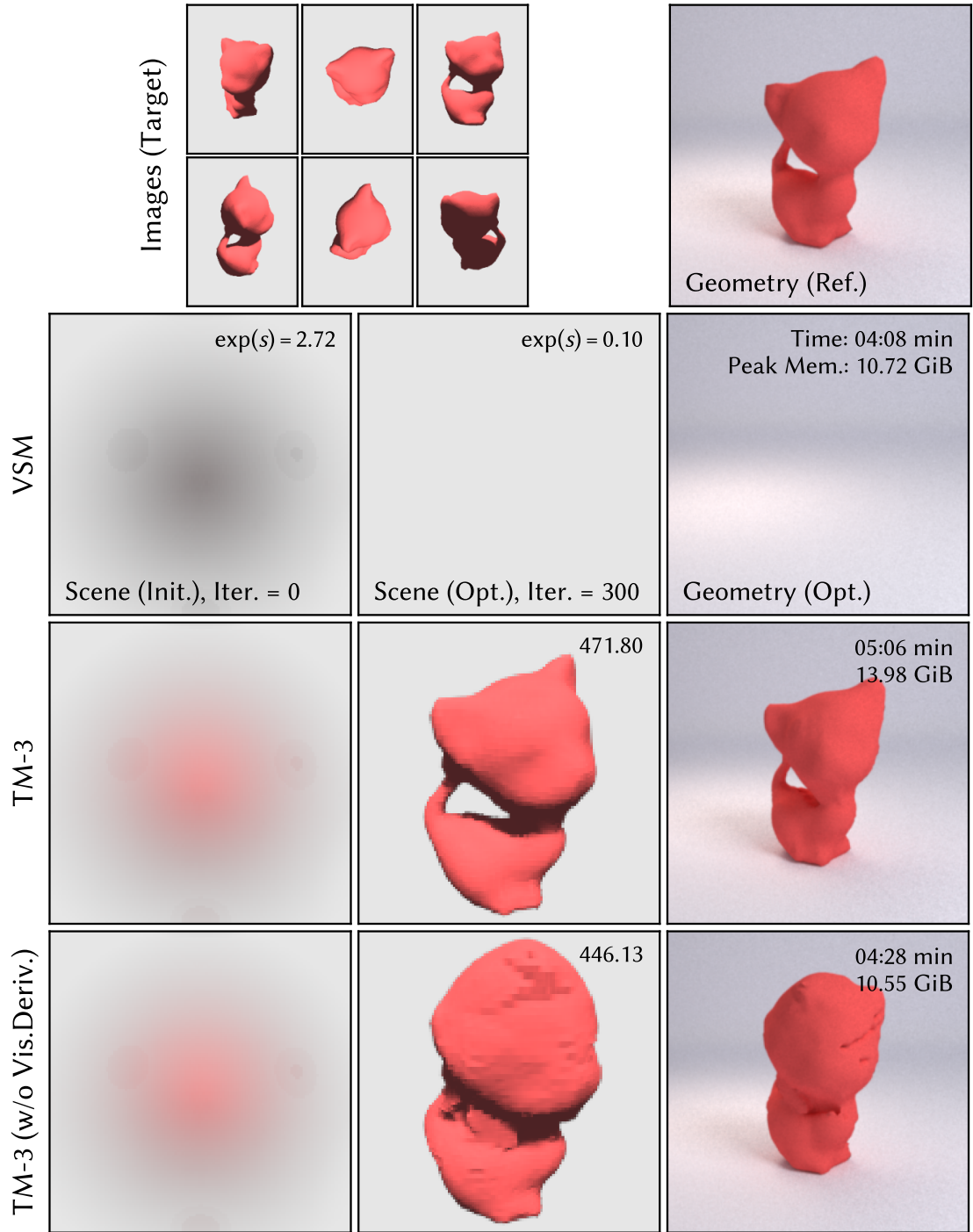


Figure 4.16: Multi-view 3D reconstruction with differentiable volume rendering using approximate visibility. Shadow mapping (VSM) provides an inaccurate estimate of visibility, leading to divergence of the optimization. The principled moment-based technique (TM-3) succeeds, at slightly higher runtime and memory consumption. When visibility gradients are detached, the reconstruction fails (last row). Light and albedo are assumed to be known.

ume rendering collapses to surface rendering. Since s (c.f. Eq. 4.39) is often optimized with the volume, from low to high [Wang et al., 2021; Li et al., 2023], the point at which shadow mapping becomes a good approximation depends on the optimization schedule. This leads to an intricate balance: starting with a very low s induces dense gradients, which more robustly localize the surface, independent of the initial state. However, the inaccurate visibility approximation at earlier iterations increases the risk of becoming stuck in suboptimal local minima.

This effect is observable in 3D reconstruction from multi-view images (Fig. 4.16). Shadow mapping (VSM) navigates into an unrecoverable state by decreasing the target density until the volume dissolves and the implicit surface disappears, while transmittance mapping obtains a reasonable reconstruction of the target shape. The visibility gradients are substantial in this setting: with visibility detached from the AD graph, e.g. as in NVDIFFRECMC [Hasselgren et al., 2022, Sec.3.1, Shadow Gradients], the reconstruction fails.

We also want to highlight that this optimization, in each iteration, renders all views (each 190×256 pixels) using 128 samples while maintaining a performance of one iteration per second and reasonable memory consumption (Fig. 4.16). Approaching this task with shadow rays would be difficult without stochastic ray sampling.

4.5 Conclusion

In this chapter, we have shown that measures can be bounded in a differentiable manner by proving that the lower and upper bounds arising from the truncated Hamburger moment problem are continuously differentiable in their parameters. As a consequence, the use of various approximate rendering techniques, which are based on these bounds, has been justified in differentiable rendering. We have shown that this contributes new approaches for differentiable shadow mapping and volume transmittance, which are not only more principled than existing ones, but also more accurate and equally efficient.

Limitations and Future Work The techniques proposed for differentiable shadow mapping and transmittance inherit the classical limitations of shadow mapping: the approach makes simplifying assumptions about the light sources, such that area lights or image-based lighting are not supported, and the quality of the approximation depends on the spatial discretization. If the shadow/transmittance map is too coarse, small structures are missed; one can increase the resolution, but the memory requirements increase as well. Adaptive shadow mapping techniques might be applicable in this context [Fernando et al., 2001; Lefohn et al., 2007].

Numerical stability is a major challenge of moment-based bounds, especially for higher orders, as the relevant matrices become increasingly ill-conditioned and robustly finding polynomial roots more difficult. We have shown that implementations benefit from being designed around these constraints, but our analysis is certainly not exhaustive. For the forward direction, numerical stability has mainly been investigated for quantization [Peters and Klein, 2015; Peters, 2017] and similar exploration could be useful for the backward direction.

Lastly, we have not yet covered applications that exploit the full potential of higher-order bounds. For example, the transmittance mapping framework can be used for differentiable order-independent transparency based on the work by Münstermann et al. [2018]. Subsequent work has picked up and investigated this idea: Müller et al. [2025] explore differentiable moment-based order-independent transparency for 3D Gaussian splatting [Kerbl et al., 2023] as an alternative to ray tracing or per-pixel sorting.

CHAPTER 5

Radiative Backpropagation with Non-Static Geometry

In the previous chapters, we have approached efficient differentiable rendering by investigating how certain techniques from real-time graphics can be applied in this context. While real-time techniques are the foundation of many practical differentiable rendering systems, their approximate nature remains a fundamental limitation. Various applications require a more faithful simulation of light transport, for example, multi-bounce light paths in the design of optical lenses [Sun et al., 2021; Teh et al., 2025]. Even though real-time approximations exist for more complex phenomena, most techniques are ill-suited as they strive more for plausible *appearance* than for physical accuracy – this can easily steer optimizations to undesirable solutions.

Generally, such applications are better approached with physically-based differentiable rendering, as it adheres to the equations of light transport (Section 2.1.2). When it comes to improving efficiency, one typically wants to maintain physical accuracy and avoid approximations. Other measures are taken instead, such as using specialized tooling for differentiable rendering [Jakob et al., 2022a; Bangaru et al., 2023] or decreasing the required sample budget through variance reduction [Wang et al., 2023b; Belhe et al., 2024; Su and Gkioulekas, 2024; Yan et al., 2025]. Alternatively, the efficiency of differentiation might be approached on an algorithmic level: Gradient-based optimization of an objective function commonly relies on reverse-mode derivatives (see Section 2.3), which, in the case of light transport, can be efficiently computed using radiative backpropagation (RB) [Nimier-David et al., 2020; Stam, 2020] (see Section 2.4.3). Instead of building and storing a large computation graph for automatic differentiation (AD), radiative backpropagation explicitly computes derivatives using a *differential* light transport simulation. By replaying primal light paths [Vicini et al., 2021], this requires only constant memory and linear time in path length.

A subtle detail in the original formalism is that radiative backpropagation was derived under the assumption of *static* geometry, which means that a scene parameter θ , among others, does not alter the shape or position of surfaces, lights, and sensors. Many applications of differentiable rendering violate this assumption (see Section 2.4.1). Surprisingly, to our knowledge, the work by Zeltner et al. [2021] is the only one investigating the effects of *non-static* geometry, that is, the parameter θ is allowed to affect geometry, on radiative backpropagation – almost all other research of the non-static case focuses on handling discontinuous visibility (e.g., [Li et al., 2018a; Loubet et al., 2019; Zhang et al., 2020, 2023]). This is likely because the *boundary* term resulting from discontinuous visibility is unique to non-static geometry and generally challenging to compute,

whereas the *interior* term can be computed using either radiative backpropagation or automatic differentiation. However, we find that critical implications for radiative backpropagation are missed by the work of Zeltner et al. [2021]. This renders efficient and correct computation of interior derivatives infeasible in the non-static case.

In this chapter, we revisit radiative backpropagation and re-derive the theory without assuming static geometry. A central observation is that *differential* radiance, the quantity that governs the gradient simulation, is not transported in the same manner as normal radiance. This deviation from the original formalism and its consequences are not commonly discussed. In fact, we demonstrate through simple experiments that existing implementations of radiative backpropagation are biased when geometry is non-static, and that this compromises even the simplest inverse rendering tasks. The reason is that these implementations do not conform to the theory, accounting only for a subset of the newly arising derivatives. When including the “missing term”, we find that the parameterization of the rendering integral matters: only surface integrals lead to a simple, local formulation of derivatives, i.e., one in which changes in geometry do not affect the entire path. A path replay implementation based on our derivation leads to systematic convergence to the reference solution in the test settings, and provides efficient and unbiased estimates of the interior derivatives.

5.1 Rederiving the Differential Radiance Equations

In physically-based rendering, one is interested in differentiating the rendering equation [Kajiya, 1986] (Section 2.1.2) with respect to an arbitrary scene parameter θ . Here, we explicitly allow θ to affect scene geometry, such that, for example, it may alter the shape of surfaces. While the case of non-static geometry is often discussed in the context of handling discontinuous visibility, this is orthogonal to our main aim. A scene may well have occlusion boundaries that do not affect the integrand, or that only marginally contribute to the derivatives when the geometry changes. For now, we assume such scenes and defer a discussion of discontinuities.

The central observation by Nimier-David et al. [2020] and Stam [2020] is that the derivatives for θ can be explicitly computed by another form of light transport simulation involving *differential radiance* $\partial_\theta L$. This leads to radiative backpropagation (also see Section 2.4.3): the gradient of an objective function $\partial_\theta \mathcal{L}$ can be computed by *measuring* differential radiance with the *adjoint radiance* $\frac{\partial \mathcal{L}}{\partial L}$ during the simulation. The equations for differential radiance were originally derived under the assumption of static geometry, so in the following, we rederive them without this assumption.

5.1.1 Differential Scattering

Differential radiance is governed by *differential scattering*, which is obtained by differentiating both sides of the rendering equation (2.4):

$$\begin{aligned}
\partial_\theta L_o(\mathbf{p}, \boldsymbol{\omega}) = & \underbrace{\partial_\theta L_e(\mathbf{p}, \boldsymbol{\omega})}_{\text{Term 1}} \\
& + \int_{S^2} \left[\underbrace{\partial_\theta L_i(\mathbf{p}, \boldsymbol{\omega}_i) f(\mathbf{p}, \boldsymbol{\omega}, \boldsymbol{\omega}_i)}_{\text{Term 2}} \right. \\
& \quad \left. + \underbrace{L_i(\mathbf{p}, \boldsymbol{\omega}_i) \partial_\theta f(\mathbf{p}, \boldsymbol{\omega}, \boldsymbol{\omega}_i)}_{\text{Term 3}} \right] d\boldsymbol{\omega}_i^\perp. \quad (5.1)
\end{aligned}$$

The three terms are interpreted as follows:

- **Term 1:** Differential radiance is emitted by light sources if θ affects the emission (e.g. if θ is the brightness).
- **Term 2:** Differential radiance scatters like normal radiance according to the BSDF f
- **Term 3:** Differential radiance is emitted when the BSDF changes with θ (e.g. if θ affects the albedo of a surface).

This equation remains unchanged and is exactly the one by Nimier-David et al. [2020].

5.1.2 Differential Transport

The transport from one surface to another is described by *differential transport*, which is obtained by differentiating the transport equation (2.6):

$$\partial_\theta L_i(\mathbf{p}, \boldsymbol{\omega}) = \partial_\theta L_o(\mathbf{r}(\mathbf{p}, \boldsymbol{\omega}), -\boldsymbol{\omega}). \quad (5.2)$$

This is the final form presented by Nimier-David et al. [2020], in which case differential radiance would be transported just like normal radiance. However, this misses a subtle relation: when geometry is non-static, the intersection point $\mathbf{r}(\mathbf{p}, \boldsymbol{\omega}, \theta)$ can depend on θ .

Notice that the points $\mathbf{p}$ and $\mathbf{r}$ are to be understood as points in a (hypothetical) global *parametrization* $\mathcal{M} \subset \mathbb{R}^2$ of the surface, which is embedded in the *ambient space* $\mathbb{R}^3$. We denote the mapping from a point in the parameterization $\mathbf{p} \in \mathcal{M}$ to the ambient space by $S(\mathbf{p}, \theta) \in \mathbb{R}^3$, which naturally depends on the geometry. A change of the intersection point $\mathbf{r} \in \mathcal{M}$ with θ can now arise in two ways: (1) the surface at $\mathbf{p}$ evolves in the ambient space, such that the origin $S(\mathbf{p}, \theta)$ of the ray cast to determine $\mathbf{r}$ moves; or (2) the surface at $\mathbf{r}$ evolves in ambient space.

Zeltner et al. [2021] have previously investigated a combination of radiative back-propagation and non-static geometry with reparameterization-based discontinuity handling [Loubet et al., 2019; Bangaru et al., 2020]. However, they only transform the differential scattering equation (5.1) to account for the reparameterization, but do not consider the relation above and its implications on the derivatives.

The potential movement of the intersection point with θ must be considered when differentiating the transport equation (5.2). Applying the multivariate chain rule to the equation, with an explicit dependency on θ for completeness, yields:

$$\partial_\theta L_i(\mathbf{p}, \boldsymbol{\omega}, \theta) = \partial_\theta L_o(\mathbf{r}(\mathbf{p}, \boldsymbol{\omega}, \theta_0), -\boldsymbol{\omega}, \theta) + \partial_\theta L_o(\mathbf{r}(\mathbf{p}, \boldsymbol{\omega}, \theta_0), -\boldsymbol{\omega}, \theta_0) \partial_\theta \mathbf{r}(\mathbf{p}, \boldsymbol{\omega}, \theta), \quad (5.3)$$

where $\theta_0 = \theta$. Differential radiance is *not just* transported like normal radiance (first term) but differential radiance is also generated through the transport (second term). For example, consider an area light source with an emission texture, and let θ be the position of the light. At each point on the light source, the emission is spatially varying because of the texture ($\partial_{\mathbf{p}}L_o \neq 0$), and the intersection point of a ray with the light source can depend on θ ($\partial_{\theta}\mathbf{r} \neq 0$). Therefore, any path segment that terminates on the light could generate differential radiance. Importantly, transport between *any* surfaces, including non-emissive ones, could generate differential radiance:

Differential radiance is emitted through transport from and to geometry that moves with θ .

To see why, we differentiate the scattering equation (2.4) w.r.t. the position $\mathbf{p}$

$$\begin{aligned} \partial_{\mathbf{p}}L_o(\mathbf{p}, \omega) = & \underbrace{\partial_{\mathbf{p}}L_e(\mathbf{p}, \omega)}_{\text{Term A}} \\ & + \int_{S^2} \left[\underbrace{\partial_{\mathbf{p}}L_i(\mathbf{p}, \omega_i) f(\mathbf{p}, \omega, \omega_i)}_{\text{Term B}} \right. \\ & \left. + \underbrace{L_i(\mathbf{p}, \omega_i) \partial_{\mathbf{p}}f(\mathbf{p}, \omega, \omega_i)}_{\text{Term C}} \right] d\omega_i^{\perp}. \end{aligned} \quad (5.4)$$

Given that θ affects the intersection point, i.e., $\partial_{\theta}\mathbf{r} \neq \mathbf{0}$, the three terms in this equation have the following interpretation regarding the product $\partial_{\mathbf{p}}L_o\partial_{\theta}\mathbf{r}$ from Equation (5.3)

- **Term A:** Differential radiance is emitted by the transport from light sources if their emission is spatially varying (e.g., an area light with an emission texture).
- **Term B:** Differential radiance is emitted by the transport from surfaces if the *incoming radiance* L_i is spatially varying.
- **Term C:** Differential radiance is emitted by the transport from surfaces if their BSDF is spatially varying (e.g., a surface with an albedo texture).

Term B has a recursive nature, revealed by differentiating the transport equation, where $\mathbf{p}_0$ has the value of $\mathbf{p}$:

$$\partial_{\mathbf{p}}L_i(\mathbf{p}, \omega) = \partial_{\mathbf{p}}L_o(\mathbf{r}(\mathbf{p}_0, \omega), -\omega)\partial_{\mathbf{p}}\mathbf{r}(\mathbf{p}, \omega). \quad (5.5)$$

In practice, the incident radiance at *every* surface point will almost surely be spatially varying as radiance is reflected towards a point from various other surfaces: as soon as scene geometry moves with θ , differential radiance is generated through transport, regardless of whether there are spatially varying BSDFs or emission.

Therefore, differential transport as presented by Nimier-David et al. [2020] (Eq. (5.2)) only holds if the geometry is static.

5.1.3 Case Study

We observe that existing implementations of radiative backpropagation consider only Terms A and C and omit Term B. Consequently, the geometry derivatives will be biased,

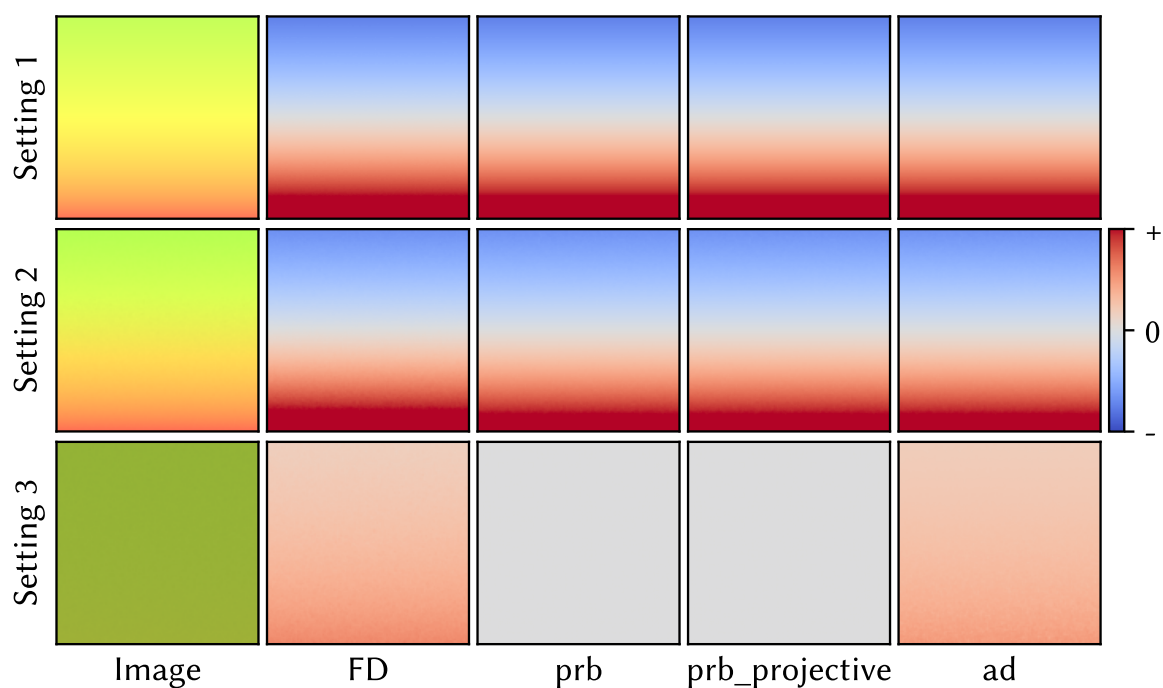
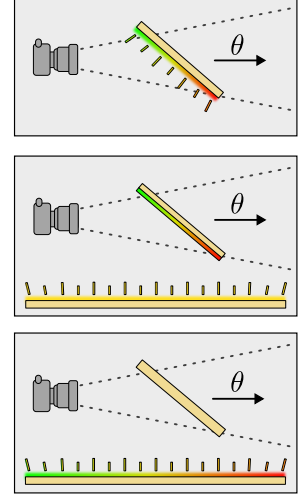


Figure 5.1: Three scenes with a backward-moving rectangle – spatially-varying emission (first row), spatially varying BSDF (second row), and indirect illumination (third row) – are rendered (first column). The derivatives of the red channel are displayed: finite differences (FD) (second column), PRB (third column), and the PRB variant of projective sampling (fourth column). All methods based on radiative backpropagation (prb*) as implemented in MITSUBA 3 (version 3.6.4) compute incorrect, zero derivatives for the last setting. Automatic differentiation (ad) yields correct derivatives (last column).

even for surfaces directly observed by a camera. We demonstrate this behavior for the different RB-based integrators in MITSUBA 3 [Jakob et al., 2022b] (version 3.6.4) using a set of simple examples. In the first setting (inset, top) the camera observes a slab with a spatially varying emission, which is moved away from the camera by θ . In the second setting (inset, middle) the camera observes the same slab, which now has a spatially varying diffuse albedo and is illuminated from below. In the third setting (inset, bottom), the moving slab has a constant albedo but is illuminated from below by another slab with spatially varying emission. Even though geometry is allowed to move, the integrand is free of discontinuities in the first setting, and discontinuities only marginally contribute to the derivative in the last two settings (the bottom slab is much larger than shown).



For the derivatives w.r.t. θ , one must account for Term A in the first setting and Term C in the second setting. Indeed, MITSUBA 3’s RB integrators compute these derivatives (Figure 5.1, rows 1 and 2). In the third setting, Term B has a noticeable contribution: the observed slab is illuminated by a spatially varying light, so the incident radiance changes when moving from one point on the surface to another ($\partial_p L_i \neq 0$ on the first slab). The RB integrators do *not* account for this (Figure 5.1, last row). This even affects the integrators designed for geometry optimization, such as `prb_projective`, which implements RB-style projective sampling [Zhang et al., 2023].

On a technical level, the reason why the implementation considers Terms A and C, even though the existing theory misses the transport-related differential radiance, is that the intersection point is attached, which means gradient tracking with respect to the intersected surface is enabled. Since the emission at the intersection and the BRDF are evaluated with an attached intersection point, the subsequent reverse accumulation automatically includes $\partial_p L_e \partial_\theta \mathbf{r}$ (Term A) and $L_i \partial_p f \partial_\theta \mathbf{r}$ (Term C). However, the ray defining the next path segment is detached from the intersection point, so changes in incident radiance are not included (Term B).

PRB and automatic differentiation The result obtained with path replay backpropagation (PRB) [Vicini et al., 2021] (`prb`), the most common variant of radiative backpropagation, should match automatic differentiation (`ad`, with detached sampling (Section 5.2.1)) by design, but it does not in the last setting (Fig. 5.1). The reason is that automatic differentiation includes Term B, which closely matches finite differences. This is not an issue of replaying paths but of the underlying theory for static geometry.

5.2 Non-Static Geometry and Detached Sampling

Radiative backpropagation is typically considered with *detached sampling*, i.e., the sampling process and probability densities in Monte Carlo estimators are not differentiated. Vicini et al. [2021] show that this leads to a simple, local approach to differentiation that does not need to consider variations of path geometry with the scene parameters.

However, the newly derived transport law (Section 5.1.2) reveals that the relevant derivatives are only local if the geometry is *static*, which significantly complicates potential implementations. Surprisingly, a truly local approach can be derived even for non-static geometry, but, as we will show in Section 5.2.2, this requires a theory for detached sampling of positions in the *three-point form*. This approach allows for a simple and efficient implementation using the path replay framework [Vicini et al., 2021].

We believe that the lack of a theory for the non-static case has led to current implementations mixing the different perspectives, without consistently following either one, leading to the behavior observed in Section 5.1.3.

In conclusion, we make two central observations that, to the best of our knowledge, are absent in the current literature:

1. Section 5.2.1: With non-static geometry, global path geometry must be considered for direction sampling *even with detached sampling*; Vicini et al. [2021] state that this is a unique to *attached* sampling, but this is only true if geometry is static.
2. Section 5.2.2: When taking the perspective of the three-point form, detached sampling makes the approach local, but one must correctly account for the (derivatives of the) transformation determinants and the intersection point.

5.2.1 Detached Sampling of Directions

Vicini et al. [2021] introduce the notion of “attached” and “detached” sampling for importance sampling of an integral

$$\int_{S^2} L(\mathbf{p}, \boldsymbol{\omega}, \theta) d\boldsymbol{\omega}, \quad (5.6)$$

which represents the scattering equation (2.4) with a spherical integration domain, i.e., L contains the emission L_e , the BSDF f , the incoming radiance L_i and the foreshortening cosine term. Importance sampling is cast as uniform sampling from the unit hypercube $\mathcal{U}$ with a change of variables

$$\int_{\mathcal{U}} \frac{L(\mathbf{p}, T(\mathbf{u}, \theta), \theta)}{p(T(\mathbf{u}, \theta), \theta)} d\mathbf{u}, \quad (5.7)$$

where $p(\boldsymbol{\omega}, \theta)$ is the target density from which the mapping $T(\mathbf{u}, \theta) = \boldsymbol{\omega}$ is constructed [Vicini et al., 2021]. Typically, both depend on the parameter θ (e.g., the roughness of a material).

Differentiating the integral yields an *attached* differential sampling strategy

$$\int_{\mathcal{U}} \partial_{\theta} \left[\frac{L(\mathbf{p}, T(\mathbf{u}, \theta), \theta)}{p(T(\mathbf{u}, \theta), \theta)} \right] d\mathbf{u} \quad (5.8)$$

that tracks how changes in θ affect the sampling process.

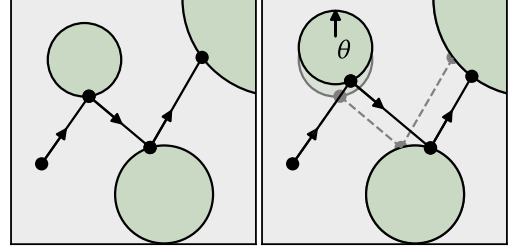
Let θ_0 be a parameter that matches the value of θ but does not participate in differentiation. A *detached* differential sampling strategy is

$$\int_{\mathcal{U}} \frac{\partial_{\theta} L(\mathbf{p}, T(\mathbf{u}, \theta_0), \theta)}{p(T(\mathbf{u}, \theta_0), \theta_0)} d\mathbf{u}. \quad (5.9)$$

Important and maybe surprising: both attached and detached strategies are valid estimators of the derivative as long as all points with $\partial_{\theta} L \neq 0$ are sampled with nonzero probability and the integrand is free of Dirac deltas [Zeltner et al., 2021].

Non-static geometry moves a path

The samples generated by the mapping T are *directions* and therefore, in detached sampling, $\omega_0 = T(\mathbf{u}, \theta_0)$ is a *direction* independent of θ . If directions along a path are sampled independent of any parameter, then the movement of intersected geometry with (another) parameter



θ can lead to the movement of successive path vertices: assume $\mathbf{p}$ is an intersection point on a surface that moves with θ . Since the sampled direction ω_0 is constant, the next intersection point $\mathbf{r}(\mathbf{p}, \omega_0)$ of a ray originating from $\mathbf{p}$ into direction ω_0 will depend on θ , because θ affects the ray origin in ambient space $S(\mathbf{p}, \theta)$. As the next intersection point depends on θ , the ones after will as well by recursion (see inset).

Computing the non-local term The connection between the derivative and global path geometry is captured by Term B in Eq. (5.4) and its recursion in Eq. (5.5), which, as shown in Section 5.1, is not accounted for by existing implementations. Strictly following the theory of detached direction sampling, one can recover the missing contribution. This leads to a construction very similar to attached sampling, where global path geometry is relevant, with the main difference being that the probability density and the sampled direction are not differentiated (see Appendix B, Section B.1).

5.2.2 Detached Sampling of Positions

The three-point form of the scattering equation [Veach, 1998] emerges when the integral over the spherical domain (Eq. (5.6)) is reparameterized as an integral over surface points in the scene

$$\int_{\mathcal{M}} L(\mathbf{p}, S(\mathbf{p}, \theta) \rightarrow S(\mathbf{p}', \theta), \theta) D(\mathbf{p}, \mathbf{p}', \theta) d\mathbf{p}'. \quad (5.10)$$

D represents the Jacobian determinant of the reparameterization

$$D(\mathbf{p}, \mathbf{p}', \theta) = G(\mathbf{p}, \mathbf{p}', \theta) |\partial_{\mathbf{p}_u} S(\mathbf{p}', \theta) \times \partial_{\mathbf{p}_v} S(\mathbf{p}', \theta)|, \quad (5.11)$$

with the geometry term

$$G(\mathbf{p}, \mathbf{p}', \theta) = V(\mathbf{p}, \mathbf{p}', \theta) \frac{|\mathbf{n}(\mathbf{p}', \theta)^\top (S(\mathbf{p}', \theta) \rightarrow S(\mathbf{p}, \theta))|}{\|S(\mathbf{p}, \theta) - S(\mathbf{p}', \theta)\|^2}, \quad (5.12)$$

where V is the binary visibility between the two points, $\mathbf{n}$ the surface normal at $\mathbf{p}'$ and $\mathbf{x} \rightarrow \mathbf{y}$ the direction from $\mathbf{x} \in \mathbb{R}^3$ to $\mathbf{y} \in \mathbb{R}^3$. Notice that this is the intrinsic form, over a global parameterization $\mathcal{M} \subset \mathbb{R}^2$, as discussed in Section 5.1.

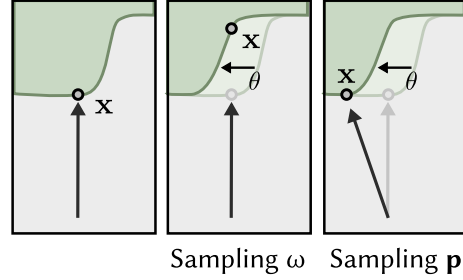
A detached sampling strategy for positions can be derived as in Section 5.2.1, by reparameterizing the integral over the unit cube

$$\int_{\mathcal{U}} \frac{L(\mathbf{p}, S(\mathbf{p}, \theta) \rightarrow S(T(\mathbf{u}, \theta), \theta), \theta) D(\mathbf{p}, T(\mathbf{u}, \theta), \theta)}{p(T(\mathbf{u}, \theta), \theta)} d\mathbf{u}, \quad (5.13)$$

differentiating and substituting θ_0 that does not participate in differentiation

$$\int_{\mathcal{U}} \frac{\partial_{\theta} [L(\mathbf{p}, S(\mathbf{p}, \theta) \rightarrow S(T(\mathbf{u}, \theta_0), \theta), \theta) D(\mathbf{p}, T(\mathbf{u}, \theta_0), \theta)]}{p(T(\mathbf{u}, \theta_0), \theta_0)} d\mathbf{u}. \quad (5.14)$$

Breaking the recursion of differential transport To see why this reparameterization makes the derivatives local, consider an ambient-space intersection point $\mathbf{x} = S(\mathbf{r}(\mathbf{p}, \omega, \theta), \theta)$ that may be computed as part of the (recursive) function L . Sampling a direction ω_0 as in Section 5.2.1, yields an intersection point $\mathbf{x} = S(\mathbf{r}(\mathbf{p}, \omega_0, \theta), \theta)$ that slides along the surface when the intersected object or the ray origin moves (inset middle). On the other hand, sampling a static point $\mathbf{p}_0 = T(\mathbf{u}, \theta_0)$ in the parameterization $\mathcal{M}$, yields an intersection point in ambient space



$$\mathbf{x} = S(\mathbf{r}(\mathbf{p}, S(\mathbf{p}, \theta) \rightarrow S(\mathbf{p}_0, \theta), \theta), \theta) = S(\mathbf{p}_0, \theta) \quad (5.15)$$

that moves *with* the shape (inset right). Since the parameter position of the intersection point is independent of θ , the differential transport equation (5.3) becomes

$$\begin{aligned} \partial_{\theta} L_i(\mathbf{p}, S(\mathbf{p}, \theta) \rightarrow S(\mathbf{p}_0, \theta), \theta) = \\ \partial_{\theta} L_o(\mathbf{p}_0, S(\mathbf{p}_0, \theta_0) \rightarrow S(\mathbf{p}, \theta_0), \theta) \\ + \partial_{\omega} L_o(\mathbf{p}_0, S(\mathbf{p}_0, \theta_0) \rightarrow S(\mathbf{p}, \theta_0), \theta_0) \partial_{\theta} [S(\mathbf{p}_0, \theta) \rightarrow S(\mathbf{p}, \theta)]. \end{aligned} \quad (5.16)$$

The differential $\partial_{\omega} L_o$ is local because only the emission and the BRDF value at the intersection point $\mathbf{p}_0$ vary with the direction ω , so, differential transport considers only local path geometry.

Contribution of individual samples While it may seem counterintuitive, the derivatives account for the spatially varying albedo and emission in the settings from Section 5.1.3, despite samples being *fixed* locations on the surface. Take, for example, Setting 2, where the camera observes an object with spatially varying albedo moving with θ : when sampling directions, the derivative of a single sample w.r.t. θ captures the changing albedo, e.g., from red to green, through the product of $L_i \partial_p f$ (Term C in $\partial_p L_o$, Eq. (5.4)) and $\partial_{\theta} \mathbf{r}$ (Eq. (5.3)). When sampling positions, the albedo at a single sample is constant,

but its derivative w.r.t. θ considers the *weighting* of it through the product of $L_i \partial_\omega f$ (part of $\partial_\omega L_o$ from Eq. (5.16)) and $\partial_\theta [S(\mathbf{p}_0, \theta) \rightarrow S(\mathbf{p}, \theta)]$, as well as through the derivative of the Jacobian determinant $\partial_\theta D$. The combination of all sample derivatives then yields a valid estimate of the differential integral.

Notice that this also leads to different noise patterns and variance, making equal sample counts not necessarily comparable.

Importance sampling and Jacobian determinant The derivative of the Jacobian determinant is easily missed when constructing a path using BSDF importance sampling. This is because the PDF for positions p is typically derived from a PDF for directions p_ω

$$p(\dots) = p_\omega(\dots) D(\dots), \quad (5.17)$$

by warping it with the determinant D . It may seem that the determinants cancel when plugging the PDF into Eq. (5.14) but this is not the case: the determinant in the numerator depends on θ , while the determinant in the denominator is part of the PDF and therefore independent of θ :

$$\frac{D(\mathbf{p}, T(\mathbf{u}, \theta_0), \theta)}{D(\mathbf{p}, T(\mathbf{u}, \theta_0), \theta_0)}. \quad (5.18)$$

The fraction always evaluates to 1, but its derivative w.r.t. θ is (generally) not 0. This is also noted by Zhang et al. [2020].

Pseudocode for path replay backpropagation The most common variant of radiative backpropagation is path replay backpropagation (PRB) [Vicini et al., 2021]: it “replays” light paths from the forward simulation during the simulation of differential radiance, or, in the case of gradients, its measurement with adjoint radiance.

In Listing 1 we provide pseudocode for detached path replay backpropagation in the three-point form, following the style of the listings by Vicini et al. [2021]. The function `sample_path` computes the radiance L measured by a path that originates from the camera at position $\mathbf{p}_{\text{cam}}$ into direction ω (this direction is derived from a sample on the sensor). The radiance L and the adjoint radiance δL (the derivative of a loss function w.r.t. L) are then passed to `sample_path_adjoint` to retrieve the derivative of the loss with respect to θ . The functions are defined as follows:

- `intersect` returns the intersection point of a ray with the scene as a point in the surface parametrization $\mathcal{M}$
- `to_ambient` transforms a point from the parametrization to ambient space $\mathbb{R}^3$, considering the scene parameters
- `detach` disables derivative tracking for the given variable
- `reparam_det` returns the value of the reparameterization determinant D
- `sample_bsdf` samples a direction from the current BSDF and also returns the PDF value of the sample

- `eval_bsdf(p, ωo, ωi)` computes the BSDF value $f(\mathbf{p}, \omega_o, \omega_i)$
- `backward_grad(f,  $\frac{\partial g}{\partial f}$ )` uses reverse-mode differentiation to evaluate $\frac{\partial f}{\partial \theta} \frac{\partial g}{\partial f}$

The code has two notable differences from the original, which uses detached sampling of directions.

First, each sample $T(\mathbf{u}_i, \theta_0)$ is a surface point $\mathbf{p}_i$ in the 2D parameter space $\mathcal{M}$, *independent* of θ . For BSDF importance sampling, this means that one first samples a direction (`sample_bsdf(...)`), then intersects the next surface but does not differentiate the operation (`detach(intersect(...))`). The surface point is transformed back into the ambient space $\mathbb{R}^3$ using the mapping $S(\mathbf{p}, \theta)$ (`to_ambient(...)`); this transformation is differentiated because it potentially depends on θ (the sample position in $\mathbb{R}^3$ follows if θ moves the corresponding shape).

Second, the Jacobian determinant D from the reparameterization must be differentiated as it might depend on θ . The BSDF importance sampling PDF contains the factor $\frac{1}{D}$, but it is detached, so the derivative of their product is generally non-zero (c.f. Eq. (5.18)). By the product rule, the derivative $\partial_\theta D$ must be multiplied by L and, for reverse-mode differentiation, also by the adjoint radiance δL . This leads to another gradient contribution. In `sample_path` the product of D and $\frac{1}{D}$ simply cancels, so it can be omitted.

5.3 Implementation and Experiments

We have implemented detached sampling in the three-point form (Section 5.2.2) in `MITSUBA 3` [Jakob et al., 2022b] as an automatic differentiation variant (`ad_threepoint`) and a path replay variant (`prb_threepoint`). We have also implemented detached sampling of directions (Section 5.2.1), but only an automatic differentiation variant (`ad`). All integrators use multiple importance sampling, combining BSDF and emitter samples.

5.3.1 Case Study – Revisited

We first revisit the case study from Section 5.1.3 and test our integrators in the three settings (Fig. 5.2). In contrast to existing implementations (recall Fig. 5.1), the integrators based on the three-point form produce non-zero derivatives in the last setting, which not only agree with the finite difference reference, but PRB and AD match exactly, consistent with the theory. Both, direction and point sampling produce similar results that differ, for example, by the noise pattern. One reason is that individual samples contribute differently to the derivative in both forms, so equal sample counts are not necessarily comparable (see Section 5.2.2, Paragraph Contribution of individual samples).

The unbiased solution (`prb_threepoint`) has an overhead over the biased solution (`prb`), which, however, is expected: the AD graph of a single loop iteration is more complex as the previous and next surface intersections are now considered.

5.3.2 Unique Discontinuities

We have so far assumed that the integrand is continuous, but non-static geometry can lead to integrands with parametric discontinuities that move with θ . What constitutes a

Listing 1 Pseudocode of detached PRB in three-point form. We omit emitter sampling, which is straightforward to add using multiple importance sampling. No derivatives are tracked across loop iterations. The used functions are defined in Section 5.2.2 and a theoretical derivation of the code can be found in Appendix B (Section B.2).

```

1  def sample_path(p_cam,  $\omega$ ):
2       $L = 0$ ,  $\beta = 1$ 
3       $p_p = p_{cam}$ 
4       $p_c = \text{intersect}(\text{to\_ambient}(p_{cam}), \omega)$ 
5      for i in range(1, N+1):
6           $x_p = \text{to\_ambient}(p_p)$ 
7           $x_c = \text{to\_ambient}(p_c)$ 
8
9           $L += \beta * L_e(p_c, x_c \rightarrow x_p)$ 
10
11          $\omega'$ , bsdf_value, bsdf_pdf = sample_bsdf(...)
12         bsdf_value = eval_bsdf( $p_c, x_c \rightarrow x_p, \omega'$ )
13          $\beta *= \text{bsdf\_value} / \text{bsdf\_pdf}$ 
14          $p_p = p_c$ 
15          $p_c = \text{intersect}(x_c, \omega')$ 
16     return L

```

```

1  def sample_path_adjoint(p_cam,  $\omega$ , L,  $\delta L$ ):
2       $\beta = 1$ 
3       $p_p = p_{cam}$ 
4       $p_c = \text{detach}(\text{intersect}(\text{to\_ambient}(p_{cam}), \omega))$ 
5      for i in range(1, N+1):
6           $x_p = \text{to\_ambient}(p_p)$ 
7           $x_c = \text{to\_ambient}(p_c)$ 
8
9          # Derivative of  $L_e$ 
10          $L_e = \beta * L_e(p_c, x_c \rightarrow x_p)$ 
11          $\delta_\theta += \text{backward\_grad}(L_e, \delta L)$ 
12
13         # Derivative of Jacobian determinant
14          $D = \text{reparam\_det}(p_p, p_c)$ 
15          $\delta_\theta += \text{backward\_grad}(D, \delta L * L / D)$ 
16
17          $L -= L_e$ 
18
19         # Derivative of BSDF
20          $\omega'$ , bsdf_pdf = sample_bsdf(...)
21          $p_n = \text{detach}(\text{intersect}(x_c, \omega'))$ 
22          $x_n = \text{to\_ambient}(p_n)$ 
23         bsdf_value = eval_bsdf( $p_c, x_c \rightarrow x_p, x_c \rightarrow x_n$ )
24          $\delta_\theta += \text{backward\_grad}(\text{bsdf\_value},$ 
25                                $\delta L * L / \text{bsdf\_value})$ 
26
27          $\beta *= \text{bsdf\_value} / \text{bsdf\_pdf}$ 
28          $p_p = p_c$ 
29          $p_c = p_n$ 
30     return  $\delta_\theta$ 

```

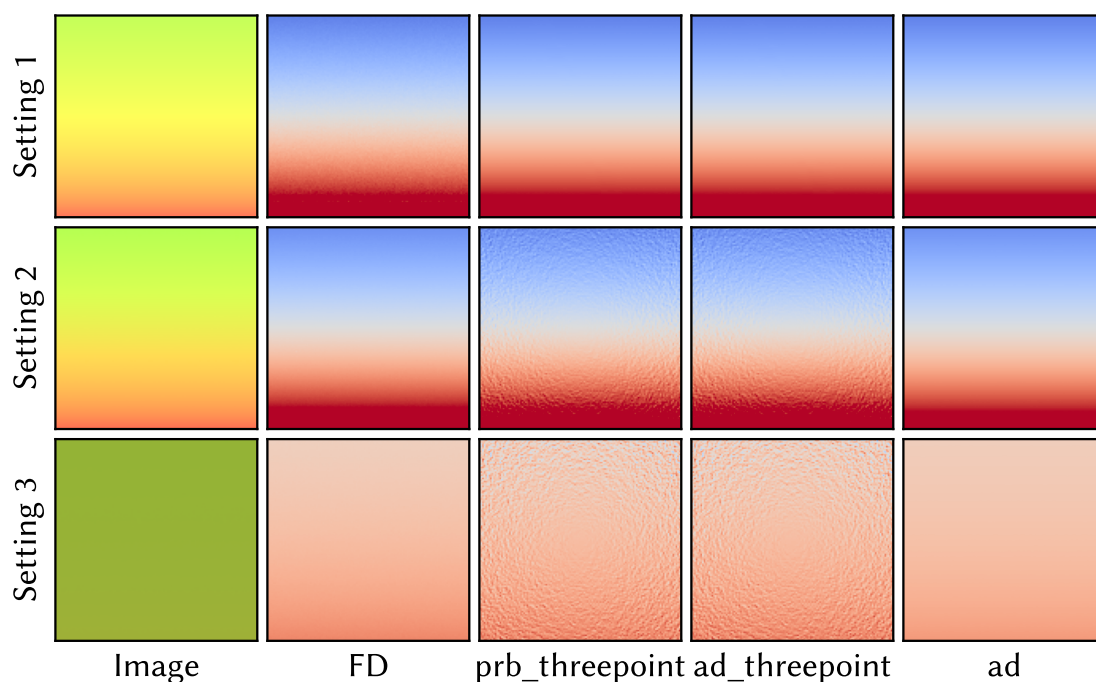


Figure 5.2: Three scenes with a backward-moving rectangle – spatially-varying emission (first row), spatially varying BSDF (second row), and indirect illumination (third row) – are rendered (first column). The derivatives of the red channel are displayed: finite differences (FD) (second column), PRB using the three-point form (third column), automatic differentiation (AD) using the three-point form (fourth column), and automatic differentiation (AD) using the spherical form (last column). The derivatives of the three-point form integrators are close to the FD reference, but they differ from the spherical form because: (1) the estimates have different noise patterns, and (2) the scenes have discontinuities in the three-point form, which leads to a small bias.

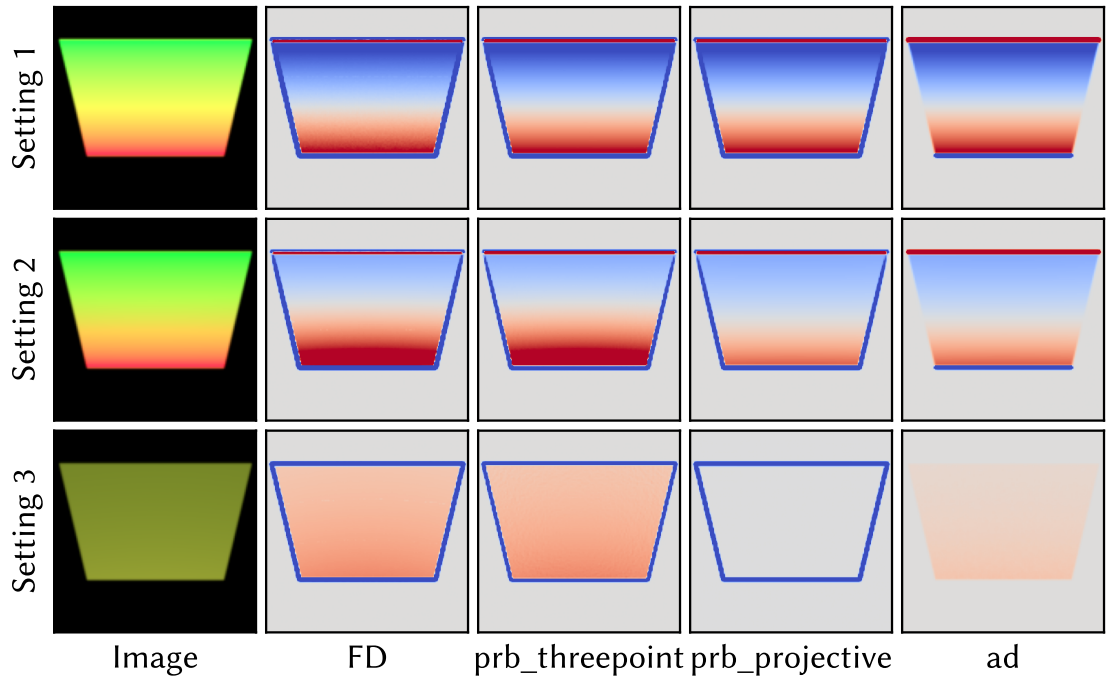


Figure 5.3: Three scenes with a backward-moving scaled rectangle – spatially-varying emission (first row), spatially varying BSDF (second row), and indirect illumination (third row) – are rendered (first column). The derivatives of the red channel are displayed: finite differences (FD) (second column), PRB integrator using three-point formulation (third column), the projective sampling variant of PRB (fourth column), and automatic differentiation (AD) using spherical formulation (last column). Only the integrator based on the three-point form computes the correct derivatives.

“discontinuity” depends solely on the integration domain: when integrating over directions, the radiance collected along a sampled direction can change discontinuously as an object moves into or out of the ray. When integrating over surface points, the collected radiance can change discontinuously as the *visibility* of a surface point changes. The two perspectives are not generally interchangeable, and a scene with no discontinuities in one form may have discontinuities in the other. For example, we initially suggested that the first case study scene with a moving slab covering the camera’s field of view was free of discontinuities (Section 5.1.3), but this is only true for the spherical integral over directions. In fact, all scenes from Section 5.1.3 have discontinuities in the three-point form, which leads to bias in the results shown in Fig. 5.2.

We have modified all scenes to be free of discontinuities in the three-point form, which is achieved by reducing the size of the slab and the emitter, making the slab fully visible to the camera and the emitter fully visible to each point on the slab. In these settings, the three-point form yields unbiased derivatives (Fig. 5.3). However, Settings 2 and 3 now have discontinuities when sampling the (hemi)sphere, which noticeably affects the result of the AD integrator (ad) based on detached sampling of directions (Section 5.2.1). The projective sampling integrator is susceptible to the same discontinuities (see Setting 2).

5.3.3 Radiative Backpropagation for Interior Path Derivatives

Although we have completely ignored discontinuities in the derivations so far, it is straightforward to combine our approach with discontinuity handling using the framework of path-space differentiable rendering. Zhang et al. [2020] show that the derivatives of the path-space integral, i.e., the integral over the product spaces of surface integrals, can be expressed as the sum of an *interior* term and a *boundary* term. Our approach for detached path replay backpropagation in the three-point form (Section 5.2.2) computes an unbiased estimate of the interior term in the so-called *material-form parameterization*, so combining it with discontinuity handling is as simple as choosing a suitable method for the boundary term and adding the result to our estimate. Previous implementations attempting to combine the boundary term with radiative backpropagation compute an *incorrect* interior term, based on direction sampling, which manifests as severe bias with inverted derivative signs (Fig. 5.4).

A simple application Despite the presence of discontinuities, the interior component is central for convergence, even in the simplest inverse rendering applications: we test implementations on a monocular pose estimation task, where the orientation and position of an object are to be inferred from a single image. The objective is to minimize the mean absolute error between a target image and the rendering. Current implementations combining a radiative backpropagation-style interior term with the boundary term (here estimated with projective sampling [Zhang et al., 2023]) often fail to converge in the given time, whereas our three-point approach converges successfully, even without explicit discontinuity handling (Fig. 5.5).

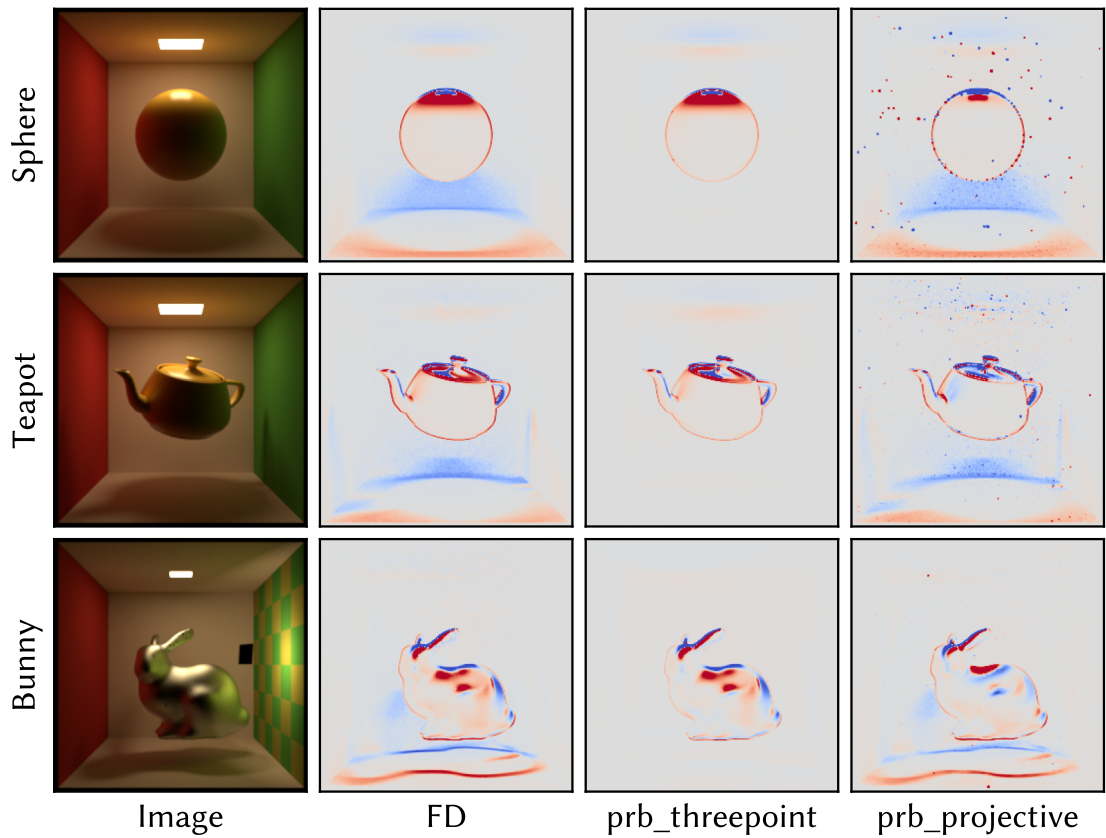


Figure 5.4: Implementations of path-space differentiable rendering combine interior derivatives computed with radiative backpropagation (RB) and boundary derivatives (prb_projective). Their interior derivatives do not match the finite difference reference (FD). An RB implementation of the surface integral (Section 5.2.2) produces the correct interior derivatives (prb_threepoint).



Figure 5.5: A simple inverse rendering task – recovering the 6D pose (position and orientation) of the teapot from a single image – fails if the interior derivatives are incorrect, even if discontinuities are handled (`prb_projective`). A radiative backpropagation-based implementation of the surface integral (Section 5.2.2) converges each time, despite not explicitly handling discontinuities (`prb_threepoint`).

Combining boundary and RB interior derivatives As a proof-of-concept, we have implemented a MITSUBA 3 integrator that combines projective sampling [Zhang et al., 2023] with our RB interior derivatives (Fig. 5.6). While the combination of derivatives matches the finite difference reference more faithfully, the result is still slightly biased. A decomposition into the interior and boundary contributions reveals that a systematic bias seems to originate from the boundary term, estimated with the official implementation of projective sampling in MITSUBA 3 (see the area around the teapot lid knob in the TWIST scene in Fig. 5.6). It is unclear, whether this is a conceptual issue or related to the implementation. However, since the boundary term is unrelated to the findings in this chapter and beyond its scope, we defer further investigation to future work.

5.4 Conclusion

The original derivation of radiative backpropagation assumed static geometry with spherical integrals. We have complemented the theory by deriving the differential radiance equations for non-static geometry and different parameterizations of the rendering integral. A key observation is that differential radiance is not only transported like normal radiance, but differential radiance is also emitted from moving geometry. Depending on the parameterization of the rendering integral, this emission term becomes non-local, so where previously derivatives could simply be computed from local path geometry, the motion of the entire light path must now be taken into account. We have shown that the derivatives become local when the rendering integral is reparameterized over surfaces, leading to a practical implementation for non-static geometry.

Impact We have shown that the `prb` and `prb_projective` integrators in MITSUBA 3 (up to and including version 3.6.4) are biased if geometry is non-static. The deprecated reparameterization integrator `prb_reparam` is very likely to be affected as well because it builds on the findings by Zeltner et al. [2021]: they arrive at a local formulation of derivatives, which does not correctly account for differential transport. Results that rely on the affected integrators (e.g., for academic publications) should likely be revalidated.

Physically-based renderers without radiative backpropagation, such as REDNER [Li et al., 2018b] (based on work by Li et al. [Li et al., 2018a]) or PSDR-JIT [Yan et al., 2020], are unaffected by this bias. While our implementation of radiative backpropagation is based on MITSUBA 3, we believe it also provides a useful reference point for other frameworks.

In Chapter 6, we extend path replay backpropagation to differentiable time-resolved rendering. Since path geometry determines the transport time of energy, the time derivatives must correctly account for perturbations caused by non-static surfaces. This is accomplished using PRB in the three-point form, as presented in Section 5.2.2.

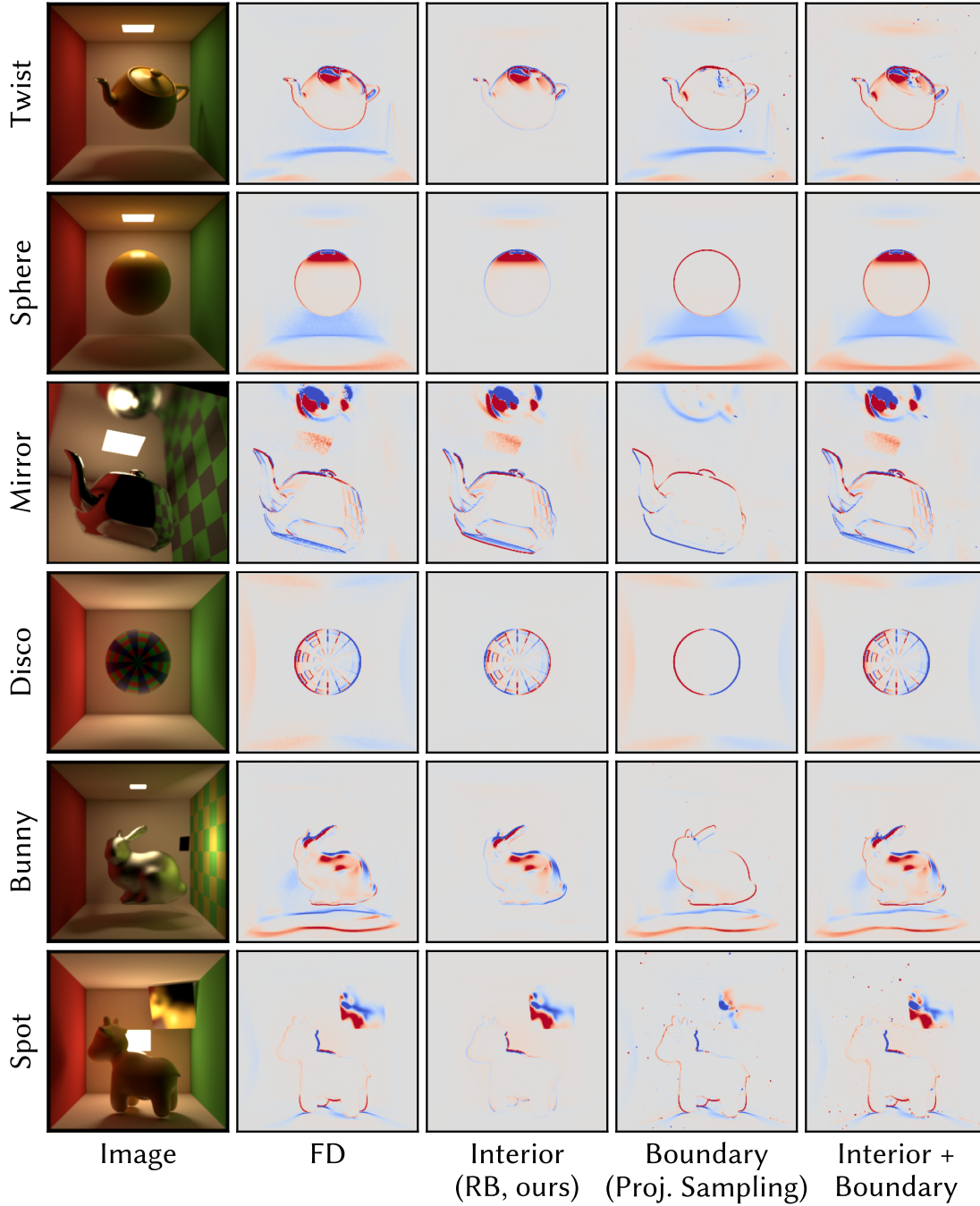


Figure 5.6: For path-space differentiable rendering, the interior component of the derivative can be computed with radiative backpropagation (RB) in the surface integral form (Section 5.2.2), and the boundary component with projective sampling [Zhang et al., 2023]. We combine both in a proof-of-concept implementation (“Interior + Boundary”). While this implementation matches the finite difference (FD) reference more closely than existing ones (compare Fig. 5.4), the boundary component still seems to lead to bias (e.g., the area around the teapot lid knob of Twist). The shown derivatives are with respect to a translation of the center object.

CHAPTER 6

Time-Resolved Path Replay Backpropagation

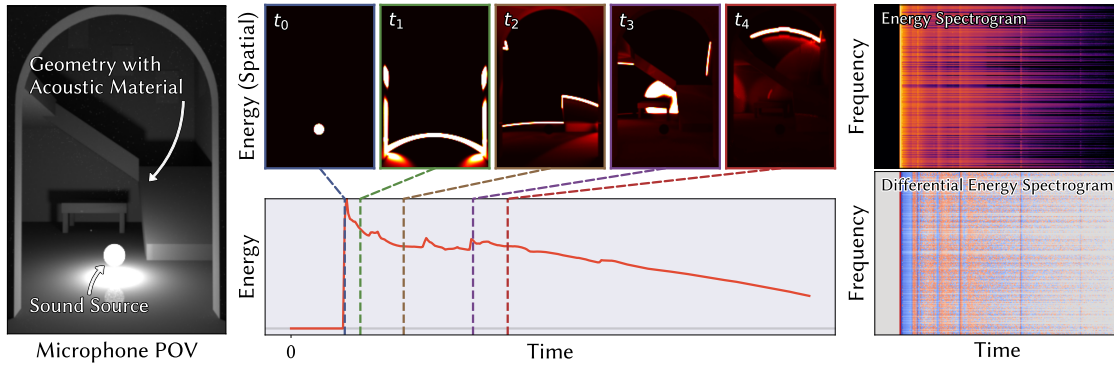


Figure 6.1: Consider the scene on the left with a single spherical emitter. In the acoustic setting, the emitter radiates sound energy, whose time-dependent transport can be simulated using path-tracing techniques in accordance with geometric acoustics. In contrast to light simulation, the energy at a single frequency is integrated over the entire spatial domain but resolved in time, resulting in an energy time curve (ETC) (bottom center). The resulting energy spectrogram (top right) shows the ETC for each frequency as a heat map. We take advantage of the analogy between light and sound and extend efficient differentiable physically-based rendering to the time domain, allowing us to compute differential ETCs (bottom right). The differential ETCs show the effect of translating the sound source towards the microphone.

A central but subtle assumption underlying the previous chapters has been that of a steady state: the distribution of energy in the scene has reached an equilibrium and does not change over time. Hence, energy transport can be assumed to happen instantaneously, such that simulations may safely ignore the finite speed of light. In practice, however, transport time cannot be disregarded in all settings, for example, for the simulation of time-of-flight cameras. This leads to so-called *transient* rendering [Smith et al., 2008], where energy is not only resolved spatially over a sensor but also in time.

Similar to the steady-state case, there has been a recent surge in methods that use differentiable rendering for solving inverse problems in transient rendering [Plack et al., 2023; Choi et al., 2023; Malik et al., 2025]. While many differentiable renderers are special-purpose, for example, with a three-bounce assumption [Plack et al., 2023; Choi et al., 2023], others have adopted physically-based differentiable rendering in the tran-

sient setting [Yi et al., 2021; Wu et al., 2021]. These physically-based renderers are general and correctly handle discontinuities in transient path integrals, but efficiency is lacking: the systems by Yi et al. [2021] and Wu et al. [2021] use (forward-mode) automatic differentiation on the CPU, which is only suitable for a few parameters and simple applications. Yet, the practical implications of this limitation are unclear since (1) efficient, specialized renderers seem to suffice for many applications, and (2) inverse transient rendering remains relatively niche, measured by the size of the field.

If we look beyond graphics and the simulation of light, we find other areas that would greatly benefit from general and efficient tools for time-resolved differentiable rendering. One of those is *room acoustics*, a subfield of acoustics that is concerned with sound propagation in (partially enclosed) rooms. Since sound travels much slower than light, numerical simulations in (room) acoustics naturally consider the propagation time. On a technical level, these simulations are closely related to physically-based rendering in graphics: sound is often assumed to propagate in straight lines, following the principles of *geometric acoustics*, while ignoring wave phenomena like diffraction or interference. Hence, among other approaches, sound propagation in room acoustics is simulated by solving a (time-dependent) rendering equation with path tracing [Siltanen et al., 2007; Kiminki, 2005; Schröder, 2011; Cao et al., 2016] (Section 6.1). Despite the limitations, geometric acoustics methods are widely used in room acoustics because they offer a computationally efficient, high-frequency approximation. In contrast, accurate wave-based simulations are prohibitively expensive in memory and computation time when simulating high frequencies in extended rooms.

As for the case of light, not only is the forward simulation of interest, but also the solution to acoustic optimization problems. For example, a typical task in room acoustics is the optimization of speaker placement and panel geometry in concert halls to meet certain levels of listener clarity. Surprisingly, it is still common practice to optimize model parameters manually [Labia et al., 2020] or use time-consuming genetic algorithms [Christensen et al., 2014; Pilch, 2020]. Although differentiable acoustic rendering has recently appeared in the literature [Zhi et al., 2023; Stanziola et al., 2023], to our knowledge, there is no general approach for differentiable acoustic path tracing.

In this chapter, we propose methods for efficient and fully-differentiable acoustic path tracing. This is made possible by exploiting the analogy between light and sound: we show that many of the techniques developed for (steady-state) physically-based differentiable rendering can be adapted for differentiable acoustic simulations. Our central contribution is *time-resolved path replay backpropagation* (TPRB) an extension of path replay backpropagation [Vicini et al., 2021] (Section 2.4.3) for time-resolved differentiable rendering. Similar to its steady-state counterpart, it is based on the principles of radiative backpropagation [Nimier-David et al., 2020], and allows for efficient, reverse-mode differentiation with constant memory and linear time in path length. The decoupling of storage is essential when optimizing hundreds to millions of parameters because the almost mirror-like reflectivity of many surfaces typically requires acoustic simulations with very long paths. Using existing, automatic differentiation-based systems from transient rendering [Yi et al., 2021; Wu et al., 2021] would be infeasible in this setting. The core of our time-resolved extension to path replay backpropagation is correctly accounting for variations of propagation time with scene geometry. We not only extend

the theory of radiative backpropagation in that regard, but also use our insights from Chapter 5 to account for the relevant derivatives in differential transport. Surprisingly, despite handling (discretized) functions over time, the required storage can also be decoupled from the time resolution such that the algorithm has only minimal additional overhead over the steady-state variant.

The resulting system is capable of simulating sound according to the geometric acoustics model and efficiently computing derivatives of the output energy spectrograms with respect to arbitrary parameters of the scene, including materials, emitters, microphones, and scene geometry. We believe that this opens up a new direction for acoustic optimization that could be as impactful for the acoustic community as differentiable rendering was for the graphics and vision communities.

6.1 Background: Acoustic Path Tracing

The rendering equation for light transport [Kajiya, 1986] (Section 2.1.2) provides the foundation for sound simulation in geometric acoustics, in particular, its temporal extension [Siltanen et al., 2007]. The acoustic rendering equation is derived from the analogy between visual and acoustic concepts: camera sensors to microphones, light sources to speakers, and images to energy spectrograms. Unsurprisingly, the equation has a form similar to the rendering equation for light, only that the energy varies with time t :

$$L_o(\mathbf{x}, \boldsymbol{\omega}, t) = L_e(\mathbf{x}, \boldsymbol{\omega}, t) + \int_{S^2} f(\mathbf{x}, \boldsymbol{\omega}, \boldsymbol{\omega}_i) L_i(\mathbf{x}, \boldsymbol{\omega}_i, t) d\boldsymbol{\omega}_i^\perp. \quad (6.1)$$

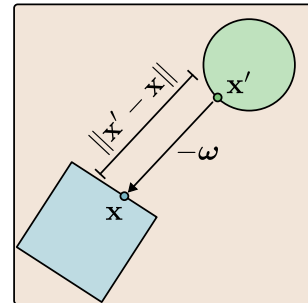
L_o is the outgoing energy (radiance, for light) at position $\mathbf{x}$ in direction $\boldsymbol{\omega}$, L_e and L_i refer to the emitted and incident energy, f is the bidirectional scattering distribution function (BSDF, see Appendix C.2), and the integral is over the sphere S^2 , with $\boldsymbol{\omega}_i^\perp$ being the projected solid angle. Here, the highlights mark the (minor) differences to the equations for light transport.

The incoming and outgoing energy are related by

$$L_i(\mathbf{x}, \boldsymbol{\omega}, t) = L_o(\mathbf{x}'(\mathbf{x}, \boldsymbol{\omega}), -\boldsymbol{\omega}, t_r(\mathbf{x}, \boldsymbol{\omega}, t)), \quad (6.2)$$

where $\mathbf{x}'(\mathbf{x}, \boldsymbol{\omega})$ is the intersection point of the ray starting at $\mathbf{x}$ along direction $\boldsymbol{\omega}$ and t_r is the time that compensates for the propagation speed of energy (see inset), commonly referred to as *retarded time*. If the propagation speed is c (in our case, the speed of sound), then the time needed for energy to travel between two surfaces (*local delay*) is given by

$$\tau = \frac{\|\mathbf{x}'(\mathbf{x}, \boldsymbol{\omega}) - \mathbf{x}\|}{c}. \quad (6.3)$$



Therefore, the energy incoming at the blue surface at time t is simply the energy emitted from the green surface at time

$$t_r(\mathbf{x}, \boldsymbol{\omega}, t) = t - \tau. \quad (6.4)$$

Sensors and Measurement In acoustics simulations, the sensors are typically (omni)-directional *microphones* and the results are stored in *energy spectrograms*, which hold the energy at a time t (x-dimension) at certain frequencies (y-dimension) (see Fig. 6.1). Therefore, the output is two-dimensional, like images in light transport. However, the rows of the energy spectrogram are *independent* because frequencies are independent in linear systems. For simplicity, without loss of generality, we consider a single row of an energy spectrogram, referred to as *energy histogram*. An energy histogram captures the energy received at a microphone at position $\mathbf{m}$ as a function of time

$$H(t) = \int_{S^2} W_e(\omega_i) L_i(\mathbf{m}, \omega_i, t) d\omega_i, \quad (6.5)$$

where $W_e(\omega_i)$ is a weighting function expressing the directional sensitivity of the microphone. This resembles the light transport measurement equation, except that the measurement is comparable to a single “pixel” with a time-dependent value. The weighting allows modeling different directivity patterns, such as Subcardioid, Cardioid, Hypercardioid, or Figure of Eight. For brevity, we assume omnidirectionality in Section 6.2, i.e., $W_e(\omega_i) = 1$. The applications in Section 6.4 show experiments with directional microphones, where the weights are given by a von Mises-Fisher distribution.

6.2 Method

Going forward, we derive an efficient algorithm for differentiating acoustics simulations based on the time-dependent rendering equation introduced in Section 6.1. Consequently, these simulations may be used as part of gradient-based optimization of arbitrary objective functions. The derivation is comprised of four main parts:

1. Section 6.2.1: We establish a functional perspective that casts energy transport in acoustics as transport of time-dependent *functions*; this allows treating acoustics simulations similar to scalar energy transport in light rendering.
2. Section 6.2.2: We introduce the realization of differentiable rendering in the acoustic setting and show that the functional perspective naturally preserves some properties of differentiable rendering for light.
3. Section 6.2.3: We differentiate the functional transport equation which leads to additional time-dependent terms unique to non-instantaneous energy transfer; we show that the derivatives can be computed by replaying transport paths.
4. Section 6.2.4: We point out that a naive implementation based on the functional perspective has storage complexity linear in time resolution. We show that, surprisingly, backward-mode differentiation can be performed with constant storage complexity, independent of time resolution.

6.2.1 A Functional Perspective of Rendering Integrals

The rendering equation based on *retarded time* (Eq. (6.1)) makes it difficult to reason about a *single* point in time across bounces or paths because the evaluation time t is directly influenced by the transport equation. However, the rendering equation can be equivalently rewritten in terms of *advanced time*

$$L_o(\mathbf{x}, \boldsymbol{\omega}, T, t) = L_e(\mathbf{x}, \boldsymbol{\omega}, T, t) + \int_{S^2} f(\mathbf{x}, \boldsymbol{\omega}, \boldsymbol{\omega}_i) L_i(\mathbf{x}, \boldsymbol{\omega}_i, T, t) d\boldsymbol{\omega}_i^\perp, \quad (6.6)$$

where the incoming and outgoing energy are related through

$$L_i(\mathbf{x}, \boldsymbol{\omega}, T, t) = L_o(\mathbf{x}', -\boldsymbol{\omega}, T + \tau, t), \quad (6.7)$$

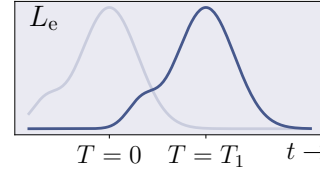
and the *delay* T accumulates the propagation time along a path.

The effect of the travel time is postponed to the evaluation of the emission, which is related to the emission term for retarded time by

$$L_e(\mathbf{x}, \boldsymbol{\omega}, T, t) := L_e(\mathbf{x}, \boldsymbol{\omega}, t - T), \quad (6.8)$$

(for advanced time) (for retarded time)

such that a value of t globally identifies a single point in time. In particular, it is now apparent that the collected emission is simply the emission profile L_e delayed to T (see inset). We emphasize that L_e is a *function*¹ of time t ; whereas light path tracing treats it as a single value, assuming monochromatic outputs. In this framework, the integrands in the rendering equation are functions of t , and the resulting histogram is simply an integral over these functions². The functional perspective is appealing because it allows reasoning about time-dependent Monte Carlo integration and algorithms for differentiable path tracing in the same way as if the collected energy were a scalar.



In particular, the corresponding measurement equation

$$H(t) = \int_{S^2} L_i(\mathbf{m}, \boldsymbol{\omega}_i, 0, t) d\boldsymbol{\omega}_i. \quad (6.9)$$

can be solved with standard Monte Carlo integration techniques: draw samples $s_i = (\boldsymbol{\omega}_{i,0}, \boldsymbol{\omega}_{i,1}, \dots)$, where each sample defines a path using a set of directions, and then add the individual energy functions. Since functions over t form a vector space, all operations, like addition or scalar multiplication, are defined (Figure 6.2).

Going forward, we adopt the continuous functional perspective and delay remarks on the discretization to Section 6.2.4. Notably, we will often omit the dependence on the parameter t for clarity, which, for example, means using the shorthand notation $L_e(\mathbf{x}, \boldsymbol{\omega}, T)$ or even $L_e(T)$ for $L_e(\mathbf{x}, \boldsymbol{\omega}, T, t)$ as “the emission function delayed by T ”.

¹To render an impulse response, one would usually use a delta-distribution. Our generalization to arbitrary emission profiles will become relevant in Section 6.2.4

²Mathematically, this resembles working with spectral color representations that use functions of wavelength λ , except that emission profiles in the time domain are shifted depending on the path length. Semantically, however, emission spectra in wavelength/frequency space are fundamentally distinct from emission profiles in the time domain.

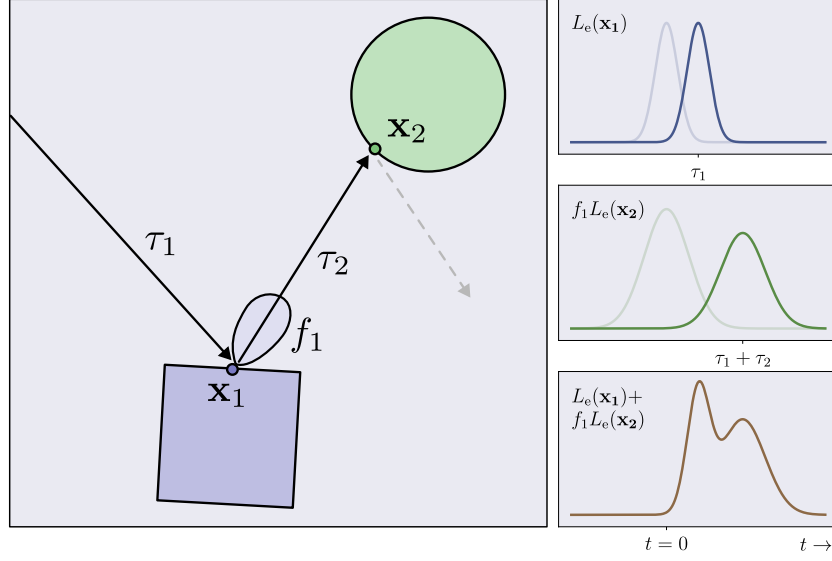


Figure 6.2: Time-dependent energy accumulated on a sub-path through the scene. The energy functions of the respective emitters $L_e(\mathbf{x}_1, t)$ and $L_e(\mathbf{x}_2, t)$ are delayed by the time traveled along the ray and simply combined in a sum, weighted by the BSDF.

6.2.2 Differentiable Rendering in Acoustics

For differentiable rendering of light (Section 2.4), the rendering operator $R(\theta)$ generates an image. In the acoustic setting, it generates a time-dependent energy histogram $R(\theta) = H(\theta, t)$. An objective function in the acoustic setting then is a *functional*

$$\mathcal{L}[H], \quad (6.10)$$

which takes the histogram and returns a scalar. For example, in the case of light, it is typical to measure the error between a reference image and the rendered image; the equivalent functional, comparing a reference histogram $H_{\text{ref}}(t)$ to the simulation, is

$$\mathcal{L}_{\text{ref}}[H] = \int_{\mathcal{T}} (H(\theta, t) - H_{\text{ref}}(t))^2 dt. \quad (6.11)$$

In general, functionals are naturally compatible with acoustic metrics, as these can be expressed as integrals over time – a form we will assume for $\mathcal{L}$ from now on.

Limiting the discussion to a single parameter θ , as indicated in Section 2.4, the derivative needed for gradient-based optimization is

$$\partial_{\theta} \mathcal{L}[H] = \int_{\mathcal{T}} \underbrace{\delta H(t)}_{:= (\partial_H \mathcal{L}[H])(t)} \partial_{\theta} H(\theta, t) dt, \quad (6.12)$$

which has been expanded using the chain rule. This is equivalent to the Jacobian-vector product in differentiable rendering from Eq. (2.38). Intuitively, the *adjoint energy* $\delta H(t)$ captures the sensitivity of the objective regarding changes in the histogram at a time t and the *differential energy* $\partial_{\theta} H(\theta, t)$ encodes the infinitesimal change of the histogram

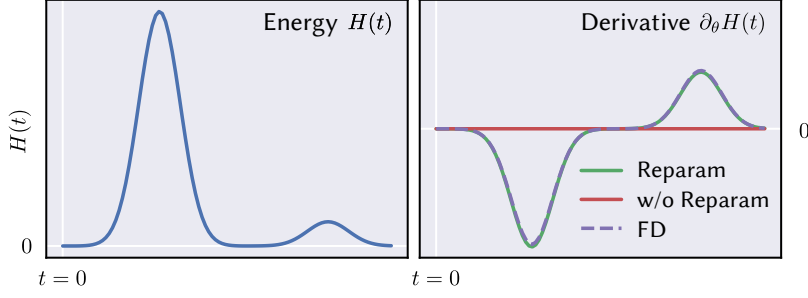


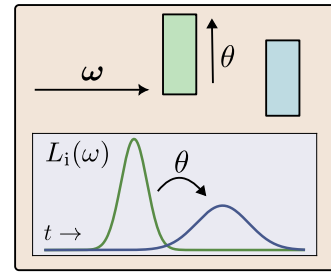
Figure 6.3: Discontinuities moving with a scene parameter θ cause a discontinuous movement of energy in time (see inset in Section 6.2.2). With the functional perspective, these discontinuities can be handled using methods from differentiable light transport, e.g., reparameterization [Loubet et al., 2019; Bangaru et al., 2020], and derivatives agree with finite differences (FD).

with θ , which are both functions of time. The adjoint energy can be easily computed by differentiating the objective. We specifically focus on the differential energy $\partial_\theta H$, as it requires differentiating the underlying simulation.

Using the functional perspective established in Section 6.2.1, we can plug in Eq. (6.9), implicitly assuming that any quantity in the scene (e.g., material or geometry) may depend on θ , and obtain

$$\partial_\theta H(\theta, t) = \partial_\theta \int_{S^2} L_i(\mathbf{m}, \omega_i, 0, t) d\omega_i. \quad (6.13)$$

Generally, the integrand contains discontinuities in ω that move with θ , so that the derivative cannot simply be moved inside the integral. In our functional framework, this is perceived by a sudden jump of the energy function along a direction ω (see inset). The same effect is observed in light transport, where the incoming radiance abruptly changes at geometry discontinuities. There is a recent line of work on handling such discontinuities for light transport [Li et al., 2018a; Loubet et al., 2019; Bangaru et al., 2020; Zhang et al., 2020] (also see Section 2.4.2). The same techniques can be applied here because the discontinuities in the time-dependent form also arise from moving geometry. Therefore, we can now assume that the derivative can be moved into the integral, e.g., because the integration domain has been reparameterized to follow the discontinuities [Loubet et al., 2019; Bangaru et al., 2020]. This allows for *automatic differentiation* of the Monte Carlo samples to correctly estimate the gradient. Figure 6.3 shows experimental verification for the scene in the inset: reparameterizing the integral yields derivatives that agree with finite differences, accounting for the energy shift to a later point in time as the first emitter moves out of the field of view. Without handling discontinuities, the derivatives remain zero.



6.2.3 Path Replay for Temporal Derivatives

In the following, we show that the acoustic rendering equation (6.6) can be differentiated using a path replay backpropagation-like method, following a derivation similar to the one for light transport (Section 2.4.3). For brevity, we omit the non-local term arising from non-static geometry (see Chapter 5) in this derivation as it can be easily added to the final formula (Appendix C.1).

We limit our discussion to $\partial_\theta L_o$ because the pixel integral, or in our case the integral in $\partial_\theta H$ (Eq. (6.9)), can be regarded as a special case of L_o with $L_e(0) = 0$ and $f = 1$. To highlight that the energy transport relation between L_i and L_o contains the local time delay τ we substitute L_i with L_o in the acoustic rendering equation (6.6):

$$L_o(\mathbf{x}, \omega, T) = L_e(\mathbf{x}, \omega, T) + \int_{S^2} f(\mathbf{x}, \omega, \omega_i) \underbrace{L_o(\mathbf{x}', -\omega_i, T + \tau)}_{L_i(\mathbf{x}, \omega_i, T)} d\omega_i^\perp. \quad (6.14)$$

In the following, we drop $\mathbf{x}, \mathbf{x}', \omega$ and ω_i for clarity but assume them to be present implicitly. Applying the multivariate chain rule leads to the following differential:

$$\begin{aligned} \partial_\theta L_o(T) = \partial_\theta L_e(T) + \int_{S^2} f \partial_\theta L_o(T + \tau) \\ + \partial_\theta f L_o(T + \tau) \\ + f \partial_T L_o(T + \tau) \partial_\theta \tau d\omega_i^\perp, \end{aligned} \quad (6.15)$$

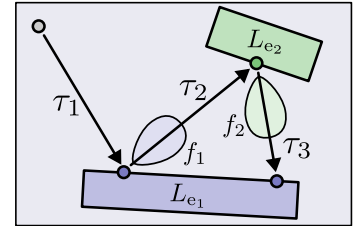
as τ might depend on θ . The first two lines are the acoustic equivalent to Eq. (2.48). The last term describes how the energy collected at later intersections changes if θ changes the length of the current path segment. Although semantically different, this term is structurally similar to the second term: evaluate some derivative with respect to the scene parameter ($\partial_\theta \tau$ vs. $\partial_\theta f$) and multiply it with some term ($f \partial_T L_o$ vs. L_o) that is unknown a-priori. But given that neither f nor τ depend on T , the derivative of L_o w.r.t. T is

$$\partial_T L_o(T) = \partial_T L_e(T) + \int_{S^2} f \partial_T L_o(T + \tau) d\omega_i^\perp, \quad (6.16)$$

so $\partial_T L_o$ behaves exactly like normal energy L_o . Given that $\partial_T L_e$ is known for every emitter, one can estimate L_o and $\partial_T L_o$ in a single forward pass. The computation can be performed with similar formulas as in Section 2.4.3, derived in the following.

Recursive Formula for the Total Incoming Energy

Consider the configuration in the inset with a single path of length N . At each bounce k there is a (potentially zero) emission profile L_{e_k} that gets shifted by $T = \sum_{j=1}^k \tau_j$, where τ_j is the propagation time for each path segment j ; f_k is a BSDF value. The collected energy L_N of a Monte Carlo sample is then (division by pdf omitted):



$$L_N = L_{e1}(\tau_1) + f_1 L_{e2}(\tau_1 + \tau_2) + f_1 f_2 L_{e3}(\tau_1 + \tau_2 + \tau_3) + \dots \quad (6.17)$$

This can be calculated iteratively, analogous to light rendering:

$$\begin{aligned}\beta_0 &= 1, & \beta_k &= \beta_{k-1}f_k, \\ T_0 &= 0, & T_k &= T_{k-1} + \tau_k, \\ L_0 &= 0, & L_k &= L_{k-1} + \beta_{k-1}L_{e_k}(T_k).\end{aligned}\tag{6.18}$$

Recursive Formula for the Derivative One can again approximate the derivative with respect to a scene parameter θ by summing the derivatives of Monte Carlo samples. As each $\tau_k(\theta)$ can depend on θ , one can apply the multi-variate chain rule:

$$\begin{aligned}\partial_\theta L_N &= \partial_\theta L_{e_1}(\bar{T}_1) + \partial_\theta f_1(L_{e_2}(\bar{T}_2) + f_2 L_{e_3}(\bar{T}_3) + \dots) \\ &+ f_1 [\partial_\theta L_{e_2}(\bar{T}_2) + \partial_\theta f_2(L_{e_3}(\bar{T}_3) + \dots)] \\ &+ f_1 f_2 [\partial_\theta L_{e_3}(\bar{T}_3) + \partial_\theta f_3(\dots)] + \dots \\ &+ \partial_\theta \tau_1 [\partial_T L_{e_1}(\bar{T}_1) + f_1 \partial_T L_{e_2}(\bar{T}_2) + f_1 f_2 \partial_T L_{e_3}(\bar{T}_3) + \dots] \\ &+ f_1 \partial_\theta \tau_2 [\partial_T L_{e_2}(\bar{T}_2) + f_2 \partial_T L_{e_3}(\bar{T}_3) + \dots] \\ &+ f_1 f_2 \partial_\theta \tau_3 [\partial_T L_{e_3}(\bar{T}_3) + \dots] + \dots\end{aligned}\tag{6.19}$$

where $\bar{T}_k = \sum_{j=1}^k \tau_j(\theta_0)$, with $\theta_0 = \theta$, so that $\bar{T}_k$ does not depend on θ . This equation reveals the similarities to Eq. (2.52). Its first part is time-independent and analogous to light rendering. The second part is time-dependent and describes the contribution of the third line in Eq. (6.15): for each term, the derivative of τ is multiplied with the *remaining incoming differential delay energy* $\partial_T L_{k,N}$. The *total incoming differential delay energy* $\partial_T L_N$ can be calculated using the following formula, according to Eq. (6.16)

$$\partial_T L_0 = 0, \quad \partial_T L_k = \partial_T L_{k-1} + \beta_{k-1} \partial_T L_{e_k}(T_k),\tag{6.20}$$

where $\partial_T L_{e_k}(T_k)$ can be evaluated locally using automatic differentiation (in forward mode). The remaining incoming differential delay energy then is $(\partial_T L_N - \partial_T L_k)/\beta_k$. This leads to an iterative formula for the derivative of the total energy:

$$\begin{aligned}\partial_\theta L_0 &= 0, \quad \partial_\theta L_k = \partial_\theta L_{k-1} \\ &+ \beta_{k-1} [\partial_\theta L_{e_k}(T_k) + \partial_\theta f_k(L_N - L_k)/\beta_k] \\ &+ \beta_{k-1} \partial_\theta \tau_k [(\partial_T L_N - \partial_T L_{k-1})/\beta_{k-1}],\end{aligned}\tag{6.21}$$

where L_N and $\partial_T L_N$ are precomputed in the primal pass. Again, the remaining incoming energies $L_{k,N} = L_N - L_k$ and $\partial_T L_{k,N} = \partial_T L_N - \partial_T L_k$ can be stored in a single variable each:

$$\begin{aligned}L_{0,N} &= L_N, & L_{k,N} &= L_{k-1,N} - \beta_{k-1} L_{e_k}(T_k) \\ \partial_T L_{0,N} &= \partial_T L_N, & \partial_T L_{k,N} &= \partial_T L_{k-1,N} - \beta_{k-1} \partial_T L_{e_k}(T_k).\end{aligned}\tag{6.22}$$

Note that although the formulas look very similar to those presented in Section 2.4.3, $L_{i,N}$ and $\partial_T L_{i,N}$ are *functions over time*. In practice, these functions must be discretized, so the memory requirements of the variables above would depend on the time resolution.

6.2.4 Backwarding with Resolution-Independent Storage

We have not yet covered how to evaluate the integral of adjoint energy $\delta H(t)$ and differential energy $\partial_\theta H(t)$ in Eq. (6.12). Ideally, one would like to backpropagate the adjoint energy during an adjoint pass in the style of radiative backpropagation, and we have shown that all required quantities can be computed by replaying paths.

For light transport path replay backpropagation, Vicini et al. [2021] expand the product of adjoint radiance δL and differential radiance $\partial_\theta L_o$ as

$$\begin{aligned} \delta L \cdot \partial_\theta L_o &= \delta L \cdot \partial_\theta L_e + \int_{\mathcal{S}^2} f \delta L \cdot \partial_\theta L_o \\ &\quad + \partial_\theta f \delta L \cdot L_o \, d\omega_i^\perp, \end{aligned} \quad (6.23)$$

and backpropagate at each intersection, effectively accumulating terms contributing to $\partial_\theta \mathcal{L}$ (see their Section 4.1 for details).

The adjoint radiance δL is the derivative of the objective w.r.t. to the contribution of a single path, derived from a small region in the adjoint image. A temporal equivalent to δL is difficult to obtain because a single acoustic path contributes to many different times in an energy histogram (each intersection point has a distinct travel time) and it would be a function, not a scalar. Instead, we directly “backpropagate” the adjoint histogram δH , shared by all paths, by expanding Eq. (6.12) using the differential transport equation (6.15):

$$\begin{aligned} \langle \delta H, \partial_\theta L_o(T) \rangle &= \langle \delta H, \partial_\theta L_e(T) \rangle + \int_{\mathcal{S}^2} f \langle \delta H, \partial_\theta L_o(T + \tau) \rangle \\ &\quad + \partial_\theta f \langle \delta H, L_o(T + \tau) \rangle \\ &\quad + f \langle \delta H, \partial_T L_o(T + \tau) \rangle \partial_\theta \tau \, d\omega_i^\perp, \end{aligned} \quad (6.24)$$

where we use the notation

$$\langle \delta H, G(\dots) \rangle := \int_{\mathcal{T}} \delta H(t) G(\dots, t) \, dt. \quad (6.25)$$

The transformation is valid because the order of integration over $\mathcal{S}^2$ and $\mathcal{T}$ can be changed.

Path Replay with Measured Energy This change does not directly lead to an efficient algorithm due to the functional perspective: so far, we have treated the transported energies as continuous functions over time, but an implementation must discretize them. A straightforward and common approach is to discretize these functions using (not necessarily) uniformly-sized time bins, such that the energy histogram H is a one-dimensional array. Each Monte Carlo sample then generates a high-dimensional quantity – the time-dependent energy – whose size depends on the time resolution. Fortunately, in the primal simulation, instead of storing one (large) array per sample, one can directly accumulate energy into a single histogram. However, naively implementing Eq. (6.24) for PRB-style differentiation leads to prohibitive storage requirements: to recover the incoming energy $L_o(T + \tau)$ and differential delay energy $\partial_T L_o(T + \tau)$ *each* Monte

Carlo sample must hold two arrays for the functions L and $\partial_T L$. With potentially millions of Monte Carlo samples and thousands of time bins this approach quickly becomes infeasible. Our key insight is that one can evaluate Eq. (6.24) in a way that decouples the storage complexity from the time resolution *without* approximations – storing only two scalars per path. Underlying is the observation that, for the gradient, one does not need to store a function L but only its *measurement* $\langle \delta H, L \rangle$.

Similar to the initial transform, one can expand the *measured energy* $\langle \delta H, L_o \rangle$, part of the second term in the integral in Eq. (6.24), using the primal transport equation (6.14):

$$\langle \delta H, L_o(T) \rangle = \langle \delta H, L_e(T) \rangle + \int_{S^2} f \underbrace{\langle \delta H, L_o(T + \tau) \rangle}_{= \langle \delta H, L_i(T) \rangle} d\omega_i^\perp. \quad (6.26)$$

The primal transport equation previously described the transport of *functions*, while this equation describes the transport of *scalars*. Consequently, instead of collecting (functional) energy in the primal pass, it is sufficient to collect (scalar) measured energy; and the incoming measured energy $\langle \delta H, L_o(T + \tau) \rangle$ can be recovered exactly like incoming energy, by replaying transport paths.

Using the adjoint energy δH in the primal pass may seem counter-intuitive at first but is enabled by the structure of path replay backpropagation [Vicini et al., 2021]: the forward simulation performs a primal pass; then, for the derivative, *another* primal pass follows that decorrelates the gradient simulation from the forward simulation [Azinovic et al., 2019], followed by the adjoint pass that replays the paths of the second primal pass. In the gradient primal pass, the adjoint energy is known and we accumulate the measured energy according to Equation (6.26). This is a notable difference to existing methods: our primal pass collects a different quantity than the gradient primal pass (energy vs measured energy).

As for the measured energy, we can expand the *measured differential delay energy* $\langle \delta H, \partial_T L_o \rangle$, part of the third term in the integral in Equation (6.24), using Equation (6.16)

$$\langle \delta H, \partial_T L_o(T) \rangle = \langle \delta H, \partial_T L_e(T) \rangle + \int_{S^2} f \langle \delta H, \partial_T L_o(T + \tau) \rangle d\omega_i^\perp, \quad (6.27)$$

which, like differential delay energy, can be computed using forward-mode differentiation, again in the gradient primal pass.

Pseudocode Listing 2 shows the pseudocode for the algorithm: each sample in the gradient primal pass computes two scalars, the measured energy and measured differential delay energy, which are then used in the adjoint pass to recover the incoming quantities (function definitions are consistent with Vicini et al. [2021]). The function `backward_grad(<expr>, grad_out)` computes the product between the Jacobian of the expression `<expr>` and the gradient w.r.t. its output `grad_out` for a parameter θ . The function `forward_grad(<var>, {<expr_1>, <expr_2>, ...})` computes the Jacobians of all expressions $\{ \langle \text{expr}_i \rangle \}$ w.r.t. the variable `<var>` and the function `detach(<var>)` detaches `<var>` from the AD graph.

Listing 2 Pseudocode of temporal PRB with time resolution independent storage. We run `sample_path` first to get the total measured energy δHL and total measured differential delay energy $\delta H \partial_T L$ for a sample; then compute the gradient in `sample_adjoint_path`.

```
def sample_path(ray,  $\delta H$ ):
     $\delta HL = 0$ ,  $\beta = 1$ ,  $T = 0$ ,  $\delta H \partial_T L = 0$ 
    for i in range(1, N+1):
         $T += \text{time\_on\_segment}(\dots)$ 

        # Measured energy
         $\delta HL += \beta * \langle \delta H, L_e(T, \dots) \rangle$ 
         $\omega'$ , bsdf_value, bsdf_pdf = sample_bsdf(...)

        # Measured differential delay energy
         $\delta H \partial_T L += \beta * \text{forward\_grad}(T, \langle \delta H, L_e(T, \dots) \rangle)$ 

         $\beta *= \text{bsdf\_value} / \text{bsdf\_pdf}$ 
    return  $\delta HL$ ,  $\delta H \partial_T L$ 

def sample_path_adjoint(ray,  $\delta HL$ ,  $\delta H \partial_T L$ ,  $\delta H$ ):
     $\beta = 1$ ,  $T = 0$ 
    for i in range(1, N+1):
         $\tau = \text{time\_on\_segment}(\dots)$ 
         $T += \text{detach}(\tau)$ 

        # Backpropagate from "standard" terms
         $\delta HE = \beta * \langle \delta H, L_e(T, \dots) \rangle$ 
         $\delta HL -= \delta HE$ 

         $\delta_\theta += \text{backward\_grad}(\delta HE, 1)$ 

         $\omega'$ , bsdf_value, bsdf_pdf = sample_bsdf(...)
         $\delta_\theta += \text{backward\_grad}(\text{bsdf\_value}, \delta HL / \text{bsdf\_value})$ 

        # Backpropagate from time-dependent term
         $\delta_\theta += \text{backward\_grad}(\tau, \delta H \partial_T L)$ 
         $\delta H \partial_T L -= \beta * \text{forward\_grad}(T, \langle \delta H, L_e(T, \dots) \rangle)$ 

         $\beta *= \text{bsdf\_value} / \text{bsdf\_pdf}$ 
    return  $\delta_\theta$ 
```


Continuous Emitter Curves So far, we have not made any assumptions about the form of the emission curve $L_e(\mathbf{x}, \omega, T, t)$, which is the time-dependent energy radiating from an emitter, except that it needs to be differentiable with respect to time. In acoustic simulations, in particular when measuring the impulse response of a system, the emission is assumed to be a Dirac delta, which is constant over the emitter surface and independent of direction $L_e(\mathbf{x}, \omega, T, t) = \delta(t - T)$. In this slightly informal notation, $\delta(t - T)$ can be thought of as a single peak located at T . The energy histogram H can be interpreted as the *energy time curve* (ETC), which is a time-discrete and squared version of the *room impulse response*. The ETC is the basis for room acoustic metrics used to assess a room’s acoustic qualities (see Appendix C.3). With Dirac delta emission, however, it is impossible to track changes in energy that arise from moving geometry: the differential delay energy in Equation (6.16) simply evaluates to zero, because $\partial_T L_e(T, t)$ vanishes almost everywhere.

We propose instead to use Gaussian emission curves of the form

$$L_e(\mathbf{x}, \omega, T, t) = \exp\left(-\frac{(t - T)^2}{\sigma^2}\right). \quad (6.28)$$

This assumption is not only key for differentiation but also grounded in reality: a sound emission always has a nonzero duration and the emitted energy does not reach its maximum instantaneously. The differential emission $\partial_T L_e(T, t)$ for Gaussian curves is defined and continuous; this allows to account for changes in transport time when computing the differential energy.

Gaussians offer an additional practical advantage over arbitrary emission profiles: the integral of $\langle \delta H, L_e(T) \rangle$ considers the entire time domain, but, while Gaussians have global support, the contribution beyond a given threshold (e.g. four standard deviations) is negligible and may be ignored in practice. Effectively, all time integrals for the measured quantities are localized because the emission curves and their derivative have (approximately) local support. In practice, we use a very narrow Gaussian with a standard deviation of 0.25 time bins and therefore still interpret the result as an ETC.

6.3 Implementation and Validation

Our method is implemented on top of MITSUBA 3 [Jakob et al., 2022b], a system for differentiable light transport simulation. We found that most of its components are readily suitable for the simulation of sound, such as the different shapes and emitters. We implemented a custom sensor, a (possibly omni-)directional Microphone, and a custom film, a Tape. We model point sources using small spherical (omnidirectional) area emitters; directional patterns to model speaker membranes are left to future work.

The core of our method is implemented in three custom integrators:

1. `AcousticPRB` implements time-resolved path replay backpropagation for efficient gradient estimation of scene parameters using the three-point form parametrization (see Chapter 5 and Appendix C). It does not handle visibility discontinuities, so the gradients will be biased when scene parameters change the visibility between surfaces.

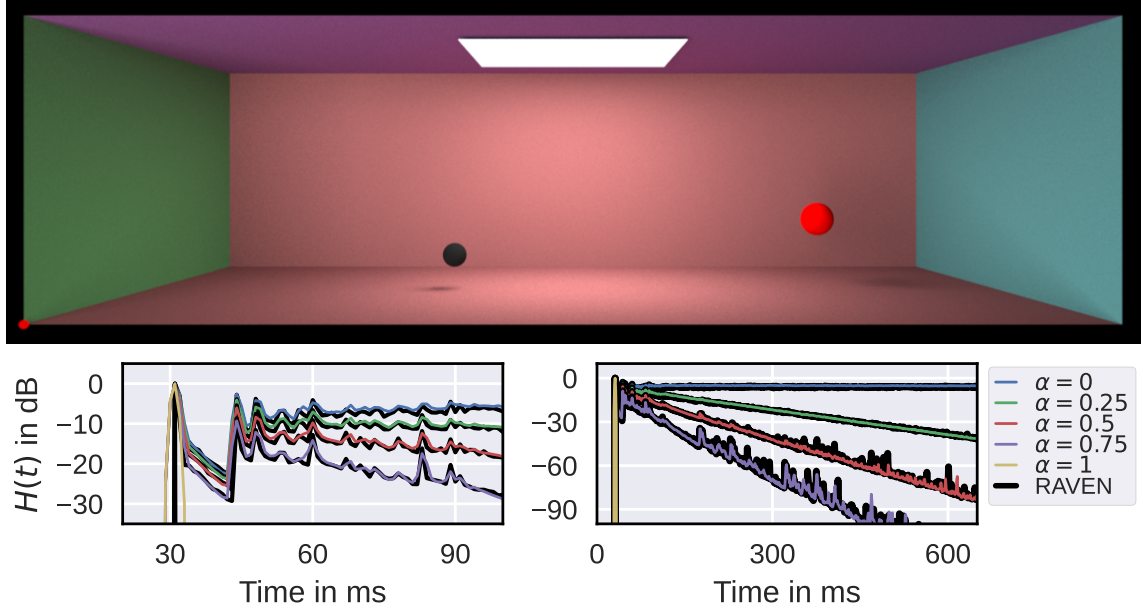


Figure 6.4: Energy time curves of the simulated “shoebox” room (top), shown with increasing absorption coefficient α and a uniform scattering coefficient of $s = 0.5$. Our simulation shows good agreement with a path tracing simulation in the software RAVEN.

2. `AcousticAD` uses conventional automatic differentiation of an acoustic rendering integrator with detached sampling of surface points (the computed gradients are equivalent to `AcousticPRB`).
3. `AcousticADReparam` uses automatic differentiation and includes a reparameterization [Loubet et al., 2019; Bangaru et al., 2020] to correctly account for derivatives of visibility discontinuities. This is the only integrator that can handle discontinuities.

If not stated otherwise, we use the `AcousticPRB` integrator. The experiments were performed on an Intel Core i9-14900KF workstation with 32 GB RAM and a NVIDIA GeForce RTX 4090. All experiments use ADAM [Kingma and Ba, 2017] for optimization.

6.3.1 Verification of Primal Simulation

Before we investigate the derivatives of acoustic simulations, we validate our forward simulation by comparing its output to RAVEN [Schröder, 2011], a widely-used simulation software for room acoustics. The test scene is a “shoebox”, commonly used in acoustic simulations (Fig. 6.4, top), with a single emitter and an omnidirectional sensor. To ensure comparability with RAVEN, the walls are modeled with the classical acoustic BSDF (Appendix C, Eq. (C.7)), where uniform values are assigned to the absorption and scattering coefficients of all surfaces. The absorption coefficient α is varied within the range of 0.1 to 0.9, while s spans from 0 to 1.

For each combination of absorption and scattering coefficients, we obtain the energy time curve from the rendered energy impulse response and subsequently calculate the reverberation time T_{60} and clarity index C_{80} (Appendix C.3).

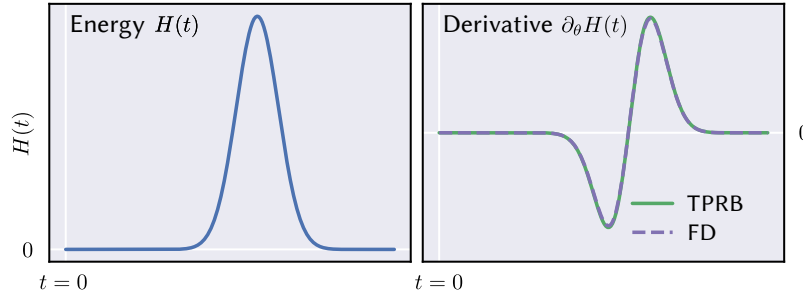


Figure 6.5: Moving reflector scene, see inset in Section 6.3.2. Our AcousticPRB integrator, based on time-resolved path replay backpropagation (TPRB), correctly handles time derivatives along the path, matching those computed with finite differences.

Fig. 6.4 shows the energy time curves for one scattering coefficient. The simulations show good agreement with RAVEN, with reverberation time differences below 8 %, averaging 2 %, and C_{80} differences below 1.5 dB, averaging 0.34 dB. In addition to path tracing, RAVEN uses the image source model (ISM) [Vorländer, 2020, Ch. 11.4] to simulate ideal specular reflections. When the image source model is disabled, the average differences decrease significantly (T_{60} : 0.33 %, C_{80} : 0.06 dB). This is no surprise, as RAVEN and our implementation now rely on the same physical model.

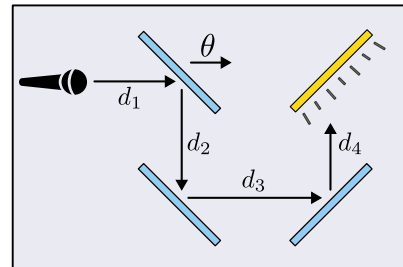
Nevertheless, the largest deviations are in the range of the just noticeable differences (JNDs): the ISO 3382-1 [2009] standard reports JNDs of 1 dB for C_{80} , and 5 % for perceived reverberance (although more recent work suggests that the JND of the reverberation time might be higher [Blevins et al., 2013]).

6.3.2 Validation of Derivatives

We now move on to a verification and exploration of our differentiation algorithm, time-resolved path replay backpropagation, as this is the main focus of the chapter.

Propagation of Time Derivatives

We first validate that variations in transport time resulting from non-static geometry are correctly handled by our algorithm. To show that the temporal derivatives not only account for variations locally but also correctly incorporate multiple bounces, we use a simple scene with three reflecting surfaces and an emitter: the scene parameter θ translates the reflector positioned next to the microphone away from the microphone, increasing the distance d_1 (see inset). To compute the derivative $\partial_\theta H(t)$, the change in d_1 must be “propagated” through the second, third, and fourth bounce to the emitter. Indeed, Fig. 6.5 shows that the derivative computed with AcousticPRB is identical to finite differences, demonstrating that the algorithm correctly handles shifts of propagation time across multiple bounces.



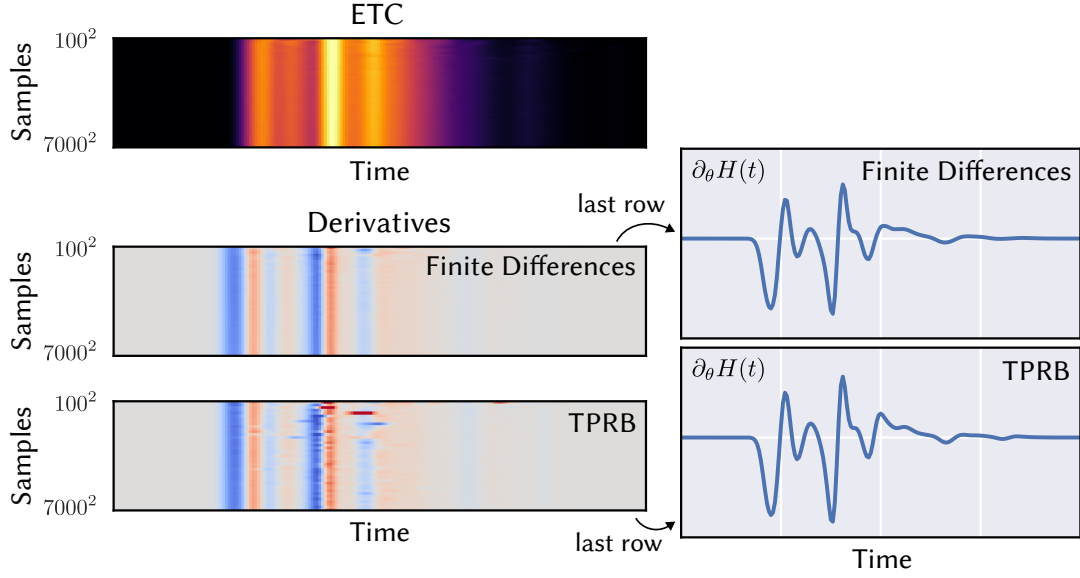


Figure 6.6: Variance of the energy histogram derivatives (see Section 6.3.2). The top left image shows the ETC computed with different numbers of samples. The derivatives w.r.t. an x-scaling of the enclosing room are computed with finite differences and our time-resolved path replay backpropagation (TPRB) algorithm. The respective results of the last row, i.e., with the highest sample count, are on the right.

Smooth Derivative Curves

The derivatives depicted so far are smooth curves, which, at first sight, seems surprising if one is used to derivative images from differentiable light transport, where noise is often clearly visible. The smoothness is easily explained: the contribution of each path in the forward simulation is a weighted and translated Gaussian (Paragraph *Continuous Emitter Curves*), so the derivative is the *sum* of weighted and translated Gaussian derivatives.

The variance present in acoustic derivatives becomes visible when repeating the same computation with varying numbers of samples. We demonstrate this with an experiment where we enclose the speaker and microphone in a cube, and the parameter θ scales the cube in x direction (paths are limited to five bounces). We compute derivatives with an increasing number of samples ranging from 10,000 to 49,000,000 and compare them with finite differences (Fig. 6.6). The forward simulation varies only slightly, while our derivatives, although pointing in the right direction, vary more. This is partially because importance sampling only considers the primal simulation, not the derivatives [Zeltner et al., 2021]. We find that the variance increases for later time bins, which is unsurprising: the early energy is concentrated in relatively few time bins, while the late energy is diffusely spread across many bins, each containing fewer samples per (nonzero) time bin. This variance is difficult to reduce because we can only increase the total sample count, but do not directly sample in the time domain.

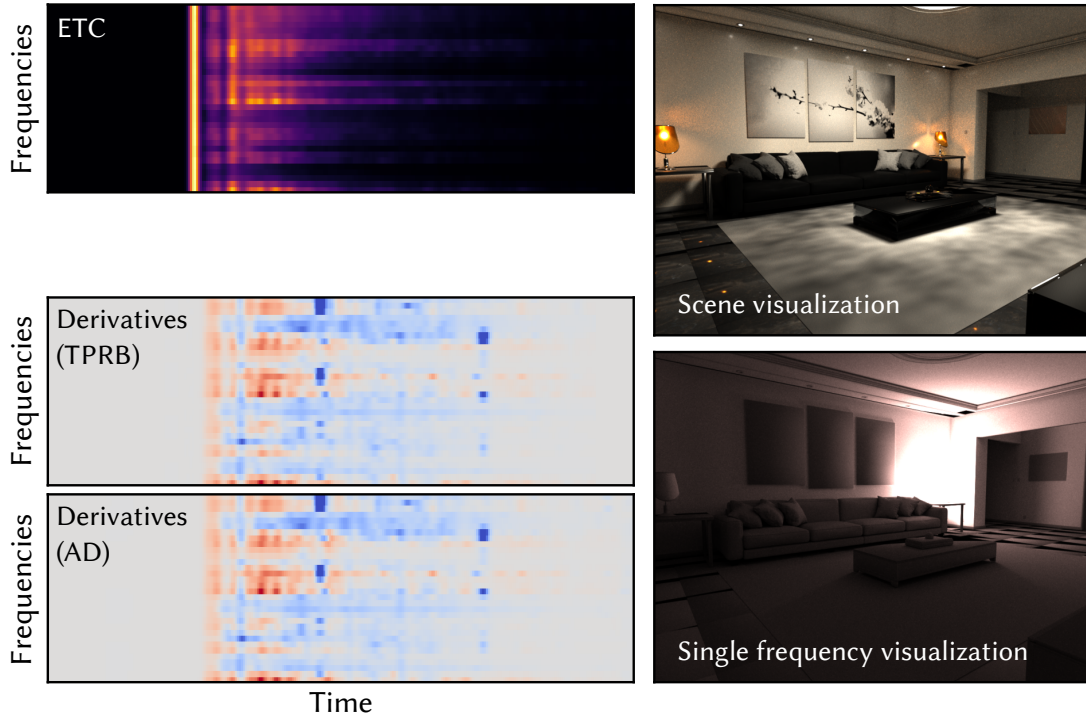
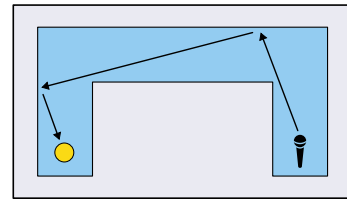


Figure 6.7: A detailed scene visualized using light transport (top right). A single emitter replaces the right lamp in the acoustic scene. The scene is rendered from the perspective of the microphone using the absorption and scattering values of the first frequency (bottom right). The spectrogram and the corresponding derivatives (as described in Section 6.3.2) are visualized on the left. AcousticPRB (TPRB) and AcousticAD (AD) agree on the derivatives.

Time-Resolved PRB compared to AD

Theoretically, AcousticPRB should produce the same derivative as AcousticAD. To test this, we set up a complex living room scene consisting of a single spherical emitter in one corner and a microphone in another corner of the room. The scene is visualized from the point-of-view of the microphone in Fig. 6.7 (right). We give each surface a random but smoothly varying absorption and scattering spectrum over 32 different wavelengths. The derivative is computed with respect to a single scene parameter θ (with initial value 1). The absorption spectrum of the walls is multiplied by θ and the absorption spectrum of the floor is multiplied by $\frac{1}{\theta}$ to get derivatives in both positive and negative directions. As shown in Fig. 6.7, the derivatives of AcousticPRB match those of AcousticAD.

Time and Memory We now verify that time-resolved path replay backpropagation has constant memory requirements and linear time complexity in path length, using a more complex scene: An omnidirectional emitter is placed in a hallway at the end of a branch, and a receiver is located at a symmetrical branch on the other side (see inset). At



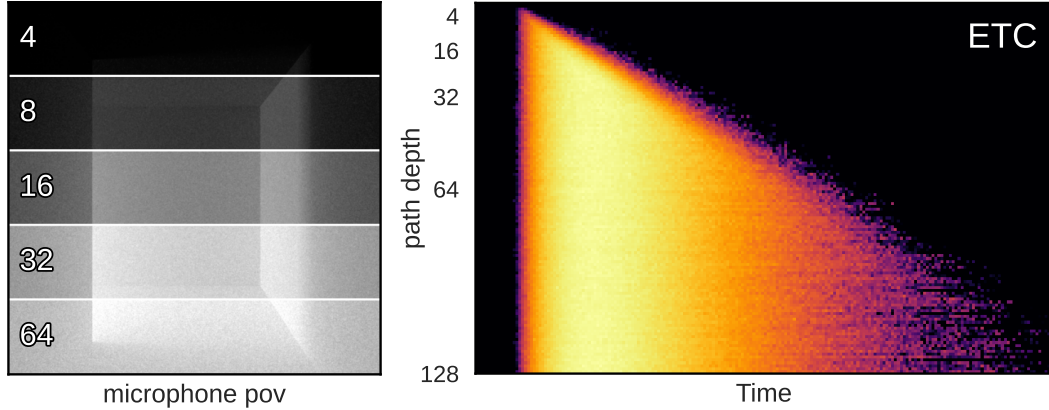


Figure 6.8: Forward simulation of a hallway scene as described in Section 6.3.2. A path depth of around 100 is necessary to estimate the entire span of the ETC (right). The same is true for the light transport simulation (left).

Table 6.1: Comparison of PRB and AD. The initial time, median and memory usage of 5 executions of a rendering and successive reverse-mode differentiation in the hallway scene described in Section 6.3.2. The last run of AcousticAD did not finish (DNF).

depth	AcousticPRB			AcousticAD		
	time		memory	time		memory
	initial	median		initial	median	
4	1.207s	0.213s	6.75MiB	22.8s	0.730s	13.5MiB
8	1.343s	0.343s	6.75MiB	85.7s	1.873s	27MiB
16	1.645s	0.616s	6.75MiB	444s	7.111s	54MiB
32	2.219s	1.246s	6.75MiB	3407s	17.178s	108MiB
64	3.681s	2.699s	6.75MiB	DNF	DNF	DNF

least three bounces are required to connect the emitter to the sensor. Generally, this is a typical scenario where only long paths will lead to correct results, as shown in the forward simulation (Fig. 6.8).

We compute the gradient using reverse-mode differentiation of the summed squared energy in the ETC with respect to three scene parameters controlling the global position of the entire hallway (which is equivalent to moving the emitter and the sensor in the same direction). The results for different path lengths are shown in Table 6.1. As expected, the results show that the memory is constant for AcousticPRB and linear for AcousticAD, but more importantly, that the execution time is lower. This is most noticeable in the first run, where the underlying JIT compiler DR.JIT [Jakob et al., 2022a] assembles a mega kernel. This even resulted in a simulation with depth 64 being terminated by the operating system before completion.

Efficiency We further demonstrate the efficiency of PRB-style differentiation in a setting with more than three parameters. For this, we place an omnidirectional emitter and receiver in the “shoebox” room and add $\frac{n}{6}$ boxes at random positions. The n scene

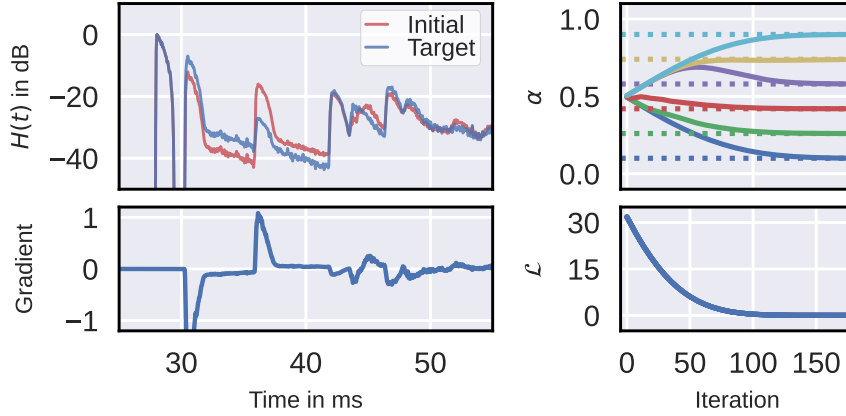


Figure 6.9: Optimizing absorption parameters of a simple room. The ETCs of the initial and target configuration (top left) and the respective gradient of the absorption factor of a single wall (bottom left). Notice that the direct sound (first peak) does not depend on the material parameters, so it does not contribute to the gradient. The absorption factors of the six walls (each plotted with a distinct color) are optimized using the mean squared error to the target ETC (right). Reference values are visualized as dashed lines.

parameters, θ , correspond to the six BSDF parameters (scattering, reflectance, specular reflectance, specular lobe width) of each box. We calculate the gradient of a simple loss defined as the squared norm of the ETC with respect to all parameters θ using reverse-mode differentiation. For each n , we run the experiment 5 times with 725^2 samples and provide the median (inset). This emphasizes the significance of path replay backpropagation: the computation of time-dependent derivatives in prior work takes 10s of seconds or minutes, even for a single scene parameter [Wu et al., 2021; Yi et al., 2021], while our implementation performs reverse-mode differentiation for thousands of parameters in a few seconds, and achieves sub-second runtimes for hundreds of parameters.

n	time
6	0.173s
60	0.271s
300	0.740s
600	1.38s
1200	2.634s
1800	3.680s

6.4 Applications

6.4.1 Optimization of Material Parameters

A common first step when building a room model for acoustic simulations is fitting the acoustic properties of the model’s walls (absorption and/or scattering) such that the simulation matches a measurement. In this section, we demonstrate that such a task can be efficiently performed using time-resolved path replay backpropagation, even for many material parameters: we will minimize the mean squared error between the estimated and the reference ETC (see Eq. (6.11)) to optimize material parameters. Since the geometry is fixed, no discontinuities occur, and `AcousticPRB` computes unbiased gradients.

A Simple Scene Although it may seem simple, finding the material parameters of all six surfaces (walls, floor, and ceiling) within a shoebox scene is a challenging task when

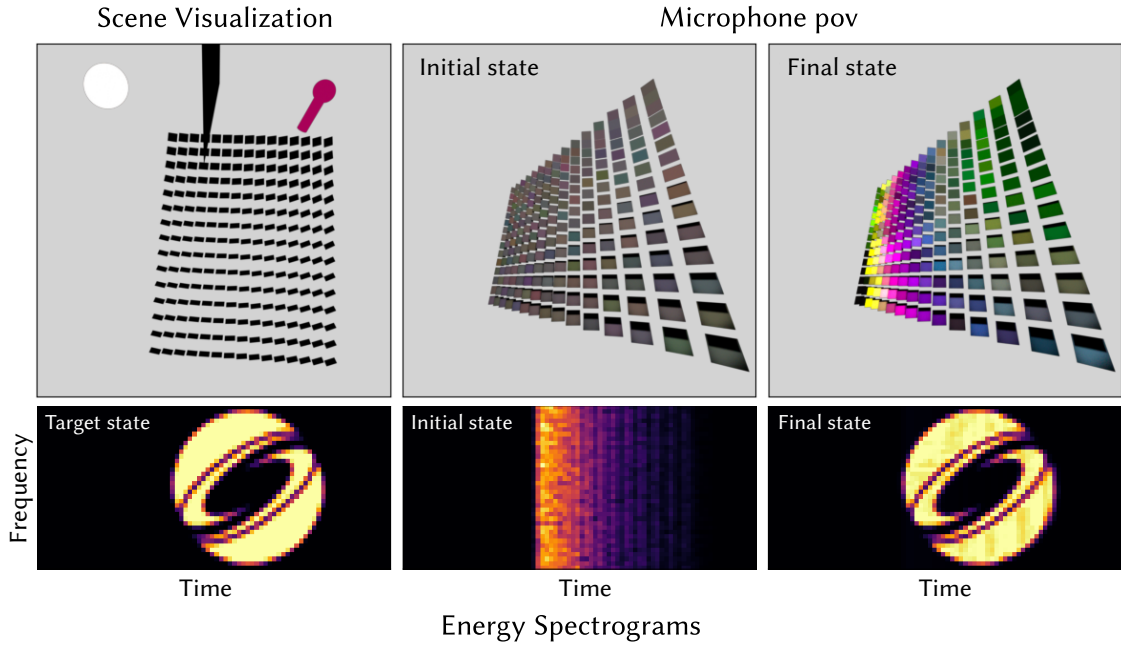


Figure 6.10: Optimizing many material parameters given a target spectrogram (lower left). The scene on the top left is defined by a spherical emitter (white), a directional microphone (red) without line-of-sight to the emitter, and 225 slabs. The scene is rendered from the perspective of the microphone before (top center) and after (top right) optimization, by mapping the absorption spectrum of the slabs from Hz to the visual wavelengths between 300-800 nm. The energy spectrogram after the optimization is shown at the bottom right.

relying solely on a single measured ETC because the contributions of different surfaces will overlap in the measurement. We use an artificial reference ETC, ensuring that any discrepancies arise from the computed gradients rather than measurement-simulation discrepancies and optimize the absorption factors of the walls. As shown in Fig. 6.9, our method correctly computes gradients for the individual surface parameters and reliably converges to the reference.

Many Coefficients To demonstrate that our implementation can handle many absorption coefficients, we set up a scene with 15 by 15 slabs, each with an absorption spectrum defined by 37 linearly interpolated values, resulting in $15 \times 15 \times 37 = 8325$ parameters. To avoid direct energy, the emitter and sensor do not share a line of sight. All energy must be reflected by the slabs. As target we use an artificial energy spectrogram representing the SIGGRAPH logo. The scene is initialized with random, high absorption coefficients. The degrees of freedom are enough to reconstruct the logo reasonably (Fig. 6.10).

6.4.2 Optimization of Scene Geometry

Apart from selecting surface materials, room acoustic design often involves strategic placement of reflectors, dividers, or source/listener positions. In new constructions, even

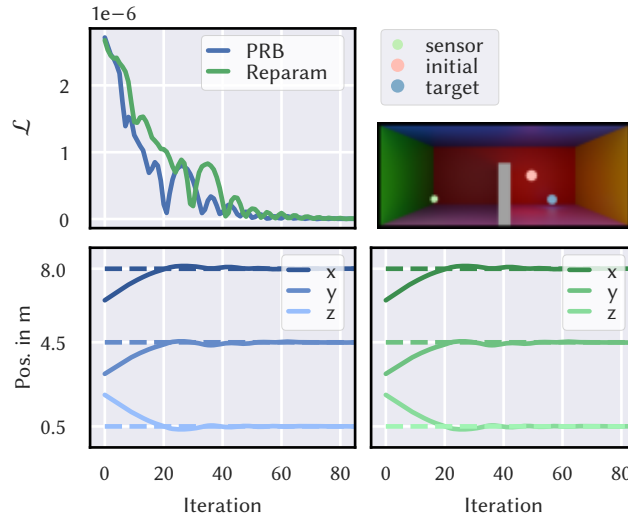


Figure 6.11: The position of an emitter in a shoebox scene is reconstructed, given a target ETC without direct line-of-sight between receiver and emitter (top right). Both *AcousticPRB* (blue, bottom left) and *AcousticADReparam* (green, bottom right) are able to recover the target emitter position.

the room’s size and shape may be subject to optimization.

No Discontinuity Handling The gradients of the *AcousticPRB* integrator are biased when geometry is non-static because it does not handle discontinuous visibility. For light transport, it is well-known that ignoring the derivatives arising from discontinuities can severely compromise an optimization (also see Section 2.4.2). Yet, for the case of time-resolved differentiable rendering and respective inverse problems, the implications are generally less studied. An exhaustive study is beyond the scope of this chapter, but we will demonstrate that optimization of geometry is feasible, even without discontinuity handling. Consider a simple shoebox scene with a vertical wall at its center, an omnidirectional receiver placed on the left side, and an omnidirectional emitter placed on the right side, such that there is no direct line-of-sight between the two (Fig. 6.11, top right). Given the ETC of a target state, the goal is to optimize the position of the emitter by minimizing the mean squared distance to the target ETC (Fig. 6.11).

Despite having biased gradients, the *AcousticPRB* integrator successfully recovers the target position. Its convergence behavior similar to that of the *AcousticADReparam* integrator, which uses reparameterization-based discontinuity handling [Loubet et al., 2019; Bangaru et al., 2020]. On the one hand, this can certainly be attributed to the *AcousticPRB* integrator using the surface parameterization, as it is free of some discontinuities [Zhang et al., 2020] (also see Chapter 5). On the other hand, resolving the time dimension and tracking variations of propagation time might be the dominant factor in time-resolved optimizations – this is correctly handled by our path replay algorithm.

Optimizing Ceiling Reflector Angles We now move on to a more realistic scenario, which addresses a common challenge in the acoustic design of rooms for speech and

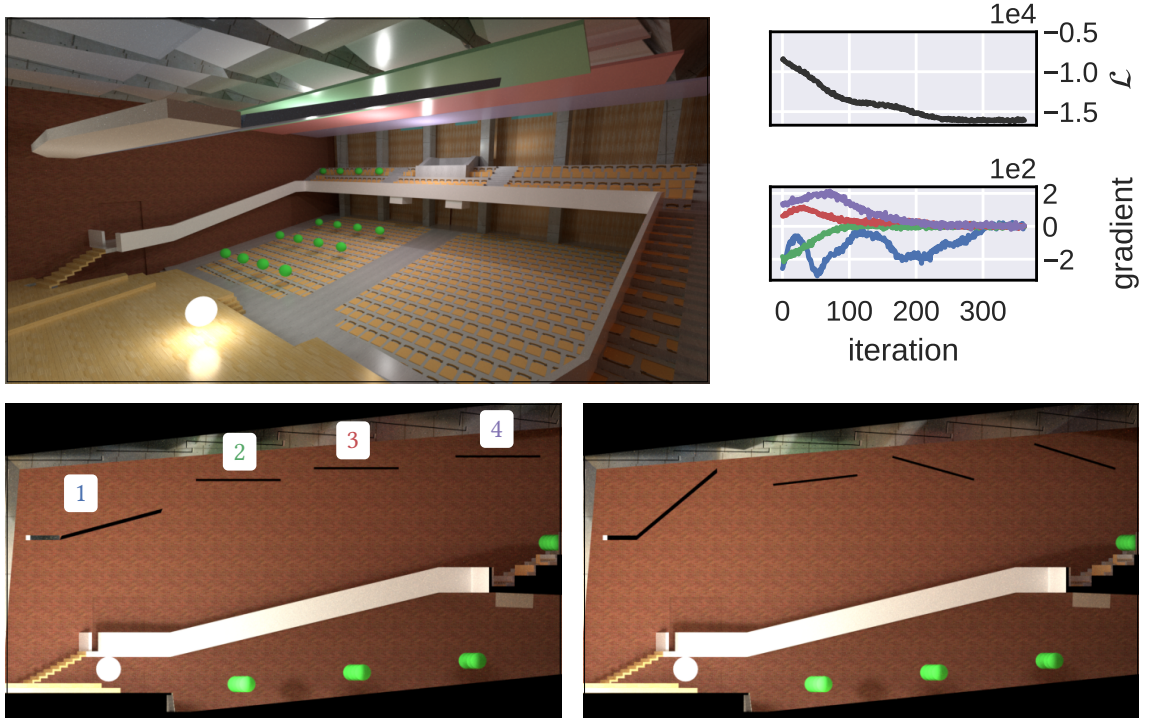


Figure 6.12: Optimization of the ceiling reflector angles in an auditorium. The loss $\mathcal{L}$ converges to a minimum and the gradients vanish as the scene converges from the initial configuration (bottom left) to the final configuration (bottom right). The reflectors are numbered 1 to 4, and the colors indicate the corresponding reflectors.

music: achieving both high intelligibility and clarity, while providing a uniform listener experience. Typically, it is desirable to maximize clarity measures such as C_{80} , i.e. to maximize the energy within the first 80 ms after the direct sound, while minimizing the variation in energy across listener positions. In concert halls, ceiling reflectors are often used to direct sound to the audience. Finding the optimal shape and positioning of these reflectors is not a trivial task, as the reflected sound reaches the audience not only via first-order reflections, but also via later, higher-order reflections. We demonstrate that our method can, in this complex setting, simultaneously optimize the angle of one reflector positioned above the stage and three additional reflectors positioned above the audience. The scene is a lecture hall, taken from the BRAS Benchmark [Brinkmann et al., 2021; Aspöck et al., 2020, CR4] (Fig. 6.12, top left).

The objective function that both maximizes the clarity and minimizes the variance across listener positions can be formulated as a joint loss function, as proposed by [IEC 60268-16, 2020]:

$$\mathcal{L}(E) = \text{std}(E_{\text{early}}) - \text{mean}(E_{\text{early}}). \quad (6.29)$$

This formulation assumes that the clarity depends only on the early energy E_{early} , which arrives within 80 ms after the direct sound and the late energy remains unchanged. This is a reasonable assumption since we only adjust the position of reflectors and don't introduce additional absorptive surfaces. This allows us to restrict the maximum rendering length, which speeds up computation times.

The optimization process, shown in Fig. 6.12, redirects early sound energy towards the back of the audience, where the ratio between early and late reverberant energy is comparatively low. This results in an overall increase in clarity across the audience (from 3.7 dB to 4.3 dB), while the standard deviation of early energy across listener positions is largely maintained (increasing slightly from 0.94 dB to 0.96 dB). The most significant improvements can be observed at individual receiver positions, where clarity increases by up to 2.9 dB in the higher ranks, which is clearly above the just-noticeable difference of approximately 1 dB [ISO 3382-1, 2009].

6.5 Conclusion

This chapter has transferred concepts of differentiable path tracing from light transport to the acoustic setting, leading to time-resolved path replay backpropagation, an efficient backpropagation algorithm for time-dependent energy transfer. It extends path replay backpropagation [Vicini et al., 2021] to time-resolved rendering, thereby addressing an open problem previously stated in the context of differentiable transient rendering [Yi et al., 2021; Wu et al., 2021]. Our experiments demonstrated that differentiable path tracing of sound propagation is feasible and enables the optimization of various scene parameters, including material properties and geometry, to achieve desired acoustic characteristics in room environments. This opens up applications that appear to go significantly beyond traditional techniques in acoustics. Our implementation can be easily extended to model properties that are currently not supported, such as sources or receivers with arbitrary directional characteristics.

Biased gradients An apparent limitation is that our AcousticPRB integrator produces biased gradients when visibility is affected by scene parameters. As shown in Chapter 5, there is currently no implementation that simultaneously handles visibility discontinuities and is PRB-based. A combination of the two will likely benefit differentiable acoustic path tracing, although we have shown that geometry optimization is still feasible without it (Section 6.4.2).

Efficient Differentiation for Transient Rendering Our approach is connected to differentiable transient rendering, which is commonly derived from a path space formulation [Wu et al., 2021; Yi et al., 2021]: although our formalism is based on the classical rendering equation with a spherical integral, our implementation uses a surface integral, whose derivatives are closely related to the interior derivatives of the path-space form.

We see an exciting avenue in combining our insights with the path-space formalism of transient rendering, which may lead to more robust discontinuity handling in differentiable acoustic simulations on the one hand, and more efficient differentiable transient rendering on the other hand.

Wave effects Our current implementation is based on path tracing, which ignores all effects related to the wave nature of sound. While this is consistent with ray tracing techniques used in acoustics, it is nonetheless a more serious simplification of physical

phenomena compared to the simulation of light transport. The literature in acoustics, but also in graphics, has shown many efficient techniques for incorporating wave effects into the simulation [Kirsch and Ewert, 2023; Svensson and Calamia, 2006; Pohl and Stephenson, 2014; Steinberg et al., 2024]. It will be worthwhile to explore how such approaches could be integrated into the Monte Carlo simulation framework. For improved fidelity and simulation in arbitrary frequency bands, full spectral path tracing could be implemented as suggested by Radziszewski et al. [2009]. Spectral rendering with arbitrary sensitivity spectra and logarithmic sampling in octave bands has already been successfully applied to the forward rendering pass [Jüterbock et al., 2024].

CHAPTER 7

Neural Deferred Shading

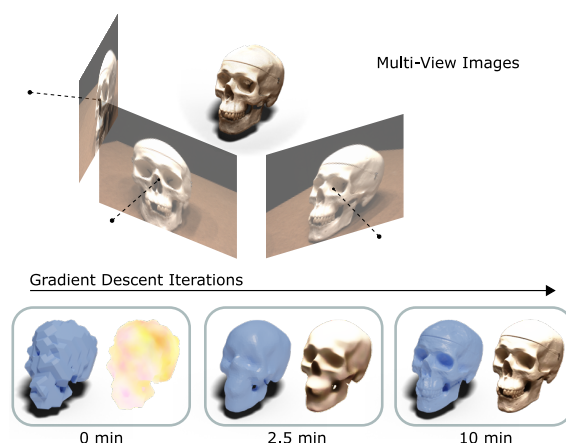


Figure 7.1: We reconstruct an object from images by simultaneously deforming a triangle mesh and optimizing a neural shader, comparing the renderings to the input images.

The previous chapters have been mainly concerned with differentiating traditional rendering systems: Chapter 3 and Chapter 4 focused on rasterization-based rendering with approximations from real-time graphics, whereas Chapter 5 and Chapter 6 were concerned with physically-based rendering using Monte Carlo methods to simulate energy transport. The resulting tools are general and can be applied in a wide range of applications. However, by catering to all *possible* applications these tools might not be the optimal choice in terms of effectiveness or efficiency for *any single* application.

For example, suppose that a set of images is given, all showing the same object from different viewpoints. The task is to estimate its 3D geometry, i.e., a virtual representation of the surface. If one were to approach this task by using differentiable rendering to minimize an objective that measures the difference between rendered images and target images, then it might be reasonable to use a physically-based differentiable renderer: it is less restrictive and can model the various light transport effects observed in general capture settings. However, since traditional rendering systems simulate (or approximate) light transport in a virtual scene, one must know many “meta” parameters of the scene a priori: the emitter model, the number of emitters, a suitable material model for the object, etc. Many of these parameters can be known in well-calibrated capture setups [Schreer et al., 2019], but generally they are not. Even if one *would* take an educated guess and select reasonable models, this choice and the optimized model parameters would really only be relevant for the optimization procedure and not the final result – after all, the only relevant output is the estimated geometry.

In that regard, inverse applications open up a design space for differentiable rendering systems that are tailored to the optimization objective. A recent example is 3D Gaussian splatting [Kerbl et al., 2023]: even for target images showing scenes with significant geometric and visual complexity, it requires no coherent model for surfaces, materials, or illumination; and substantial approximations in rendering suffice [Celarek et al., 2025]. This works because the application is novel view synthesis: one is neither interested in recovering accurate 3D geometry, nor in estimating surface materials.

The same principle can be applied in the multi-view reconstruction setting above: instead of fixing material and illumination models to simulate their interaction, it is sufficient to model the *appearance* of the object, which generally is view-dependent and spatially-varying. This idea is popular in modern 3D reconstruction pipelines because it requires minimal prior knowledge about the material of objects or the environment in which the input images were captured [Niemeyer et al., 2020; Yariv et al., 2020; Oechsle et al., 2021; Li et al., 2023]. This also contrasts traditional approaches, which often rely on photo-consistency between images [Fua and Leclerc, 1995; Furukawa and Ponce, 2010], assuming Lambertian surfaces and constant illumination. Instead, neural networks commonly serve as a flexible representation of appearance *and* geometry [Niemeyer et al., 2020; Yariv et al., 2020], both jointly optimized: one set of network weights models appearance, i.e., the interaction of geometry, material and light, and another set of (potentially shared) weights models the geometry, often as an implicit surface [Park et al., 2019; Liu et al., 2019b]. Although fully neural approaches are general, they exhibit excessive runtime, making them impractical for reconstruction of large object collections or 3D reconstruction from multi-view video (e.g., of human performances [Starck and Hilton, 2007; Vlasic et al., 2009; Guo et al., 2019; Collet et al., 2015; Schreer et al., 2019]).

In this chapter, we propose *neural deferred shading* (NDS) an efficient differentiable rendering approach tailored to 3D surface reconstruction from multi-view images. Our idea is to combine the best from real-time graphics and 3D vision: represent surfaces as triangle meshes and model appearance using neural networks. This provides a general and efficient tool for reconstruction tasks because (1) triangle meshes can be efficiently rasterized, whereas neural implicit surfaces require expensive ray marching, and (2) the renderer can reproduce arbitrary appearance, including global illumination, without costly light transport simulations. The design is inspired by real-time rendering with *deferred shading* [Deering et al., 1988]: a differentiable rasterizer generates per-pixel attributes from a triangle mesh (e.g., the world-space position), and a *neural shader* then processes these attributes to generate the final color. Similar to previous work [Yariv et al., 2020], our shader is a shallow multi-layer perceptron (MLP), a representation that is also common for neural material models [Sztrajman et al., 2021; Kuznetsov et al., 2022; Zeltner et al., 2024]. Although in a different context, Thies et al. [2019] and Nalbach et al. [2017] also present pipelines for mesh rendering with neural deferred shading. However, unlike our MLP shader, their convolutional networks can “hallucinate” colors at image regions not covered by geometry – this interferes with the goal of optimizing geometry to represent the observations. We use our differentiable renderer to jointly optimize a triangle mesh and the shader in a 3D reconstruction setting, and demonstrate on a public dataset that the reconstruction accuracy is close to that of traditional baselines and neural approaches, while significantly surpassing them in optimization runtime.

7.1 Method

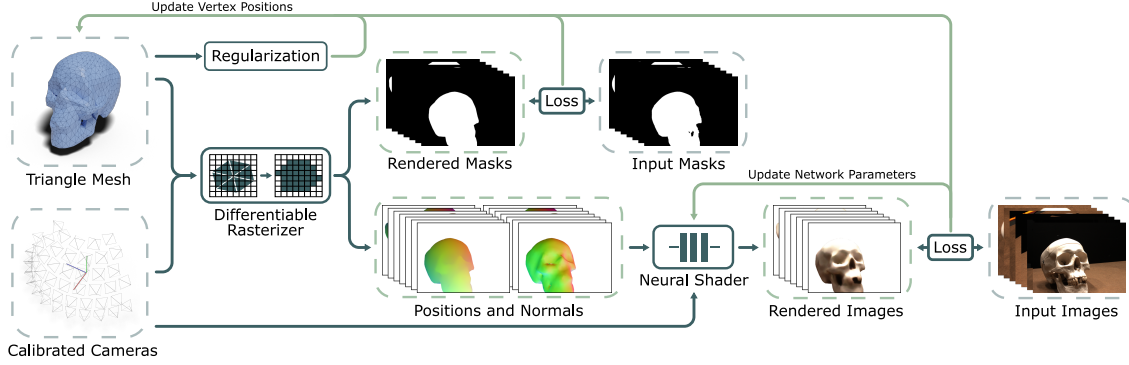


Figure 7.2: Overview of the 3D reconstruction pipeline. We use a differentiable rasterizer to obtain per-pixel attributes from a triangle mesh and shade the results with a neural network to synthesize an image for each input view. The shader is updated based on the difference between rendered and input images, whereas the mesh vertices are also updated based on the silhouette and a geometric regularization term.

Given a set of images $\mathcal{I} = \{I_1, \dots, I_n\}$ from calibrated cameras and corresponding masks $\mathcal{M} = \{M_1, \dots, M_n\}$, we want to estimate the 3D surface of an opaque object shown in the images. To this end, we follow an analysis-by-synthesis approach: we find a surface that reproduces the images when rendered from the perspective of the input views. In this work, the surface is represented by a triangle mesh $\mathcal{G} = (\mathbf{V}, \mathcal{E}, \mathcal{T})$, consisting of vertex positions $\mathbf{V}$, a set of edges $\mathcal{E}$, and a set of faces $\mathcal{T}$. Faithfully reproducing the images via rendering requires an estimate of the surface material and illumination if we simulate light transport, e.g., with a differentiable path tracer [Li et al., 2018a; Nimier-David et al., 2019]. However, as initially stated, our focus is on geometry reconstruction. Therefore, we combine the interaction of material and illumination, and only estimate *appearance*, which avoids restrictions imposed by opting for specific material and light models: our rendering pipeline uses differentiable rasterization and a neural shader, which can represent arbitrary appearance (Section 7.1.1). We solve the optimization problem using gradient descent, gradually deforming the mesh and adjusting the weights of the neural shader using an objective function that compares renderings of the mesh to the input images (Section 7.1.2). Fig. 7.2 shows an overview of our approach.

7.1.1 Neural Deferred Shading

Our differentiable mesh renderer follows the structure of a deferred shading pipeline from real-time graphics: Given a camera i , the mesh is rasterized in a first pass, yielding a triangle index and barycentric coordinates *per pixel*. This information is used to interpolate both vertex positions and vertex normals, creating a geometry buffer (G-buffer) with per-pixel positions and normals. In a second pass, the G-buffer is processed by a shader

$$f_{\theta}(\mathbf{x}, \mathbf{n}, \omega_o) \in [0, 1]^3 \quad (7.1)$$

with parameters θ . The shader returns an RGB color for a given position $\mathbf{x} \in \mathbb{R}^3$, normal $\mathbf{n} \in \mathbb{S}^2$, and view direction $\omega_o = \frac{\mathbf{c}_i - \mathbf{x}}{\|\mathbf{c}_i - \mathbf{x}\|}$, with $\mathbf{c}_i \in \mathbb{R}^3$ being the center of camera i . We represent the shader as a shallow multi-layer perceptron, with θ being the parameters of the fully-connected layers (Figure 7.3). The network is designed in two parts: The first part takes the position $\mathbf{x}$ as input and extracts a feature vector from it. The second part takes this positional feature vector concatenated with the view direction ω_o and normal $\mathbf{n}$ as input, and generates the view-dependent color. In this context, it has been shown that providing the normal and view direction with the position is necessary for disentangling the geometry from the appearance [Yariv et al., 2020]. In addition to a color image, the renderer also produces a mask that indicates if a pixel is covered by the mesh.

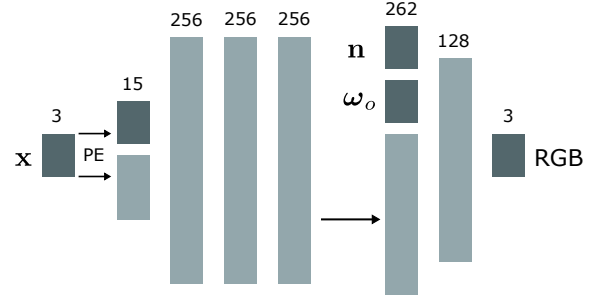


Figure 7.3: Architecture of the neural shader. The position $\mathbf{x}$ is transformed by a positional encoding (PE) [Tancik et al., 2020] and processed by 3 fully-connected layers. The resulting feature vector is concatenated with the normal $\mathbf{n}$ and the view direction ω_o and processed by two more layers. We use ReLU activations for the hidden layers and a sigmoid activation for the last layer.

7.1.2 Objective Function

Finding an estimate of shape and appearance formally corresponds to solving the following minimization problem in our framework

$$\arg \min_{\mathbf{V}, \theta} L_{\text{appearance}}(\mathcal{G}, \theta; \mathcal{I}, \mathcal{M}) + L_{\text{geometry}}(\mathcal{G}), \quad (7.2)$$

where $L_{\text{appearance}}$ compares the rendered appearance of the estimated surface to the camera images and L_{geometry} regularizes the mesh to avoid undesired vertex configurations.

Appearance

The appearance objective is composed of two terms

$$L_{\text{appearance}} = L_{\text{shading}} + L_{\text{silhouette}}, \quad (7.3)$$

where the shading term

$$L_{\text{shading}} = \lambda_{\text{shading}} \frac{1}{|\mathcal{I}|} \sum_{i=1}^{|\mathcal{I}|} \|I_i - \tilde{I}_i\|_1 \quad (7.4)$$

ensures that the color images produced by the shader $\tilde{I}_i$ correspond to the input images and the silhouette term

$$L_{\text{silhouette}} = \lambda_{\text{silhouette}} \frac{1}{|\mathcal{M}|} \sum_{i=1}^{|\mathcal{M}|} \|M_i - \tilde{M}_i\|_1 \quad (7.5)$$

ensures that the rendered masks $\tilde{M}_i$ match the input masks for all views. Here, $\|\cdot\|_1$ denotes the mean absolute error of all pixels in an image. Formally, the masks $\tilde{M}_i$ are functions of the geometry $\mathcal{G}$ and the parameters of camera i , while the color images $\tilde{I}_i$ are also functions of the neural shader (or more precisely its parameters θ).

Separating the shading from the silhouette objective mainly has performance reasons: For a camera view i , the rasterization considers all pixels in the image, therefore computing the mask $\tilde{M}_i$ is cheap. However, shading is more involved and requires invoking the neural shader for all pixels after rasterization, which is an expensive operation. In practice, we only shade a subset of pixels inside the intersection of input and rendered masks while comparing the silhouette for all pixels. Additionally, we also limit the number of camera views considered in each gradient descent iteration.

Geometry Regularization

Naively moving the vertices unconstrained in each iteration quickly leads to undesirable meshes with degenerate triangles and self-intersections. We use a geometry regularization term that favors smooth solutions and is inspired by Luan et al. [2021]:

$$L_{\text{geometry}} = L_{\text{laplacian}} + L_{\text{normal}}. \quad (7.6)$$

Let $\mathbf{V} \in \mathbb{R}^{n \times 3}$ be a matrix with vertex positions as rows, the Laplacian term is defined as

$$L_{\text{laplacian}} = \lambda_{\text{laplacian}} \frac{1}{n} \sum_{i=1}^n \|\delta_i\|_2^2, \quad (7.7)$$

where

$$\delta_i = (LV)_i \in \mathbb{R}^3 \quad (7.8)$$

are the differential coordinates [Alexa, 2003] of vertex i , $L \in \mathbb{R}^{n \times n}$ is the graph Laplacian of the mesh $\mathcal{G}$ and $\|\cdot\|_2$ is the Euclidean norm. Intuitively, by minimizing the magnitude of the differential coordinates of a vertex, we minimize its distance to the average position of its neighbors.

The normal consistency term is defined as

$$L_{\text{normal}} = \lambda_{\text{normal}} \frac{1}{|\tilde{\mathcal{T}}|} \sum_{(i,j) \in \tilde{\mathcal{T}}} (1 - \mathbf{n}_i \cdot \mathbf{n}_j)^2, \quad (7.9)$$

where $\tilde{\mathcal{T}}$ is the set of triangle pairs that share an edge and $\mathbf{n}_i \in \mathbb{S}^2$ is the normal of triangle i (under an arbitrary ordering of the triangles). It computes the cosine similarity between neighboring face normals and enforces additional smoothness.

While some prior work (e.g., [Liu et al., 2018; Luan et al., 2021]) uses El Topo [Brochu and Bridson, 2009] for robust mesh evolution, we found that our geometric regularization sufficiently avoids degenerate vertex configurations. Without El Topo, we are unable to handle topology changes, but also avoid its impact on runtime.

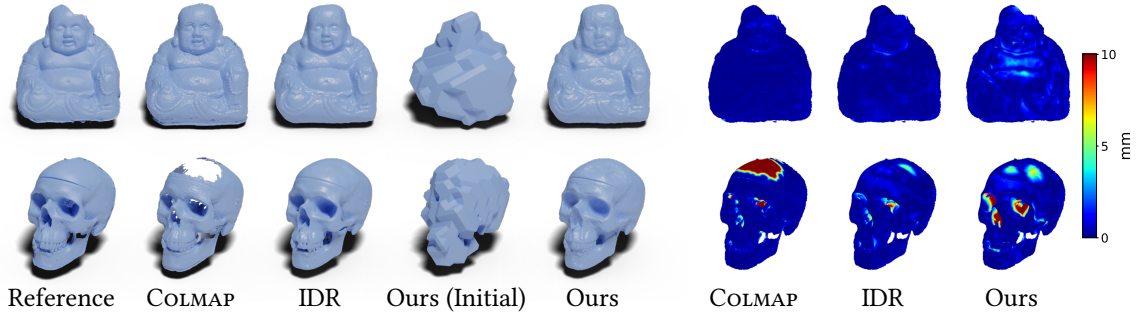


Figure 7.4: Qualitative comparison on the DTU dataset. Left: Reference geometry and reconstruction results. Right: Point-to-mesh distance between the reference scan and the reconstructed mesh.

7.1.3 Optimization Schedule

Our optimization starts from an initial mesh that is computed from the masks and resembles a visual hull [Laurentini, 1994]. Alternatively, it can start from a custom mesh.

Similar to prior work, we use a coarse-to-fine scheme for the geometry: Starting from a coarsely triangulated mesh, we progressively increase its resolution during optimization. Inspired by Nicolet et al. [2021] we remesh the surface with the method of Botsch and Kobbelt [2004], halving the average edge length multiple times at fixed iterations. After mesh upsampling, we also increase the weights of the regularization terms by 4 and decrease the gradient descent step size for the vertices by 25 %, which we empirically found helps to improve the smoothness for highly tessellated meshes.

Since some quantities in geometry regularization (e.g., graph Laplacian) only depend on the connectivity of the mesh, we save time by precomputing them once after upsampling and reusing them in the iterations after.

7.2 Experiments

We have implemented our method on top of the automatic differentiation framework PyTorch [Paszke et al., 2019] and use the ADAM [Kingma and Ba, 2017] optimizer for momentum-based gradient descent. Our differentiable rendering pipeline uses the high-performance primitives by Laine et al. [2020]. In our experiments, we run 2000 gradient descent iterations and remesh after 500, 1000, and 1500 iterations. We randomly select one view per iteration to compute the appearance term and shade 75 % of mask pixels. The individual objective terms are weighted with $\lambda_{\text{shading}} = 1$, $\lambda_{\text{silhouette}} = 2$, $\lambda_{\text{laplacian}} = 40$, and $\lambda_{\text{normal}} = 0.1$. All time measurements were taken on a workstation with an Intel Xeon 32×2.1 GHz CPU, 128 GB RAM, and an NVIDIA Titan RTX GPU with 24 GB VRAM.

7.2.1 3D Reconstruction

We demonstrate that our method can be used for multi-view 3D reconstruction. Starting from a coarse visual hull-like mesh, it quickly converges to a reasonable estimate of the object surface. We test our method on the DTU multi-view dataset [Jensen et al.,

Table 7.1: Quantitative results for multi-view reconstruction of objects from the DTU dataset. Chamfer scores are in *millimeters* and the COLMAP runtime is for the trim7 configuration.

Scan	COLMAP trim7 (trim0)		IDR		NDS (Ours)	
	↓ Chamfer-L1	↓ time [min]	↓ Chamfer-L1	↓ time [min]	↓ Chamfer-L1	↓ time [min]
24	0.45 (0.81)	66.81	1.58	551.83	4.24	12.24
37	0.90 (2.03)	81.19	2.06	566.13	5.25	7.56
40	0.36 (0.75)	65.47	0.75	550.19	1.30	10.69
55	0.36 (1.20)	64.71	0.43	565.27	0.53	7.54
63	0.90 (1.75)	75.62	1.06	553.07	2.47	6.18
65	0.94 (1.55)	62.62	0.79	568.41	1.22	9.95
69	0.53 (1.02)	86.77	0.68	557.58	1.35	9.64
83	1.16 (3.03)	77.96	1.38	745.18	1.59	4.76
97	1.08 (1.42)	50.89	1.17	743.11	2.77	7.32
105	0.63 (1.96)	51.59	0.88	742.31	1.15	7.00
106	0.48 (0.99)	108.01	0.63	735.23	1.02	7.41
110	0.58 (1.33)	85.87	0.99	752.94	3.18	6.68
114	0.31 (0.50)	88.39	0.37	730.50	0.62	7.99
118	0.44 (0.78)	105.75	0.50	748.60	1.65	6.77
122	0.43 (1.17)	80.80	0.52	747.90	0.91	5.76
mean	0.64 (1.35)	76.83	0.92	657.22	1.95	7.83

2014] with the object selection and masks from previous work [Niemeyer et al., 2020; Yariv et al., 2020]. We compare our results to two methods: (1) COLMAP [Schönberger and Frahm, 2016; Schönberger et al., 2016], a traditional SfM pipeline that serves as a baseline, and (2) IDR [Yariv et al., 2020], a state-of-the-art analysis-by-synthesis method that uses neural signed distance functions as geometry representation. By default, our COLMAP results include trimming (trim7) and we indicate untrimmed results explicitly (trim0).

Figure 7.4 shows qualitative results for two objects from the DTU dataset and Table 7.1 contains quantitative results for all objects. We used the official DTU evaluation script to generate the Chamfer-L1 scores and benchmarked all workflows for the time measurements (including data loading times). For IDR and our method we disabled any intermediate visualizations.

In absolute terms (millimeters), the reconstruction accuracy of our method is close to both the traditional baseline and the state-of-the-art neural method. Although COLMAP reconstructs many surfaces accurately, only IDR and our method properly handle regions dominated by view-dependent effects (e.g., non-Lambertian materials) and produce watertight surfaces that can remain untrimmed. When COLMAP is used without trimming, the reconstruction becomes more complete but it is less accurate than ours for some objects. Our method is limited by the topology and genus of the initial mesh and therefore cannot capture some geometric details that can be recovered with more flexible surface representations. We also observe that our surfaces are often not as smooth as IDR’s and concave regions are not as prominent. The latter is potentially related to

the mesh stiffness induced by our geometry regularization.

On the other hand, our method is significantly faster: roughly 10 times faster than COLMAP and 80 times faster than IDR in the default configuration. Since the number of iterations is a hyper parameter for IDR and our method (and also has different semantics), we show equal time results for a fair comparison of both (Figure 7.5). Our method quickly converges to a detailed estimate, while IDR only recovers a rough shape in the same time. Even after 50 minutes, IDR still lacks details present in our result.

Since IDR and our method have similar architectures, i.e., both perform analysis-by-synthesis and use gradient descent to jointly optimize shape and appearance, we can meaningfully compare the runtime in more detail. Table 7.2 shows the runtime of *one* gradient descent iteration. An iteration in IDR takes roughly twice the time as in our method, with the majority of time spent for ray marching the implicit function. In contrast, the time taken for rasterizing the triangle mesh is negligible. Most of the ray marching time in IDR can be attributed to evaluating the neural network, thus switching to a more shallow neural representation could be a way to reduce the runtime. The remaining time is spent on operations like root finding, which could be accelerated by a more optimized implementation.

However, the runtime difference in gradient descent iterations cannot be the sole reason for our fast convergence. Even though iterations in IDR require twice the time, it requires more than twice the total time to show the same level of detail as our method (see Figure 7.5). We noticed that after mesh upsampling finer geometric details quickly emerge. Thus, the fast convergence time might partially be related to the fact that we can locally increase the geometric freedom with a finer tessellation, while IDR and similar methods have no *explicit* control over geometric resolution.

7.2.2 Analysis of the Neural Shader

Although recovering geometry is the main focus of our work, investigating the neural shader after optimization can potentially give insights into the reconstruction process

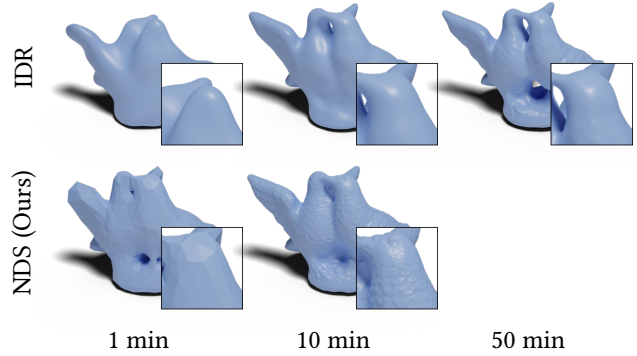


Figure 7.5: Equal time reconstruction. We consider our result converged after 10 minutes. Even after 50 minutes, IDR lacks the details present in our reconstruction (e.g., feathers and eyes).

Table 7.2: Average runtime of one gradient descent iteration of IDR and our method, decomposed hierarchically. Geometry rendering time excludes shading and corresponds to ray marching 2048 pixels for IDR and rasterizing to 1.9 million pixels in our case.

	IDR		NDS (Ours)	
	time [s]	share	time [s]	share
Gradient descent iteration	0.3577	100 %	0.1561	100 %
↳ Geometry rendering	0.2099	↳ 58.7 %	0.0034	↳ 2.2 %
↳ SDF evaluation	0.1472	↳ 70.1 %	–	

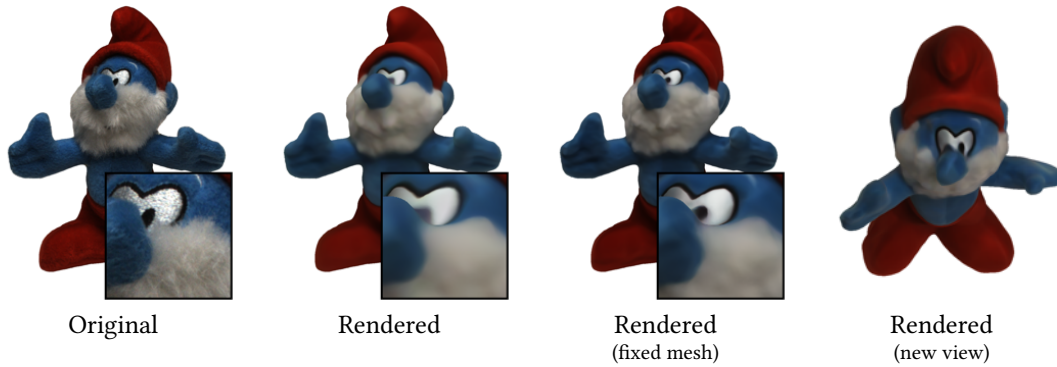


Figure 7.6: View synthesis for an input view and a new view. By fixing the mesh after reconstruction and continuing shader optimization, we can further improve the view synthesis results.

and the information that is encoded in the network.

Interactive Novel View Synthesis Figure 7.6 shows that rendering with a trained shader can be used for basic view synthesis, producing reasonable results for input and novel views. Thus, the shader seems to learn a meaningful representation of appearance, disentangled from the geometry. If desired, the view synthesis quality can be further improved by continuing the optimization of the shader while keeping the mesh fixed.

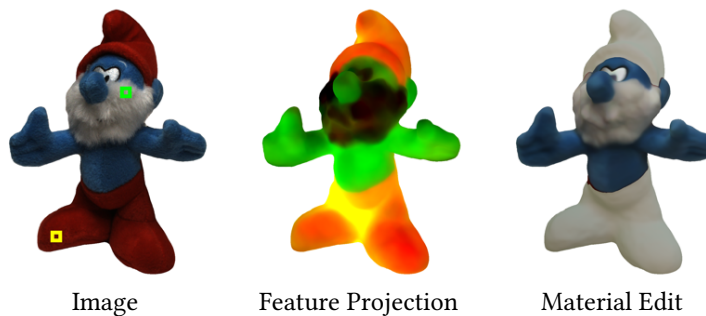


Figure 7.7: A principle component analysis of the shader’s positional features reveals a connection to materials. We show the projection to the two largest principle components. Replacing features in the positional latent space before computing the view-dependent color enables simple material editing. The feature vector at the yellow square (pants material) determines regions that are replaced with the feature vector at the green square (beard material).

The neural shader can be readily integrated into other real-time rendering pipelines after training, enabling interactive synthesis of novel views. As a demonstration, we have implemented a “neural” viewer that uses OpenGL to rasterize positions and normals. Then, we use OpenGL-CUDA interoperability and PyTorch to shade the buffers with a pre-trained neural shader.

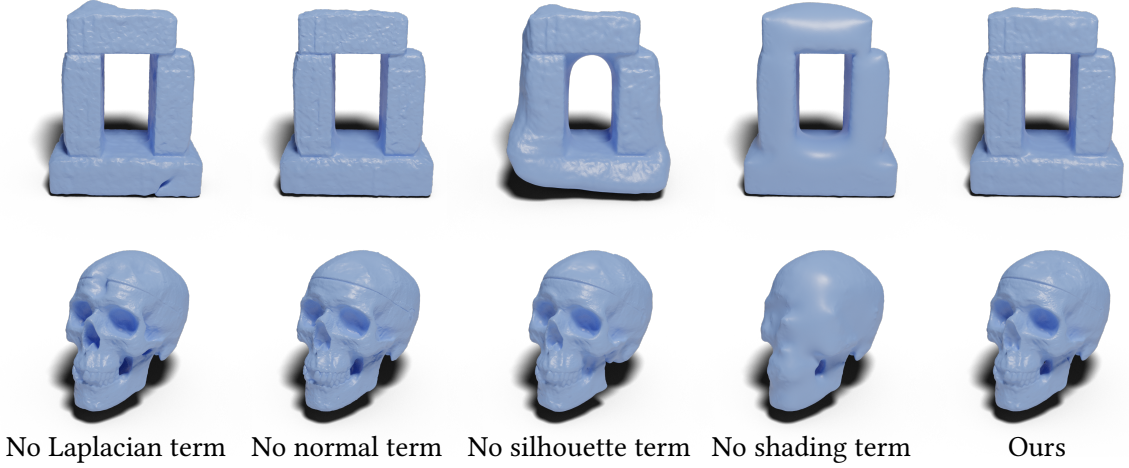


Figure 7.8: Ablation study of the objective function.

Positional Features The neural shader is a composite of two functions (Figure 7.3)

$$f_{\theta}(\mathbf{x}, \mathbf{n}, \omega_o) = c(h(\mathbf{x}), \mathbf{n}, \omega_o), \quad (7.10)$$

where $h : \mathbb{R}^3 \rightarrow \mathbb{R}^{256}$ transforms points in 3D space to positional features and $c : (\mathbb{R}^{256} \times \mathbb{S}^2 \times \mathbb{R}^3) \rightarrow [0, 1]^3$ then extracts view-dependent colors. Both functions dependent on parameters θ_h and θ_c , respectively. To further analyze the shader, we perform a principle component analysis of the positional features from h and project them to the two dominant components (Figure 7.7): The shader naturally learns similar positional features for regions with similar material, despite their distance in space. Variations in illumination also seem to be encoded in the positional features because shadowed regions have slightly different features than exposed regions of the same material.

We also investigate the behavior of the view-dependent part by replacing feature vectors in the positional latent space and then extracting colors with the function c . More specifically, we replace all features representing one material by a feature representing another material (Figure 7.7). The results suggest that the function c reasonably generalizes to different view and normal directions, which in this example leads to geometric features of the mesh being still perceivable. This allows simple material editing.

7.2.3 Ablation Studies

The method has several hyper-parameters, which we ablate in the following.

Objective Function We investigate the influence of the individual terms of our objective function (Figure 7.8). Without the Laplacian term, we observe noticeable bumps and crack-like artifacts. The normal term has a small influence but improves smoothness around edges. Without the silhouette term, the object boundaries are not properly reconstructed. Without the shading term, the reconstructed shape resembles a smooth visual hull, missing almost all details.

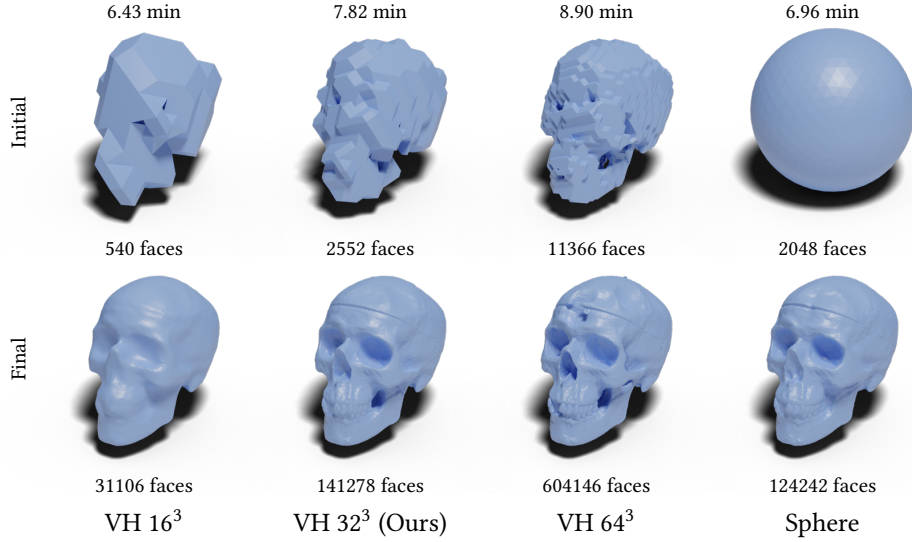


Figure 7.9: Reconstruction results for different initial meshes. We vary the resolution of the grid used to build the visual hull (VH) and also start from a sphere.

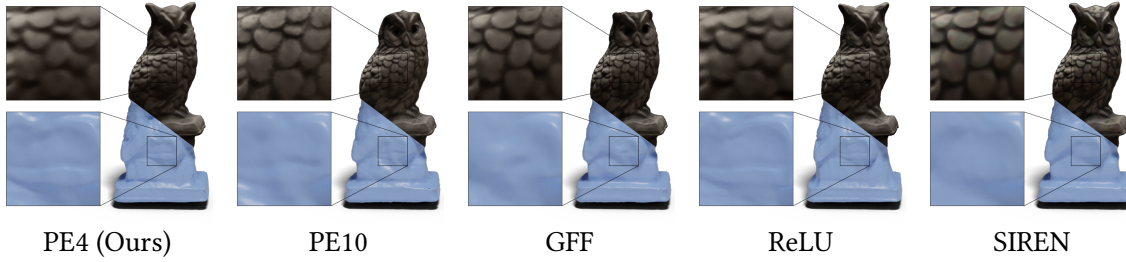


Figure 7.10: Ablation of different encodings for the positional features and activation functions. Note that the ears of the owl are only present in few views and not part of the reference geometry, thus are not considered in the comparison. For ReLU and SIREN we use no encoding.

Initial Mesh We investigate how initial meshes with different resolutions, topology, and geometric distance to the target affect the reconstruction and runtime (Figure 7.9). Very coarse initial meshes result in missing details, while fine meshes provide too much geometric freedom, which leads to artifacts. Since we do not support topology changes, holes are only reconstructed if they are present in the initial geometry.

The initial meshes for all DTU objects consist of around 2000 triangles and the output meshes of around 100000 triangles. In terms of scalability, our method handles high-resolution meshes efficiently: in the last 500 iterations (with 2000 iterations in total), the optimization runs on a mesh with the output resolution and still performs fast updates.

Positional Encoding and Network Architecture We experiment with different encodings and network sizes for the position dependent part of the neural shader. In Figure 7.10, we show the different results using positional encoding (PE) [Tancik et al., 2020], Gaussian Fourier features (GFF) [Tancik et al., 2020], sinusoidal activations (SIREN) [Sitz-

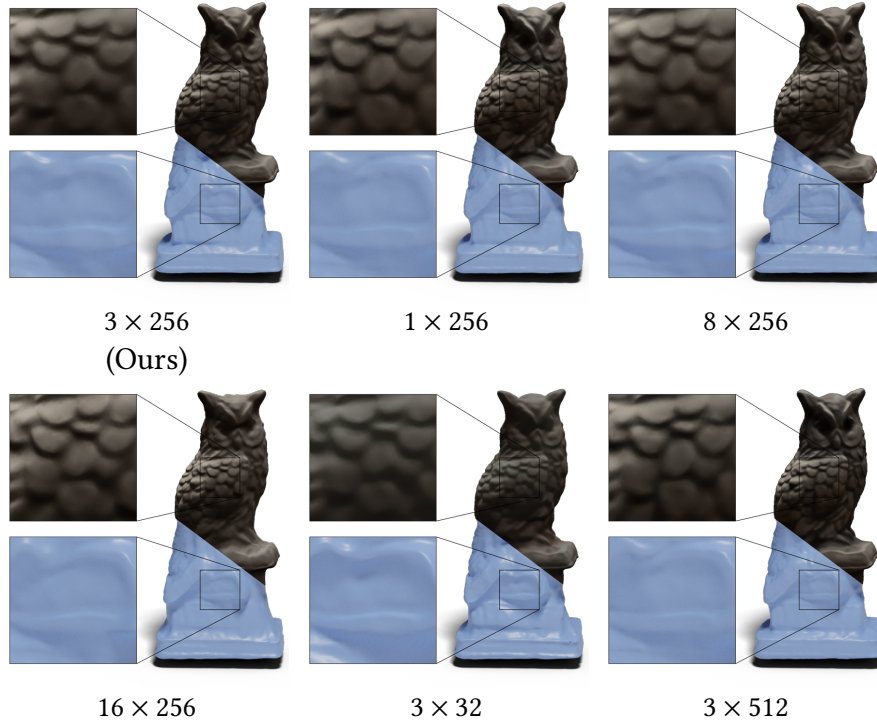


Figure 7.11: Different sizes of the positional part of the neural shader (number of layers $\times$ width of layers).

mann et al., 2020] and a standard MLP with ReLU activations.

While some of these methods can generate acceptable renderings, they do not necessarily guarantee a more detailed geometry. In particular, we observe that although the image rendered with SIREN or GFF has sharp features, the geometric detail of the mesh is less adequate. A possible explanation is that the network quickly overfits and compensates for geometrical inaccuracies using the appearance. Conversely, finding the correct direction along which to move the mesh vertices might be more difficult without positional encoding. We have obtained accurate reconstructions using positional encoding with 4 octaves, as opposed to the recommended 10 [Mildenhall et al., 2020].

In Figure 7.11, we show results for different network architectures. While there are no dramatic differences between the results, we observe that configurations with less than 2 layers or with more than 512 units per layer result in fewer geometric details.

7.2.4 Failure Cases

Figure 7.12 shows some failure cases of our reconstruction method. Since the reconstruction starts from a visual hull-like mesh, deep concavities require large movements of the mesh. This movement is driven by relatively weak shading gradients, which cannot recover these concavities before the mesh becomes overly stiff (e.g., because the gradient descent step size is reduced during remeshing). On the other hand, silhouette gradients are larger than shading gradients, so the mesh easily “grows” to fill the masks.

We also observed that the remeshing operation sometimes produces tangled meshes

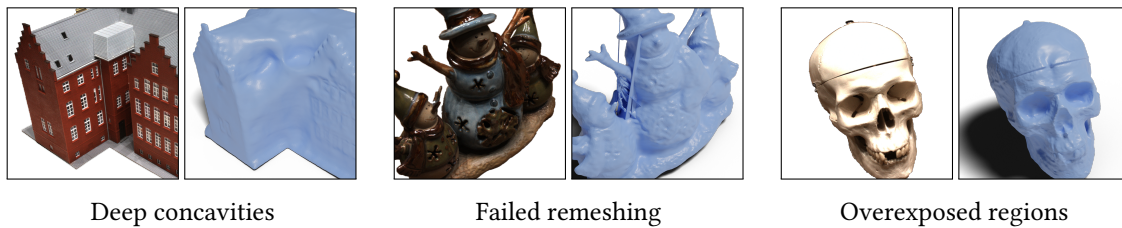


Figure 7.12: Failure cases of our reconstruction approach.

for some objects. This failure case is unrecoverable and the reconstruction needs to be restarted. The reason for this is unclear but it is likely related to the implementation of the remeshing library [Sellán and Nicolet, 2021].

Some regions show weak structure in the majority of images for a variety of reasons, for example because they are overexposed. Since we randomly sample one camera view per iteration, there is a high probability to select an image with low structure information for these regions. Using such a view can drive the regions’ vertices in unexpected directions away from the actual surface. If such movement occurs at the wrong time (e.g., right before remeshing), the optimization often fails to correct those vertices.

7.3 Conclusion

We have presented a fast analysis-by-synthesis approach for 3D surface reconstruction from multiple images. We jointly optimize a triangle mesh and a neural shader as representations of geometry and appearance to reproduce the input images, using a custom differentiable renderer that combines the speed of triangle mesh rasterization with the generality of neural rendering. Our method is comparable to state-of-the-art in reconstruction accuracy, yet is significantly faster with average runtimes below 10 minutes.

Limitations and Future Work. While triangle meshes are a simple and efficient representation, optimizing them robustly in inverse settings remains an open challenge. We currently avoid undesirable mesh configurations (e.g., self-intersections) with carefully weighted geometry regularizers that steer the optimization towards smooth solutions, but this involves a compromise between rendering and regularization objectives. Nicolet et al. [2021] instead precondition the gradient to favor smooth deformations of the mesh, which would be worth exploring in our setting. For the coarse-to-fine process, we currently remesh the surface at fixed points during the optimization, which does not always yield the required resolution necessary for reconstructing fine details. A promising alternative could be continuous remeshing [Palfinger, 2022]. Topology changes are challenging for most mesh-based methods, including ours, and computationally expensive [Brochu and Bridson, 2009; Liu et al., 2018]. Hybrid approaches appear to be a reasonable compromise: a more flexible surface representation (e.g., a density field or an implicit surface) is used in the first part of the optimization to obtain a coarse estimate of the scene; then a mesh is extracted and refined using differentiable mesh rendering [Tang et al., 2023].

While neural shading is a powerful component of our system, allowing us to handle non-Lambertian surfaces and complex illumination, it is also a major obstruction for interpretability. We experimentally provided preliminary insights into the black-box shader but also think that a more thorough analysis is needed. Alternatively, physical light transport models could be combined with more specialized neural components (e.g., irradiance or material) to isolate their effects. In this context, pre-training components to include learned priors is a promising direction [Boss et al., 2021; Zhang et al., 2021].

Impact Our approach has since inspired several other works. For example, Walker et al. [2024] replace the explicit vertex deformation with a neural deformation field, which is more robust and avoids over-smoothing the solution with strong regularization terms. Bharadwaj et al. [2023] use our framework to reconstruct avatars from monocular video efficiently. They deform the canonical mesh vertices of a parametric face model and replace the black-box neural shader with a physically-based shading model and neural illumination to make the avatars relightable. Algan et al. [2025] combine our pipeline with a diffusion model to stylize and edit reconstructed meshes using text instructions.

CHAPTER 8

Conclusion

In this thesis, we explored different ways to improve the efficiency of differentiable rendering in terms of runtime and memory requirements. This has led to several methods and algorithms that are not only faster but also more lightweight than previous alternatives. Efficiency is a central factor that determines whether an approach is practical in real-world applications or not. We believe that our contributions provide tools that will broaden the scope of differentiable rendering by enabling approaches that previously appeared computationally infeasible.

In Chapter 3, we began with the realization that real-time rendering provides many approximations that could be useful for differentiable rendering. In particular, global effects like shadows have received less attention, or when they have, the attempt did not yield derivatives that would correctly account for the variation with scene parameters. We have shown that one of the oldest approximations for shadows, shadow maps, can be adapted to work in differentiable rendering frameworks. The central idea was to combine pre-filtered shadow mapping, a framework for softening shadow boundaries, and differentiable rasterization. Our approach to computing shadows maintains the efficiency of traditional shadow mapping, is differentiable, and orders of magnitude faster than comparable techniques from physically-based differentiable rendering.

We have then continued this line of work in Chapter 4, driven by the observation that the approach derived in Chapter 3 was merely a special case of a more general family of approximations based on the theory of moments. A central takeaway from our previous efforts, and many other efforts in the existing literature, was that automatically differentiating arbitrary approximations from real-time rendering does not necessarily yield useful derivatives. Instead of relying on empirical studies to analyze the properties of these moment-based approximations, we mathematically proved that they are, in fact, differentiable. The careful theoretical considerations turned out to be useful on the practical side, and led to an implementation that is robust and efficient. On the application side, the more general moment-based techniques improve over the simpler shadow mapping variant used in Chapter 3, at almost the same cost. They also provide new tools for approximating transmittance through volumes in differentiable rendering, significantly surpassing existing approximations in accuracy while being equally fast.

While real-time techniques are central building blocks in many practical differentiable rendering systems, they are fundamentally limited by their approximate nature, rendering them unsuitable for applications that require precise physical accuracy. In Chapter 5, we have hence turned our focus to physically-based differentiable rendering, where approximations are typically avoided, so efficiency must be achieved through other means. Previous work has shown that the algorithmic level provides significant

leverage: instead of storing huge computational graphs for complex light transport simulations, reverse-mode differentiation can be achieved using a different type of light transport simulation involving differential quantities – with the same time and memory complexity as the forward simulation. However, we demonstrated that this algorithm does not account for variations in scene geometry, which renders it unsuitable for a wide range of applications. We revisited the underlying theory and presented the necessary corrections to address these issues. Using a reparameterization of the rendering integral, we arrived at a variant of the algorithm that correctly computes derivatives for scene geometry, yet remains nearly as simple as the original formulation. As a result, many applications can now benefit from its algorithmic advances.

While differentiable rendering has reached a mature state in computer graphics, where many of the underlying techniques originated, other fields are only beginning to adopt similar ideas. In Chapter 6, we remained within the physics-based realm but turned to the simulation of sound according to the principles of geometric acoustics, which is widely used in the subfield of room acoustics. In this setting, existing optimization approaches often rely on genetic algorithms, manual parameter tuning, or simplified differentiable simulators. We showed that many techniques developed for differentiable light transport can be transferred, which enables fully-differentiable acoustic path tracing. A central challenge in differentiating these simulations is efficiently and correctly treating long transport paths and scene geometry, since variations in transport time play a fundamental role. While the algorithm introduced in Chapter 5 provides a natural starting point, it was originally derived for steady-state transport. We therefore extended it to the time-resolved setting. Surprisingly, the resulting algorithm has only minimal additional overhead over its steady-state counterpart. The methods and algorithms developed in this chapter open up a new direction for acoustic optimization that could be as impactful for the acoustics community as differentiable rendering has been for graphics and vision. At the same time, the algorithm may also benefit graphics applications, as it should naturally extend to time-resolved rendering of light.

Finally, in Chapter 7 we turned our attention back to graphics. A central observation is that inverse applications do not necessarily need to rely on traditional rendering algorithms. Instead, much more efficient solutions can often be obtained by tailoring differentiable rendering algorithms to the specific task at hand. We demonstrated this principle at the example of 3D reconstruction, by deriving a rendering algorithm that combines the efficiency of triangle rasterization with the expressiveness of neural rendering. This design enables efficient 3D reconstruction in environments with arbitrary materials and illumination without explicitly simulation light transport.

8.1 Outlook

General vs. Application-Driven Approximate differentiable rendering remains highly relevant, as demonstrated by many recent applications. Many rendering algorithms are tailored to the requirements of a specific task, often introducing approximations motivated by the optimization objective. While such designs can be very effective and efficient in their intended setting, they may not generalize well and are therefore often difficult to reuse in other contexts.

This suggests that approximations in differentiable rendering still offer considerably unexplored potential. Developing them more systematically could lead to techniques that remain broadly applicable while retaining their efficiency. In this context, automatically differentiating real-time renderers is an important step [Bangaru et al., 2023], but as demonstrated in Chapter 3 and Chapter 4, additional care is often required to ensure that the resulting derivatives are meaningful.

Gradient Variance Since physically-based differentiable rendering relies on Monte Carlo estimators, the gradients exhibit some form of variance. In practice, gradient-based optimization often uses only a small number of samples per iteration, yet optimization procedures still tend to converge. While the relationship between gradient quality and convergence speed has not been studied extensively, it is reasonable to expect that lower-variance gradient estimates enable more effective steps through the loss landscape and thus faster convergence. Alternatively, variance reduction could allow the same gradient quality to be achieved with fewer samples, reducing the overall computational cost.

A promising direction in that regard is sample reuse [Bitterli et al., 2020] across optimization iterations as successive iterations in gradient descent are closely related, similar to consecutive frames in forward rendering. While this idea has been explored previously [Wang et al., 2023b; Chang et al., 2023], several challenges remain, including how to extend such approaches to global illumination.

Another direction arises from the observation that the rendering integral can be parameterized in multiple ways, which leads to differential formulations with substantially different variance (Chapter 5). Current approaches typically use a single parameterization along an entire light path, either spherical integrals [Li et al., 2018a; Soroka et al., 2025] or path integrals [Zhang et al., 2020]. Mixing parameterizations along the path and selecting those that minimize variance could provide a powerful alternative, though this raises open questions such as how to combine parameterizations and correctly handle boundary terms.

Approximations in Physics-based Differentiable Rendering Physically-based differentiable rendering often places strong emphasis on unbiased gradient estimators. While this property is appealing from a theoretical perspective, it raises the question of whether strict unbiasedness is always necessary in practice. For example, in Chapter 5 and Chapter 6, we have shown that several optimizations can succeed even when the boundary term is ignored. Although these are primarily educational scenarios, it suggests that allowing controlled bias in parts of the derivative, e.g., using approximations, could potentially enable substantial performance improvements while still providing gradients that are sufficiently accurate for optimization.

Differentiable Rendering for Rendering Differentiable rendering is often introduced in the context of inverse problems, where a scene is optimized to match observations, and derivatives are computed with respect to scene parameters. However, the term “scene parameter” can be misleading: in principle, differentiable rendering allows optimizing *any* parameter that influences the rendered image and the resulting loss, including parameters of the renderer. While most existing work focuses on inverse tasks,

comparatively little attention has been paid to how differentiable rendering could be used to directly improve rendering algorithms in terms of image quality and efficiency.

A possible direction is to optimize a rendering pipeline using image-based objectives. Although it is possible to replace the entire pipeline with a neural network [Zeng et al., 2025b], rendering algorithms incorporate extensive domain knowledge that would have to be rediscovered by data-driven models. Instead, a promising alternative is to retain the structure of a renderer while optimizing only the individual components within it. The domain knowledge would then remain encoded in the structure and component interactions. While components could be optimized in isolation, e.g., the shadow computation [Datta et al., 2022; Hu et al., 2025], differentiable rendering enables their *joint* optimization by measuring output quality directly on the generated image. This accounts for the interplay between not only traditional components, but also potentially ones based on neural networks. For example, one could try to optimize approximations in a rasterization-based renderer to match the output of a physically-based renderer.

APPENDIX A

Differentiable Moment Bounds

A.1 Transmittance as Opacity

Differentiable volume rendering frameworks in computer vision are often designed in a way that allows simple evaluation of expressions

$$\int_0^\infty \mathbf{v}(\mathbf{r}(t)) \underbrace{T(t) \sigma(\mathbf{r}(t))}_{w(\mathbf{r}(t))} dt, \quad (\text{A.1})$$

where $\mathbf{v} : \mathbb{R}^3 \rightarrow \mathbb{R}^d$ is some field (e.g., color or more general features) [Li et al., 2023; Wang et al., 2021; Mildenhall et al., 2020]. The weights are pre-computed such that the numerical integration is

$$\frac{1}{N} \sum_{i=1}^N \mathbf{v}_i w_i \quad (\text{A.2})$$

Our differentiable volume shadows based on transmittance estimates can be trivially integrated into such frameworks, which is clear if one defines transmittance using opacity

$$T(t) = 1 - \underbrace{\int_0^t T(z) \sigma(\mathbf{r}(z)) dz}_{\text{opacity}} \quad (\text{A.3})$$

As an alternative to bounding the absorbance, we bound the opacity, which is also a non-decreasing measure. The moments can be simply computed with the weights and the N distance samples $\{t_i\}$

$$\mathbf{m} = \frac{1}{N} \sum_{i=1}^N \mathbf{u}(t_i) w_i \quad (\text{A.4})$$

This is equivalent to the approach by Peters et al. [2016].

A.2 Accuracy Verification Results

Table A.1 reports the results from the accuracy verification (Section 4.3.2, Paragraph *Accuracy Verification*), which has the purpose of empirically validating the accuracy of the (custom) C++/CUDA implementation (not establishing a ranking over accuracy). From

Table A.1: Accuracy of the implementations, measured as average over 1 Million random samples $\{(\mathbf{m}_k, \eta_k)\}$.

		32 bits		64 bits	
		$\Delta \mathbf{m}$	$\Delta \mathbf{m}$	$\Delta \frac{\partial B}{\partial \mathbf{m}}$	$\Delta \frac{\partial B}{\partial \eta}$
n = 1	PyTorch	1.70e-07	3.26e-16	3.54e-07	2.20e-07
	C++	1.68e-07	3.20e-16	3.54e-07	2.20e-07
2	PyTorch	1.53e-05	2.58e-14	2.27e-06	8.37e-07
	C++	3.14e-06	5.93e-15	2.27e-06	8.37e-07
3	PyTorch	7.07e-04	1.47e-12	1.90e-05	3.12e-06
	C++	2.51e-04	4.80e-13	1.90e-05	3.12e-06
4	PyTorch	1.56e-01	2.63e-10	3.46e-04	9.08e-06
	C++	3.96e-02	1.00e-10	3.46e-04	9.08e-06
5	PyTorch	–	1.10e-08	5.79e-03	2.82e-05
	C++	–	2.86e-09	5.79e-03	2.82e-05

1 million random samples, ones with a singular Hankel matrix or η near a singularity are omitted. For a positive moment vector $\mathbf{m}_k$ and a given η_k , $\mathbf{m}_k$ can be reconstructed from the roots and weights computed by the primal procedure (assuming no biasing). $\Delta \mathbf{m}$ is the Euclidean distance between a moment vector and its reconstruction $\|\mathbf{m}_k - \sum_{i=0}^n w_i \mathbf{u}(x_i)\|$, where $\mathbf{u}(x) = (1, x, x^2, \dots, x^{2n})^\top$ evaluates the moment curve (c.f. Theorem 1). With larger n , the moment reconstruction becomes less meaningful ($n = 5$ involves 10th power), so we omit the measurement for 32-bit precision. $\Delta \frac{\partial B}{\partial \mathbf{m}}$ and $\Delta \frac{\partial B}{\partial \eta}$ are the mean absolute error between finite differences and the computed bound derivative w.r.t. the moments and η , respectively.

A.3 Proofs

This section first introduces useful definitions and theorems regarding orthogonal polynomials (from the literature) and then continues with our proofs.

A.3.1 Orthogonal Polynomials

Proposition 2 (e.g. [Akhiezer and Kreĭn, 1962, p.4]). *If $\mathbf{m} = (m_0, \dots, m_{2n})$ is strictly positive, the polynomials of the sequence*

$$P_0(x; \mathbf{m}) = 1$$

$$P_k(x; \mathbf{m}) = \det \begin{bmatrix} m_0 & m_1 & \dots & m_k \\ m_1 & m_2 & & m_{k+1} \\ \vdots & & \ddots & \vdots \\ m_{k-1} & m_k & \dots & m_{2k-1} \\ 1 & x & \dots & x^k \end{bmatrix}$$

are pairwise orthogonal, i.e., $\mathfrak{S}_{\mathbf{m}}\{P_i P_j\} = 0$ for $i \neq j$ (c.f. Eq. (4.9)) under any measure μ that is a representation of the moments $\mathbf{m}$.

We recall two important facts about orthogonal polynomials

1. An orthogonal polynomial P_k has exactly k roots, all of which are real and simple (see, e.g., [Szegő, 1975, Theorem 3.3.1, p.44]).
2. Two different polynomials $P_i, P_j, i \neq j$ do not vanish at the same time, i.e., they have distinct roots. If $i = j + 1$ then the roots are interlaced

$$z_1 < y_1 < \cdots < z_j < y_j < z_i,$$

where $y_1, \dots, y_j$ are the roots of P_j and $z_1, \dots, z_i$ are the roots of P_i [Szegő, 1975, Theorem 3.3.2, p.46]

The $\{P_k\}$ lead to the polynomial [Freud, 1969, p.21]

$$\psi_{n+1}(x; \eta, \mathbf{m}) = P_{n+1}(x; \mathbf{m})P_n(\eta; \mathbf{m}) - P_{n+1}(\eta; \mathbf{m})P_n(x; \mathbf{m}). \quad (\text{A.5})$$

If η is a root of P_n , ψ_{n+1} has degree n : P_{n+1} cannot vanish at the same time because it's orthogonal to P_n and P_n has degree n by definition. If η is not a root of P_n , ψ_{n+1} has degree $n + 1$. In either case, the following result holds:

Theorem 9 ([Akhiezer, 1965, Theorem 1.2.2, p.10]). *All roots of $\psi_{n+1}(x; \eta, \mathbf{m})$ are real and simple.*

The roots of ψ_{n+1} are related to the roots of P_n :

Theorem 10 ([Tari, 2005, Theorem A.39]). *If η is not a root of P_n , the roots $x_0, \dots, x_n$ of ψ_{n+1} and the roots $y_1, \dots, y_n$ of P_n interlace*

$$x_0 < y_1 < x_1 < \cdots < x_{n-1} < y_n < x_n.$$

and the significance of ψ_{n+1} is its relation to the kernel K :

Theorem 11 (Christoffel–Darboux theorem, [Szegő, 1975, Theorem 3.2.2., p.43]). *The kernel K and ψ_{n+1} are related by*

$$K(x; \eta, \mathbf{m}) \propto \frac{\psi_{n+1}(x; \eta, \mathbf{m})}{(x - \eta)}.$$

Which yields the following characterization of the roots of K :

Corollary 4. *The kernel polynomial K has the same roots as ψ_{n+1} , except for η , thus all $\{x_i\}$ are real and simple. Either η is not a root of P_n , then ψ_{n+1} has $n + 1$ real and simple roots which are the points $x_1, \dots, x_n$ and η . Or η is a root of P_n , then ψ_{n+1} has the root η and $n - 1$ other roots $x_1, \dots, x_{n-1}$.*

A.3.2 Intermediate Statements

For the main results, some properties regarding differentiability are required. These are proven here.

Lemma 3. *The Hankel matrix $\mathbf{H}$ is continuously differentiable in the moments and, if the moments are strictly positive, its inverse $\mathbf{H}^{-1}$ is also continuously differentiable.*

Proof. The moments are the entries of the Hankel matrix, so the first claim is trivially true. For the second claim, we have $\mathbf{H}^{-1} = \frac{1}{\det(\mathbf{H})} \text{adj}(\mathbf{H})$. The entries of the adjugate $\text{adj}(\mathbf{H})$ and the determinant $\det(\mathbf{H})$ are polynomials in the entries of $\mathbf{H}$. By Theorem 4 from strict positivity follows $\det(\mathbf{H}) > 0$, so the second claim holds. $\square$

The orthogonal polynomials depend smoothly on the moments:

Lemma 4. *The coefficients of the orthogonal polynomials $P_k(x; \mathbf{m})$ are infinitely differentiable in the moments $\mathbf{m}$.*

Proof. Expanding along the last row of the determinant form given in Definition 2, we get

$$P_k(x; \mathbf{m}) = \sum_{j=0}^n (-1)^{k+j} x^j \det(\mathbf{M}_{k,j}), \quad (\text{A.6})$$

where $\det(\mathbf{M}_{k,j})$ is the (k, j) minor of the original matrix. The result is the canonical form of the orthogonal polynomial P_k with its coefficients being the determinants of matrices $\mathbf{M}_{k,j}$ formed from the set of moments. Since the determinant is a polynomial of the matrix coefficients, the minors are infinitely differentiable in the moments and so is P_k . $\square$

It is well known that, if the roots of a polynomial are real and simple, they are continuously differentiable functions of the coefficients. For orthogonal polynomials, the roots are real and simple, which gives the following result:

Corollary 5. *The n roots of the orthogonal polynomials $P_k(x; \mathbf{m})$ are continuously differentiable functions in the moments.*

These observations extend to the roots of ψ_{n+1} :

Lemma 5. *If $P_n(\eta; \mathbf{m}) \neq 0$, the $n + 1$ roots of $\psi_{n+1}(x; \eta, \mathbf{m})$ are continuously differentiable functions in η and $\mathbf{m}$. If $P_n(\eta; \mathbf{m}) = 0$, the n roots of $\psi_{n+1}(x; \eta, \mathbf{m})$ are continuously differentiable in η and $\mathbf{m}$.*

Proof. Recall that $\psi_{n+1}(x; \eta) = P_{n+1}(x)P_n(\eta) - P_{n+1}(\eta)P_n(x)$ (Eq. A.5). Lemma 4 states that P_n and P_{n+1} are infinitely differentiable in η and $\mathbf{m}$, and consequently, so are the coefficients of ψ_{n+1} . Therefore all real and simple roots of ψ_{n+1} are continuously differentiable in η and $\mathbf{m}$. $\square$

A.3.3 Proof of Theorem 5

Proof. Without loss of generality, we consider the lower bound $L(\eta; \mathbf{m})$. Choose any point $(\eta_0, \mathbf{m}_0) \in \Omega$. According to Theorem 2, a unique canonical representation exists with points $\{x_i(\eta_0; \mathbf{m}_0)\}$ and η_0 where the points can be ordered accordingly

$$x_0 < \cdots < x_{k-1} < x_k (= \eta_0) < x_{k+1} < \cdots < x_n. \quad (\text{A.7})$$

Take another point $(\eta_1, \mathbf{m}_1) \in \Omega$ in the domain. Without loss of generality, assume that at $(\eta_1, \mathbf{m}_1)$ more of the $\{x_i\}$ are larger than η_1 , i.e., the number of terms in $L(\eta_0; \mathbf{m}_0)$ is greater than in $L(\eta_1; \mathbf{m}_1)$.

The moments $\mathbf{m}_0$ and $\mathbf{m}_1$ are in the convex cone of the moment curve and η_0 and η_1 are in $\mathbb{R}$, so the points can be connected with an arbitrary, continuously differentiable curve

$$\boldsymbol{\gamma}(\tau) = (\eta(\tau), \mathbf{m}(\tau)) \in \mathbb{R} \times \mathbb{R}^{2n+1}, \quad (\text{A.8})$$

with $\boldsymbol{\gamma}(0) = (\eta_0, \mathbf{m}_0)$ and $\boldsymbol{\gamma}(1) = (\eta_1, \mathbf{m}_1)$.

Let us assume the opposite of the claim: for all $\tau \in (0, 1)$ the curve is part of the domain, i.e., $\boldsymbol{\gamma}(\tau) \in \Omega$, so that at no point is $\eta(\tau)$ a root of P_n and the moments $\mathbf{m}(\tau)$ are strictly positive.

The polynomial $\psi_{n+1}(x; \eta(\tau), \mathbf{m}(\tau))$, therefore, has $n + 1$ real and simple roots along the curve, which are, according to Corollary 4, the $x_0, \dots, x_n$, including η . The order of roots at $(\eta_0, \mathbf{m}_0)$ identifies each root uniquely and we can define a function

$$g_i(\tau) = x_i(\eta(\tau); \mathbf{m}(\tau)) - \eta(\tau), \quad (\text{A.9})$$

which indicates if root x_i is greater or smaller than η . According to Lemma 5, the roots are continuously differentiable functions for $\tau \in [0, 1]$ and so are the functions g_i .

The number of points greater than η is larger at η_1 than at η_0 , so there is at least one $x_j \neq \eta_0$ such that

$$g_j(0) < 0 \quad \text{and} \quad g_j(1) > 0, \quad (\text{A.10})$$

i.e., x_j “switches” places with η on the path. Since $\boldsymbol{\gamma}$ is continuous and g_i is continuous, according to the intermediate value theorem, there exists one $\tau' \in (0, 1)$ where

$$g_j(\tau') = 0, \quad (\text{A.11})$$

which however is a contradiction because along the curve the $n + 1$ roots of ψ_{n+1} are real and simple. Therefore, at some point along $\boldsymbol{\gamma}$, either the sequence of moments $\mathbf{m}(\tau)$ is not strictly positive or $\eta(\tau)$ is a root of $P_n(x; \mathbf{m}(\tau))$. $\square$

A.3.4 Proof of Theorem 6

Proof. Recall that, according to Equation 4.17, the weights are

$$w_i = \frac{1}{\mathbf{x}_i^\top \mathbf{H}^{-1} \mathbf{x}_i}.$$

The vectors $\mathbf{x}_i$ consist of monomials in the roots x_i (Equation 4.14) and the roots are continuously differentiable (Lemma 5). $\mathbf{H}^{-1}$ is continuously differentiable (Lemma 3). Since $\mathbf{m}$ is strictly positive, $\mathbf{H}$ is positive definite and the denominator is > 0 . The claim now follows from the properties of function composition and differentiability of matrix multiplication. $\square$

A.3.5 Proof of Theorem 7

Proof. Without loss of generality, assume $(\eta_0, \mathbf{m}_0) \in \Omega$ such that $x_j(= \eta_0) > x_0$ approaches y_j , as the other case can be treated equally. The roots $\{x_i(\eta, \mathbf{m})\}_{i=0}^n$ of the polynomial ψ_{n+1} (Corollary 4) and the roots $\{y_i(\mathbf{m})\}_{i=1}^n$ of the polynomial P_n strictly interlace at $(\eta_0, \mathbf{m}_0)$ (Theorem 10).

Let $(\eta_1, \mathbf{m}_1) \in \Lambda$ be a point on the singularity, such that $P_n(\eta_1, \mathbf{m}_1) = 0$. The roots of P_n are continuously differentiable functions in $(\eta, \mathbf{m})$ (Lemma 5), so the order established at $(\eta_0, \mathbf{m}_0)$ is maintained along a continuously differentiable path from $(\eta_0, \mathbf{m}_0)$ to $(\eta_1, \mathbf{m}_1)$. Each y_j can be uniquely identified from which follows that $y_j(\mathbf{m}_0) < x_j(\eta_0, \mathbf{m}_0)(= \eta_0) < y_{j+1}(\mathbf{m}_0)$ and $y_j(\mathbf{m}_1) = \eta_1$.

At $(\eta_1, \mathbf{m}_1) \in \Lambda$, the roots of ψ_{n+1} are the n roots of P_n (see Eq. (A.5)). The roots of ψ_{n+1} are continuously differentiable functions of η and $\mathbf{m}$ (Lemma 5), and in particular they are at $(\eta_1, \mathbf{m}_1)$. Therefore, if the path is reversed, it is a continuously differentiable path from the roots of $P_n(\mathbf{m}_1)$ to n of the roots of $\psi_{n+1}(x; \eta_0, \mathbf{m}_0)$.

By construction, $x_j(= \eta_0)$ follows a continuously differentiable path to $y_j(= \eta_1)$, so its reverse is a continuously differentiable path from y_j to x_j . If y_{j+1} exists, it must correspond to the root x_{j+1} : it cannot correspond to any root $\leq x_j$ as the roots of ψ_{n+1} are simple on the path (Lemma 9) and it cannot correspond to any root $\geq x_{j+2}$ because the x_i and y_i strictly interlace on the path and there is $x_{j+1} < y_{j+2} < x_{j+2}$. A similar argument holds for y_{j-1} if it exists. This establishes an order such that every y_j at $(\eta_1, \mathbf{m}_1)$ corresponds to x_j at $(\eta_0, \mathbf{m}_0)$.

Therefore, the only root without a corresponding y at $(\eta_1, \mathbf{m}_1)$ is x_0 . ψ_{n+1} is a degree $n+1$ polynomial that is reduced to a degree n polynomial on a continuously differentiable path from $(\eta_0, \mathbf{m}_0)$ to $(\eta_1, \mathbf{m}_1)$, so its leading coefficient a_{n+1} is taken to zero on the path. As a_{n+1} vanishes, the roots $x_1 < \dots < x_n$ remain finite, and therefore the root x_0 is unbounded and must go to $-\infty$. $\square$

A.3.6 Proof of Lemma 1

Proof. Recall the definition of the weights $w_k = \frac{1}{\mathbf{x}_k^\top \mathbf{H}^{-1} \mathbf{x}_k}$. Since the moment-generating measures are finite, the moments are finite, and therefore, there is an upper bound on the eigenvalues of $\mathbf{H}$. Consequently, there is a lower bound on the eigenvalues of $\mathbf{H}^{-1}$. $\mathbf{H}$ is positive-definite for strictly positive moments (Theorem 4) and so is $\mathbf{H}^{-1}$. As $x_k \rightarrow \pm\infty$, the norm grows $\|\mathbf{x}_k\|_2 \rightarrow \infty$, from which follows that $\mathbf{x}_k^\top \mathbf{H}^{-1} \mathbf{x}_k \rightarrow \infty$ and $w_k \rightarrow 0$. $\square$

A.3.7 Proof of Lemma 2

Proof. We first prove the case for the derivative $\frac{\partial w_k}{\partial \eta}$, which by the chain rule is

$$\frac{\partial w_k}{\partial \eta} = \frac{\partial w_k}{\partial \mathbf{x}_k} \frac{\partial \mathbf{x}_k}{\partial x_k} \frac{\partial x_k}{\partial \eta}, \quad (\text{A.12})$$

since $\mathbf{H}^{-1}$ is independent of η . The point x_k is a root of the kernel polynomial K , which is explicitly given as (Eq. (4.16))

$$K(x) = \mathbf{x}^\top \mathbf{H}^{-1} \boldsymbol{\eta}. \quad (\text{A.13})$$

The implicit function theorem relates η to the root

$$\frac{\partial x_k}{\partial \eta} = -\frac{\mathbf{x}_k^\top \mathbf{H}^{-1} \frac{\partial \eta}{\partial \eta}}{\boldsymbol{\eta}^\top \mathbf{H}^{-1} \frac{\partial \mathbf{x}_k}{\partial x_k}}. \quad (\text{A.14})$$

The other factor can be directly expressed using Eq. (4.17)

$$\frac{\partial w_k}{\partial \mathbf{x}_k} = -2 \frac{\mathbf{x}_k^\top \mathbf{H}^{-1}}{(\mathbf{x}_k^\top \mathbf{H}^{-1} \mathbf{x}_k)^2}. \quad (\text{A.15})$$

The complete derivative therefore is

$$\frac{\partial w_k}{\partial \eta} = 2 \frac{\mathbf{x}_k^\top \mathbf{H}^{-1} \frac{\partial \mathbf{x}_k}{\partial x_k} \mathbf{x}_k^\top \mathbf{H}^{-1} \frac{\partial \eta}{\partial \eta}}{(\mathbf{x}_k^\top \mathbf{H}^{-1} \mathbf{x}_k)^2 \boldsymbol{\eta}^\top \mathbf{H}^{-1} \frac{\partial \mathbf{x}_k}{\partial x_k}} \quad (\text{A.16})$$

The derivative is a rational function in x_k and the limit is determined through the (ratio of) degrees of the numerator and the denominator. We assume full degree in the numerator ($3n - 1$).

Notice that the point $(\eta, \mathbf{m})$ is bounded and finite, and consequently so are $\mathbf{H}^{-1}$, $\boldsymbol{\eta}$, and $\frac{\partial \eta}{\partial \eta}$. The quadratic part in the denominator is positive because the eigenvalues of $\mathbf{H}^{-1}$ are bounded from below ($\mathbf{m}$ is strictly positive) and the quadratic part necessarily has full degree ($4n$) because its leading coefficient is the leading coefficient of full-degree P_n (the last row/column of $\mathbf{H}^{-1}$ are the coefficients of P_n [Tari, 2005, Lemma A.26]). The second part of the denominator, $K'(x_k) = \boldsymbol{\eta}^\top \mathbf{H}^{-1} \frac{\partial \mathbf{x}_k}{\partial x_k}$, is the derivative of the kernel at its root x_k . The kernel roots are real and simple inside Ω , so the derivative is non-zero in the domain. At a singularity, the kernel has degree $n - 1$ and the derivative reduces to degree $n - 2$, with non-zero coefficient c_{n-1} (Corollary 4). For the case $n = 1$, the derivative is the vanishing coefficient c_1 , which can be reparameterized in the root as $c_1 = -\frac{c_0}{x_k}$. Since $\mathbf{H}^{-1}$ has full rank and $[c_0 \ c_1] = \boldsymbol{\eta}^\top \mathbf{H}^{-1}$, the coefficient c_0 cannot vanish at the same time as c_1 . The denominator therefore has minimal degree $5n - 2$.

We bound the derivative from above, assuming full degree in the numerator and minimal degree in the denominator:

$$\left| \frac{\partial w_k}{\partial \eta} \right| \leq c \left| \frac{x_k^{3n-1}}{x_k^{5n-2}} \right|, \quad c > 0. \quad (\text{A.17})$$

The upper bound goes to 0 as $x_k \rightarrow \pm\infty$ and so does $\frac{\partial w_k}{\partial \eta}$. $\square$

Proof. We continue with the derivative $\frac{\partial w_k}{\partial \mathbf{m}}$, and consider only a single moment m_i ($0 \leq i \leq 2n$) without loss of generality. By the chain rule the derivative is

$$\frac{\partial w_k}{\partial m_i} = \left\langle \frac{\partial w_k}{\partial \mathbf{H}^{-1}}, \frac{\partial \mathbf{H}^{-1}}{\partial m_i} \right\rangle + \frac{\partial w_k}{\partial \mathbf{x}_k} \frac{\partial \mathbf{x}_k}{\partial x_k} \frac{\partial x_k}{\partial m_i}, \quad (\text{A.18})$$

where $\langle \mathbf{A}, \mathbf{B} \rangle$ denotes the sum of the element-wise matrix product. Using the implicit function theorem for the kernel, we get

$$\frac{\partial x_k}{\partial m_i} = - \left\langle \frac{\mathbf{x}_k \boldsymbol{\eta}^\top}{\boldsymbol{\eta}^\top \mathbf{H}^{-1} \frac{\partial \mathbf{x}_k}{\partial x_k}}, \frac{\partial \mathbf{H}^{-1}}{\partial m_i} \right\rangle \quad (\text{A.19})$$

and from Eq. 4.17 we also have

$$\frac{\partial w_k}{\partial \mathbf{H}^{-1}} = -\frac{\mathbf{x}_k \mathbf{x}_k^\top}{(\mathbf{x}_k^\top \mathbf{H}^{-1} \mathbf{x}_k)^2}, \quad (\text{A.20})$$

which yields the full derivative

$$\frac{\partial w_k}{\partial m_i} = \underbrace{\left\langle -\frac{\mathbf{x}_k \mathbf{x}_k^\top}{(\mathbf{x}_k^\top \mathbf{H}^{-1} \mathbf{x}_k)^2} + 2 \frac{\mathbf{x}_k^\top \mathbf{H}^{-1} \frac{\partial \mathbf{x}_k}{\partial x_k} \mathbf{x}_k \boldsymbol{\eta}^\top}{(\mathbf{x}_k^\top \mathbf{H}^{-1} \mathbf{x}_k)^2 \boldsymbol{\eta}^\top \mathbf{H}^{-1} \frac{\partial \mathbf{x}_k}{\partial x_k}}, \frac{\partial \mathbf{H}^{-1}}{\partial m_i} \right\rangle}_{\mathbf{A}}. \quad (\text{A.21})$$

The second argument, $\frac{\partial \mathbf{H}^{-1}}{\partial m_i}$, remains finite and the result is a linear combination of the entries of the matrix $\mathbf{A} \in \mathbb{R}^{(n+1) \times (n+1)}$. So, we now study the entries of the matrix $\mathbf{A}$ and their limit behavior.

Expanding the terms to the same denominator gives

$$\mathbf{A} = \frac{-\mathbf{x}_k \mathbf{x}_k^\top \boldsymbol{\eta}^\top \mathbf{H}^{-1} \frac{\partial \mathbf{x}_k}{\partial x_k} + 2 \mathbf{x}_k^\top \mathbf{H}^{-1} \frac{\partial \mathbf{x}_k}{\partial x_k} \mathbf{x}_k \boldsymbol{\eta}^\top}{(\mathbf{x}_k^\top \mathbf{H}^{-1} \mathbf{x}_k)^2 \boldsymbol{\eta}^\top \mathbf{H}^{-1} \frac{\partial \mathbf{x}_k}{\partial x_k}}. \quad (\text{A.22})$$

Each entry $A_{i,j}$ of the matrix $\mathbf{A} = \{A_{i,j}\}_0^n$ is a rational function in x_k , with the same denominator as $\frac{\partial w_k}{\partial \eta}$ (minimal degree $5n-2$). The entry $A_{n,n}$ maximizes the degree of the numerator, so, in the limit, it gives an upper bound on all other entries. By assuming full degree in the numerator and minimal degree in the denominator, we get an upper bound on $A_{n,n}$:

$$|A_{n,n}| \leq c \left| \frac{x_k^{3n-1}}{x_k^{5n-2}} \right|, \quad c > 0. \quad (\text{A.23})$$

The upper bound goes to 0 as $x_k \rightarrow \pm\infty$ and so does $\frac{\partial w_k}{\partial m_i}$. $\square$

APPENDIX B

Radiative Backpropagation with Non-Static Geometry

B.1 An Attached Approach for Detached Sampling

Vicini et al. [2021] consider the recursion of $\partial_p L_o$ in Eq. (5.5) for the attached case of PRB but with different reasoning. The extended version of Eq. (5.7), without the emission L_e , reads [Vicini et al., 2021, Supplementary Material]:

$$\begin{aligned} \partial_\theta L_o(\mathbf{p}, \omega) = & \int_{\mathcal{U}^2} L_i(\mathbf{p}, T(\mathbf{u}, \theta)) \partial_\theta \left[\frac{f(T(\mathbf{u}, \theta), \theta)}{p(T(\mathbf{u}, \theta), \theta)} \right] \\ & + \partial_\theta [L_i(\mathbf{p}, T(\mathbf{u}, \theta))] \frac{f(T(\mathbf{u}, \theta), \theta)}{p(T(\mathbf{u}, \theta), \theta)} \\ & + \partial_\omega L_i(\mathbf{p}, T(\mathbf{u}, \theta)) \partial_\theta [T(\mathbf{u}, \theta)] \frac{f(T(\mathbf{u}, \theta), \theta)}{p(T(\mathbf{u}, \theta), \theta)} d\mathbf{u}. \end{aligned} \quad (\text{B.1})$$

The reparameterization T introduces a new term that involves $\partial_\omega L_i(\mathbf{p}, \omega)$, which can be expanded using the transport equation

$$\partial_\omega L_i(\mathbf{p}, \omega) = \partial_\omega L_o(\mathbf{r}(\mathbf{p}, \omega_0), \omega) + \partial_p L_o(\mathbf{r}(\mathbf{p}_0, \omega_0), \omega) \partial_\omega \mathbf{r}(\mathbf{p}, \omega), \quad (\text{B.2})$$

where $\omega_0 = \omega$ and $\mathbf{p}_0 = \mathbf{p}$. For attached sampling, one must therefore compute $\partial_p L_o$ from Eq. (5.4), not because geometry moves but because the sample directions depend on θ and so do the intersection points.

Vicini et al. [2021] compute $\partial_p L_o$ using forward-mode differentiation, and, by our previous arguments, this is also required in the detached case if geometry is non-static. In their Jacobian framework, the Monte Carlo integration can be written as

$$h^{(N)}(\theta, L_0, \beta_0, \omega_0, \mathbf{p}_0), \text{ with} \quad (\text{B.3})$$

$$h(\theta, L, \beta, \omega, \mathbf{p}) = [\theta, L + \beta L_e, \beta f, \omega_i, \mathbf{r}(\mathbf{p}, \omega)], \quad (\text{B.4})$$

where N is the maximum path depth and $(L_0 = 0, \beta_0 = 1, \omega_0, \mathbf{p}_0)$ is the start configuration. The PDF p is omitted for clarity and because it does not participate in differentiation in the detached case. For the derivative of each intersection (i.e., invocation of h) one has the reduced Jacobian (reduced as it ignores θ)

$$J_h = \begin{bmatrix} 1 & L_e & \beta \partial_\omega L_e & \beta \partial_p L_e \\ 0 & f & \beta \partial_\omega f & \beta \partial_p f \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \partial_p \mathbf{r} \end{bmatrix}. \quad (\text{B.5})$$

Given that Vicini et al. [2021] parametrize $(\mathbf{p}, \boldsymbol{\omega})$ as a single ray, this resembles the code presented in their Listing 3. To convert the pseudocode to a detached estimator for non-static geometry, one only needs to detach the pdf and $\boldsymbol{\omega}_i (= \boldsymbol{\omega}'$ in Listing 3). Unfortunately, there is no publicly available implementation, that could be used for validation, and we did not re-implement the approach.

B.2 Derivation of the Three-Point Form Pseudocode

To verify that all terms of the derivative are computed, we take the three-point form

$$\begin{aligned} L_o(\mathbf{p}, \boldsymbol{\omega}, \theta) &= L_e(\mathbf{p}, \boldsymbol{\omega}, \theta) \\ &+ \int_{\mathcal{M}} L_i(\mathbf{p}, \boldsymbol{\omega}'(\mathbf{p}, \mathbf{p}', \theta), \theta) \\ &\quad f_s(\mathbf{p}, \boldsymbol{\omega}, \boldsymbol{\omega}'(\mathbf{p}, \mathbf{p}', \theta), \theta) D(\mathbf{p}, \mathbf{p}', \theta) d\mathbf{p}', \end{aligned} \quad (\text{B.6})$$

with the direction $\boldsymbol{\omega}'(\mathbf{p}, \mathbf{p}', \theta) = S(\mathbf{p}, \theta) \rightarrow S(\mathbf{p}', \theta)$ and the cosine-weighted BSDF $f_s(\mathbf{p}, \boldsymbol{\omega}, \boldsymbol{\omega}', \theta) = f(\mathbf{p}, \boldsymbol{\omega}, \boldsymbol{\omega}', \theta) |\mathbf{n}(\mathbf{p}, \theta)^\top \boldsymbol{\omega}'|$. As we now sample surface points $\mathbf{p}'$ and construct the outgoing direction $\boldsymbol{\omega}'$, the transport equation simplifies to

$$\begin{aligned} L_i(\mathbf{p}, \boldsymbol{\omega}'(\mathbf{p}, \mathbf{p}', \theta), \theta) &= \\ &L_o(\mathbf{r}(\mathbf{p}, \boldsymbol{\omega}'(\mathbf{p}, \mathbf{p}', \theta), \theta), \boldsymbol{\omega}'(\mathbf{p}', \mathbf{p}, \theta), \theta) \\ &= L_o(\mathbf{p}', \boldsymbol{\omega}'(\mathbf{p}', \mathbf{p}, \theta), \theta), \end{aligned} \quad (\text{B.7})$$

as $\mathbf{r}(\mathbf{p}, \boldsymbol{\omega}'(\mathbf{p}, \mathbf{p}', \theta), \theta) = \mathbf{p}'$. Thus, the reparameterization eliminates the dependency on θ for $\mathbf{p}'$, but introduces a new dependency on θ for the direction $\boldsymbol{\omega}'$ (compare with Eq. (2.6)). The derivative with respect to θ is therefore given by

$$\begin{aligned} \partial_\theta L_o(\mathbf{p}, \boldsymbol{\omega}, \theta) &= \partial_\theta L_e(\mathbf{p}, \boldsymbol{\omega}, \theta) \\ &+ \int_{\mathcal{M}} \partial_\theta L_o(\mathbf{p}', \boldsymbol{\omega}'(\mathbf{p}', \mathbf{p}, \theta), \theta) f_s D(\mathbf{p}, \mathbf{p}', \theta) \\ &\quad + L_o(\mathbf{p}', \boldsymbol{\omega}'(\mathbf{p}', \mathbf{p}, \theta), \theta) \partial_\theta f_s D(\mathbf{p}, \mathbf{p}', \theta) \\ &\quad + L_o(\mathbf{p}', \boldsymbol{\omega}'(\mathbf{p}', \mathbf{p}, \theta), \theta) f_s \partial_\theta D(\mathbf{p}, \mathbf{p}', \theta) \\ &\quad d\mathbf{p}', \end{aligned} \quad (\text{B.8})$$

where $f_s = f_s(\mathbf{p}, \boldsymbol{\omega}, \boldsymbol{\omega}'(\mathbf{p}, \mathbf{p}', \theta), \theta)$. The consecutive derivative of L_o reveals the dependency on θ for the direction $\boldsymbol{\omega}$:

$$\begin{aligned} \partial_\theta L_o(\mathbf{p}', \boldsymbol{\omega}'(\mathbf{p}', \mathbf{p}, \theta), \theta) &= \partial_\theta L_o(\mathbf{p}', \boldsymbol{\omega}'(\mathbf{p}', \mathbf{p}, \theta_0), \theta) \\ &+ \partial_\omega L_o(\mathbf{p}', \boldsymbol{\omega}'(\mathbf{p}', \mathbf{p}, \theta), \theta) \partial_\theta [\boldsymbol{\omega}'(\mathbf{p}', \mathbf{p}, \theta)]. \end{aligned} \quad (\text{B.9})$$

We therefore need the derivative of L_o with respect to $\boldsymbol{\omega}$:

$$\begin{aligned} \partial_\omega L_o(\mathbf{p}, \boldsymbol{\omega}, \theta) &= \partial_\omega L_e(\mathbf{p}, \boldsymbol{\omega}, \theta) \\ &+ \int_{\mathcal{M}} L_o(\mathbf{p}', \boldsymbol{\omega}'(\mathbf{p}', \mathbf{p}, \theta), \theta) \partial_\omega f_s(\mathbf{p}, \boldsymbol{\omega}, \boldsymbol{\omega}'(\mathbf{p}, \mathbf{p}', \theta), \theta) \\ &\quad D(\mathbf{p}, \mathbf{p}', \theta) d\mathbf{p}'. \end{aligned} \quad (\text{B.10})$$

This reveals the locality enabled by the reparameterization as mentioned in Section 5.2.2: the derivative $\partial_{\omega}L_o$ is not recursive in contrast to $\partial_p L_o$.

In `sample_path_adjoint` in Listing 1, we keep track of the three surface points that affect the path geometry around the current intersection point (i.e., previous $\mathbf{p}_p$, current $\mathbf{p}_c$, and next $\mathbf{p}_n$) and accumulate the contribution of a single position sample over three loop iterations: within a single iteration of the for loop, we construct the direction ω , attached to the AD graph. In this way, we calculate $\partial_{\theta}L_e(\mathbf{p}, \omega)$ and $\partial_{\omega}L_e(\mathbf{p}, \omega)\partial_{\theta}[\omega'(\mathbf{p}', \mathbf{p}, \theta)]$ in line 10. The same is true for the two derivative terms related to the BSDF ($\partial_{\theta}f_s$ and $\partial_{\omega}f_s$) in line 24f. The term containing the derivative of the determinant is calculated in line 14. In the next iteration, the recursive term $\partial_{\theta}L_o$ is evaluated with the updated ω . Therefore, all terms of Eq. (B.9) are computed by `sample_path_adjoint`.

APPENDIX C

Time-Resolved Path Replay Backpropagation

C.1 Time-Resolved PRB in the Three-Point Form

The primary integrators in Chapter 6 (AcousticPRB and AcousticAD) are based on the *three-point form* of the rendering equation. The time-dependent three-point form is obtained by reparameterizing the rendering equation in advanced time (Eq. (6.6)) from an integral over the (hemi) sphere to an integral over surfaces:

$$L_o(\mathbf{p}, \boldsymbol{\omega}, T, \theta) = L_e(\mathbf{p}, \boldsymbol{\omega}, T, \theta) + \int_{\mathcal{M}} L_i(\mathbf{p}, \boldsymbol{\omega}'(\mathbf{p}, \mathbf{p}', \theta), T, \theta) f_s(\mathbf{p}, \boldsymbol{\omega}, \boldsymbol{\omega}'(\mathbf{p}, \mathbf{p}', \theta), \theta) D(\mathbf{p}, \mathbf{p}', \theta) d\mathbf{p}', \quad (\text{C.1})$$

with $\boldsymbol{\omega}'(\mathbf{p}, \mathbf{p}', \theta) = S(\mathbf{p}, \theta) \rightarrow S(\mathbf{p}', \theta)$, where $\mathcal{M}$ is a (hypothetical) global parametrization of all surfaces in the scene, $S(\mathbf{p}, \theta)$ is a mapping from the parametrization to the ambient space $\mathbb{R}^3$, $\mathbf{y} \rightarrow \mathbf{x}$ is the normalized vector pointing from $\mathbf{y}$ to $\mathbf{x}$, f_s is the cosine-weighted BSDF (c.f. Appendix B.2) and D consists of the visibility term V and the Jacobian determinant of the reparameterization to a surface integral:

$$D(\mathbf{p}, \mathbf{p}', \theta) = V(S(\mathbf{p}, \theta), S(\mathbf{p}', \theta), \theta) \frac{|\mathbf{n}(\mathbf{p}', \theta)^\top \boldsymbol{\omega}'(\mathbf{p}', \mathbf{p}, \theta)|}{\|S(\mathbf{p}, \theta) - S(\mathbf{p}', \theta)\|^2} |\partial_{\mathbf{p}_u} S(\mathbf{p}', \theta) \times \partial_{\mathbf{p}_v} S(\mathbf{p}', \theta)|, \quad (\text{C.2})$$

with $\mathbf{n}$ being the surface normal at the given (surface) point. Note that we are omitting the parameter t for brevity. The reparameterization results in a slightly different transport equation, as we sample surface points $\mathbf{p}'$ and construct the outgoing direction $\boldsymbol{\omega}'$:

$$\begin{aligned} L_i(\mathbf{p}, \boldsymbol{\omega}'(\mathbf{p}, \mathbf{p}', \theta), T, \theta) \\ &= L_o\left(\mathbf{r}(\mathbf{p}, \boldsymbol{\omega}'(\mathbf{p}, \mathbf{p}', \theta), \theta), \boldsymbol{\omega}'(\mathbf{p}', \mathbf{p}, \theta), T + \tau(\mathbf{p}, \mathbf{r}(\mathbf{p}, \boldsymbol{\omega}'(\mathbf{p}, \mathbf{p}', \theta), \theta), \theta), \theta\right) \\ &= L_o(\mathbf{p}', \boldsymbol{\omega}'(\mathbf{p}', \mathbf{p}, \theta), T + \tau(\mathbf{p}, \mathbf{p}', \theta), \theta), \end{aligned} \quad (\text{C.3})$$

as $\mathbf{r}(\mathbf{p}, \boldsymbol{\omega}'(\mathbf{p}, \mathbf{p}', \theta), \theta) = \mathbf{p}'$. Note that $\mathbf{p}$ and $\mathbf{p}'$ are defined on the surface therefore $\tau(\mathbf{p}, \mathbf{p}', \theta) = \frac{\|S(\mathbf{p}, \theta) - S(\mathbf{p}', \theta)\|}{c}$

Assuming the integrand is continuous in θ , this results in the following derivative

with respect to θ :

$$\begin{aligned} \partial_\theta L_o(\mathbf{p}, \boldsymbol{\omega}, T, \theta) &= \partial_\theta L_e(\mathbf{p}, \boldsymbol{\omega}, T, \theta) \\ &+ \int_{\mathcal{M}} \partial_\theta L_o(\mathbf{p}, \boldsymbol{\omega}'(\mathbf{p}', \mathbf{p}, \theta), T + \tau(\mathbf{x}, \mathbf{p}', \theta), \theta) \quad f_s(\mathbf{p}, \boldsymbol{\omega}, \boldsymbol{\omega}'(\mathbf{p}, \mathbf{p}', \theta), \theta) \quad D(\mathbf{p}, \mathbf{p}', \theta) \\ &\quad + L_o(\mathbf{p}, \boldsymbol{\omega}'(\mathbf{p}', \mathbf{p}, \theta), T + \tau(\mathbf{x}, \mathbf{p}', \theta), \theta) \quad \partial_\theta f_s(\mathbf{p}, \boldsymbol{\omega}, \boldsymbol{\omega}'(\mathbf{p}, \mathbf{p}', \theta), \theta) \quad D(\mathbf{p}, \mathbf{p}', \theta) \\ &\quad + L_o(\mathbf{p}, \boldsymbol{\omega}'(\mathbf{p}', \mathbf{p}, \theta), T + \tau(\mathbf{x}, \mathbf{p}', \theta), \theta) \quad f_s(\mathbf{p}, \boldsymbol{\omega}, \boldsymbol{\omega}'(\mathbf{p}, \mathbf{p}', \theta), \theta) \quad \partial_\theta D(\mathbf{p}, \mathbf{p}', \theta) \, d\mathbf{p}' \end{aligned} \quad (\text{C.4})$$

As $\boldsymbol{\omega}'$ and τ depend on θ , it is necessary to apply the multivariate chain rule for $\partial_\theta f_s$ and $\partial_\theta L_o$. While the term for $\partial_\theta D$ and $\partial_\theta f_s$ is straightforward to calculate (e.g. using (local) automatic differentiation), evaluating $\partial_\theta L_o$ is more involved:

$$\begin{aligned} \partial_\theta L_o(\mathbf{p}, \boldsymbol{\omega}'(\mathbf{p}, \mathbf{p}', \theta), T + \tau(\mathbf{x}, \mathbf{p}', \theta), \theta) \\ = [\partial_\omega L_o](\mathbf{p}, \boldsymbol{\omega}'(\mathbf{p}, \mathbf{p}', \theta), T + \tau(\mathbf{x}, \mathbf{p}', \theta), \theta) \quad \partial_\theta \boldsymbol{\omega}'(\mathbf{p}, \mathbf{p}', \theta) \\ + [\partial_T L_o](\mathbf{p}, \boldsymbol{\omega}'(\mathbf{p}, \mathbf{p}', \theta), T + \tau(\mathbf{x}, \mathbf{p}', \theta), \theta) \quad \partial_\theta \tau(\mathbf{p}, \mathbf{p}', \theta) \\ + [\partial_\theta L_o](\mathbf{p}, \boldsymbol{\omega}'(\mathbf{p}, \mathbf{p}', \theta), T + \tau(\mathbf{x}, \mathbf{p}', \theta), \theta), \end{aligned} \quad (\text{C.5})$$

The last summand defines the recursion. The second summand can be computed similarly to the approach described in the main paper, where $\partial_\theta \tau$ is obtained using automatic differentiation. The differential in the first summand can be expanded as:

$$\begin{aligned} \partial_\omega L_o(\mathbf{p}, \boldsymbol{\omega}, T, \theta) &= \partial_\omega L_e(\mathbf{p}, \boldsymbol{\omega}, T, \theta) + \\ &\int_{\mathcal{M}} L_o(\mathbf{p}', \boldsymbol{\omega}'(\mathbf{p}', \mathbf{p}, \theta), T + \tau(\mathbf{p}, \mathbf{p}', \theta), \theta) \quad \partial_\omega f_s(\mathbf{p}, \boldsymbol{\omega}, \boldsymbol{\omega}'(\mathbf{p}, \mathbf{p}', \theta), \theta) \quad D(\mathbf{p}, \mathbf{p}', \theta) \, d\mathbf{p}'. \end{aligned} \quad (\text{C.6})$$

As noted in Chapter 5, the product of this differential and $\partial_\theta \boldsymbol{\omega}'(\mathbf{p}, \mathbf{p}', \theta)$ is local and can be computed within a single loop iteration.

Overall, the reparameterization results in only a few additional lines compared to the original pseudocode (Listing 3 and Listing 4). Note that a single point $\mathbf{x}$ influences two different evaluations of τ , one where it is the previous point and one where it is the current point. This means that, similar to the code presented in Chapter 5, the contribution of a single point to the time-dependent derivatives is accumulated over two loop iterations.

C.2 Acoustic BSDF

In acoustics, the Bidirectional Scattering Distribution Function (BSDF) depends on the outgoing and incident direction $\boldsymbol{\omega}_o$ and $\boldsymbol{\omega}_i$ as well as point of reflection $\mathbf{x}$. It is commonly modeled as a combination of an ideal specular part and an ideal diffuse Lambertian part:

$$f(\mathbf{x}, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i) = \underbrace{(1-s)(1-\alpha) \frac{\delta(\boldsymbol{\omega}_r, \boldsymbol{\omega}_i)}{|\boldsymbol{\omega}_r \cdot \mathbf{n}|}}_{\text{specular}} + \underbrace{s(1-\alpha) \frac{1}{\pi}}_{\text{diffuse}}. \quad (\text{C.7})$$

Listing 3 Pseudocode for the gradient primal pass of temporal PRB using the three-point form. The function `intersect` returns the intersection point of a ray with the scene as a point in the surface parametrization $\mathcal{M}$, `to_ambient` transforms a point from the parametrization to ambient space.

```
def sample_path(pcam,  $\omega$ ,  $\delta H$ ):
     $\delta HL = 0$ ,  $\beta = 1$ ,  $T = 0$ ,  $\delta H \partial_T L = 0$ 
    pp = pcam
    pc = intersect(to_ambient(pcam),  $\omega$ )
    for i in range(1, N+1):
        xp = to_ambient(pp)
        xc = to_ambient(pc)
        T += time_on_segment(xp, x)

        # Measured energy
         $\delta HL += \beta * \langle \delta H, L_e(x_c, x_c \rightarrow x_p, T) \rangle$ 
         $\omega'$ , bsdf_value, bsdf_pdf = sample_bsdf(...)

        # Measured differential delay energy
         $\delta H \partial_T L += \beta * \text{forward\_grad}(T, \langle \delta H, L_e(x_c, x_c \rightarrow x_p, T) \rangle)$ 

         $\beta *= \text{bsdf\_value} / \text{bsdf\_pdf}$ 
        pp = pc
        pc = intersect(xc,  $\omega'$ )
    return  $\delta HL$ ,  $\delta H \partial_T L$ 
```

Listing 4 Pseudocode for the adjoint pass of temporal PRB using the three-point form. The function `intersect` returns the intersection point of a ray with the scene as a point in the surface parametrization $\mathcal{M}$, `to_ambient` transforms a point from the parametrization to ambient space, considering the scene parameters and `reparam_det` returns the value of the reparameterization determinant D .

```
def sample_path_adjoint(p_cam,  $\omega$ ,  $\delta H L$ ,  $\delta H \partial_T L$ ,  $\delta H$ ):
     $\beta = 1$ ,  $T = 0$ 
    p_p = p_cam
    p_c = detach(intersect(to_ambient(p_cam),  $\omega$ ))
    for i in range(1, N+1):
        x_p = to_ambient(p_p)
        x_c = to_ambient(p_c)
         $\tau += \text{time\_on\_segment}(x_p, x)$ 
         $T += \text{detach}(\tau)$ 

        # Derivative of emission term
         $\delta H E = \beta * \langle \delta H, L_e(x_c, x_c \rightarrow x_p, T) \rangle$ 
         $\delta_\theta += \text{backward\_grad}(\delta H E, 1)$ 

        # Derivative of Jacobian determinant
         $D = \text{reparam\_det}(x_p, x_c)$ 
         $\delta_\theta += \text{backward\_grad}(D, \delta L * L / D)$ 

         $\delta H L -= \delta H E$ 

        # Derivative of the BSDF
         $\omega'$ , bsdf_value, bsdf_pdf = sample_bsdf(...)
        p_n = detach(intersect(x_c,  $\omega'$ ))
        x_n = to_ambient(p_n)
        bsdf_value = eval_bsdf(x_c, x_c  $\rightarrow$  x_p, x_c  $\rightarrow$  x_n)
         $\delta_\theta += \text{backward\_grad}(\text{bsdf\_value}, \delta H L / \text{bsdf\_value})$ 

        # Backpropagate from time-dependent term
         $\delta_\theta += \text{backward\_grad}(\tau, \delta H \partial_T L)$ 
         $\delta H \partial_T L -= \beta * \text{forward\_grad}(T, \langle \delta H, L_e(\dots) \rangle)$ 

         $\beta *= \text{bsdf\_value} / \text{bsdf\_pdf}$ 
        p_p = p_c
        p_c = p_n
    return  $\delta_\theta$ 
```

The specular energy is reduced according to the scattering coefficient $0 \leq s \leq 1$, denoting the amount of scattered energy, and the absorption coefficient $0 \leq \alpha \leq 1$, denoting the amount of absorbed energy. It is reflected in direction of ω_r , which represents the reflection vector for ω_o reflected about the surface normal $\mathbf{n}$. The diffuse energy is weighted by the scattering coefficient and equally distributed across outgoing directions.

However, actual reflection patterns of finite surfaces always show a lobe around the specular direction [Embrevchts et al., 2001]. To account for this, Phong BSDFs have been utilized in acoustic rendering [Mückl and Dachsbacher, 2020; Binek et al., 2019, 2023]. Binek et al. [2019] report slightly improved simulation accuracy compared to the classical Specular+Lambert BSDF for Phong exponents ranging from 0–1000. Additionally, using a Phong BSDF can significantly reduce variance across simulations [Binek et al., 2023]. This is likely due to the non-zero contributions of the specular component at non-specular angles, which enable the sampling of the specular component through emitter sampling / diffuse rain. Notably, this also enables the *differentiation* of the specular component for detached sampling strategies. When modeling ideal specular reflections, only the diffuse component can be differentiated. It is therefore crucial for our method to use a continuous BSDF.

In our validation of the forward simulation we use the classical model (Eq. (C.7)) to ensure comparability to other acoustic simulation methods. In the optimization tasks we model the specular part with phong-style BSDFs using the beckmann distribution proposed by Walter et al. [2007]. Appropriate values for the modeled roughness are not systematically investigated in this work, but are not a limitation of our method. The diffuse part and the weighting between specularly and diffusely reflected energy is modeled according to Eq. (C.7).

C.3 Room Acoustics Metrics

In room acoustic design and architectural acoustics, sound propagation can assumed to be linear and time-invariant (LTI). Rooms are therefore often described by their Room Impulse Response (RIR) $h(t)$, which is the response of the room to an acoustic impulse, such as an idealized popping balloon. The RIR contains information about the energy and arrival time of direct sound and reflections, and is the central basis for deriving room acoustic metrics.

Most common room acoustic metrics do not require the phase information of the impulse response, because they quantify the temporal energy distribution in some way, so they can be derived from the energy time curve, which is the squared impulse response $H(t) = h^2(t)$. In our method, we simulate the ETC directly (see Eq. (6.5)).

The reverberation time T_{60} is arguably the most important room acoustic parameter and describes the time after which the energy has decayed to a millionth of the incident energy, corresponding to a 60 dB decrease in the sound pressure level [ISO 3382-1, 2009]:

$$T_{60} = \frac{-60 \text{ dB}}{\frac{\partial}{\partial t} \left(10 \log_{10} \left(\frac{\text{EDC}(t)}{\text{EDC}(t=0)} \right) \right)}. \quad (\text{C.8})$$

It is computed from the slope of a linear regression fitted to the normalized and loga-

rhythmized Energy Decay Curve (EDC):

$$\text{EDC}(t) = \int_t^{\infty} H(\tau) d\tau. \quad (\text{C.9})$$

The clarity is a widely accepted indicator for the transparency of musical sounds in concert halls that exploits the observation that reverberant energy arriving within the first 80 ms after the direct sound increases the intelligibility [Kuttruff, 2000]:

$$C_{80} = 10 \log_{10} \frac{\int_0^{80 \text{ ms}} H(t) dt}{\int_{80 \text{ ms}}^{\infty} H(t) dt}. \quad (\text{C.10})$$

Bibliography

- Tomas Akenine-Möller, Eric Haines, Naty Hoffman, Angelo Pesce, Michał Iwanicki, and Sébastien Hillaire. 2018. *Real-Time Rendering 4th Edition*. A K Peters/CRC Press, Boca Raton, FL, USA. 1200 pages.
- N. I. Akhiezer. 1965. *The Classical Moment Problem and Some Related Questions in Analysis* (1 ed.). Society for Industrial and Applied Mathematics, Philadelphia, PA. arXiv:<https://epubs.siam.org/doi/pdf/10.1137/1.9781611976397> <https://epubs.siam.org/doi/abs/10.1137/1.9781611976397>
- N. I. Akhiezer and M. G. Kreĭn. 1962. *Some questions in the theory of moments*. Vol. 2. American Mathematical Soc., Providence, RI.
- Marc Alexa. 2003. Differential coordinates for local mesh morphing and deformation. *The Visual Computer* 19, 2 (2003), 105–114.
- Furkan Mert Algan, Umut Yazgan, Driton Salihu, Cem Eteke, and Eckehard Steinbach. 2025. LEMON: Localized Editing With Mesh Optimization and Neural Shaders. *IEEE Open Journal of Signal Processing* 6 (2025), 1161–1168. <https://doi.org/10.1109/OJSP.2025.3627123>
- Alive Network. 2016. Charles Burns Silhouette Artist. <https://www.youtube.com/watch?v=sZEZs3yVkpQ>.
- Thomas Annen, Tom Mertens, Philippe Bekaert, Hans-Peter Seidel, and Jan Kautz. 2007. Convolution Shadow Maps. In *Proceedings of the 18th Eurographics Conference on Rendering Techniques* (Grenoble, France) (EGSR’07). Eurographics Association, Goslar, DEU, 51–60.
- Thomas Annen, Tom Mertens, Hans-Peter Seidel, Eddy Flerackers, and Jan Kautz. 2008. Exponential Shadow Maps. In *Proceedings of Graphics Interface 2008* (Windsor, Ontario, Canada) (GI ’08). Canadian Information Processing Society, CAN, 155–161.
- Lukas Aspöck, Fabian Brinkmann, David Ackerman, Stefan Weinzierl, and Michael Vorländer. 2020. BRAS - Benchmark for Room Acoustical Simulation. <https://doi.org/10.14279/depositonce-6726.3>
- Dejan Azinovic, Tzu-Mao Li, Anton Kaplanyan, and Matthias Niessner. 2019. Inverse Path Tracing for Joint Material and Lighting Estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Los Alamitos, CA, USA, 2447–2456.
- Sai Bangaru, Michael Gharbi, Tzu-Mao Li, Fujun Luan, Kalyan Sunkavalli, Milos Hasan, Sai Bi, Zexiang Xu, Gilbert Bernstein, and Fredo Durand. 2022. Differentiable Rendering of Neural SDFs through Reparameterization. In *ACM SIGGRAPH Asia 2022 Conference Proceedings* (Daegu, Republic of Korea) (SIGGRAPH Asia ’22). Association for Computing Machinery, New York, NY, USA, Article 22, 9 pages. <https://doi.org/10.1145/3550469.3555397>

- Sai Praveen Bangaru, Tzu-Mao Li, and Frédo Durand. 2020. Unbiased Warped-Area Sampling for Differentiable Rendering. *ACM Trans. Graph.* 39, 6, Article 245 (nov 2020), 18 pages. <https://doi.org/10.1145/3414685.3417833>
- Sai Praveen Bangaru, Jesse Michel, Kevin Mu, Gilbert Bernstein, Tzu-Mao Li, and Jonathan Ragan-Kelley. 2021. Systematically differentiating parametric discontinuities. *ACM Trans. Graph.* 40, 4, Article 107 (July 2021), 18 pages. <https://doi.org/10.1145/3450626.3459775>
- Sai Praveen Bangaru, Lifan Wu, Tzu-Mao Li, Jacob Munkberg, Gilbert Bernstein, Jonathan Ragan-Kelley, Frédo Durand, Aaron Lefohn, and Yong He. 2023. SLANG.D: Fast, Modular and Differentiable Shader Programming. *ACM Trans. Graph.* 42, 6, Article 264 (dec 2023), 28 pages.
- Yash Belhe, Bing Xu, Sai Praveen Bangaru, Ravi Ramamoorthi, and Tzu-Mao Li. 2024. Importance Sampling BRDF Derivatives. *ACM Trans. Graph.* 43, 3, Article 25 (April 2024), 21 pages. <https://doi.org/10.1145/3648611>
- Shrisha Bharadwaj, Yufeng Zheng, Otmar Hilliges, Michael J. Black, and Victoria Fernandez Abrevaya. 2023. FLARE: Fast Learning of Animatable and Relightable Mesh Avatars. *ACM Trans. Graph.* 42, 6, Article 204 (Dec. 2023), 15 pages. <https://doi.org/10.1145/3618401>
- Wojciech Binek, Aleksandra Chojak, and Tadeusz Kamisiński. 2023. The Impact of the Sound Reflection Model on the Ray Tracing Simulation Variability. *Vibrations in Physical Systems* 32, 2, Article 2021203 (2023), 8 pages. <https://doi.org/10.21008/J.0860-6897.2021.2.03>
- Wojciech Binek, Adam Pilch, and Tadeusz Kamisiński. 2019. Application of Fitted to Measurements Sound Reflecion Model in Geometrical Acoustics Simulations. In *INTER-NOISE and NOISE-CON Congress and Conference Proceedings*. Institute of Noise Control Engineering, Madrid, Spain, 1189–1194.
- Benedikt Bitterli, Chris Wyman, Matt Pharr, Peter Shirley, Aaron Lefohn, and Wojciech Jarosz. 2020. Spatiotemporal reservoir resampling for real-time ray tracing with dynamic direct lighting. *ACM Trans. Graph.* 39, 4, Article 148 (Aug. 2020), 17 pages. <https://doi.org/10.1145/3386569.3392481>
- Åke Björck and Victor Pereyra. 1970. Solution of Vandermonde systems of equations. *Mathematics of computation* 24, 112 (1970), 893–903.
- Matthew G Blevins, Adam T Buck, Zhao Peng, and Lily M Wang. 2013. Quantifying the Just Noticeable Difference of Reverberation Time with Band-Limited Noise Centered around 1000 Hz Using a Transformed up-down Adaptive Method. In *International Symposium on Room Acoustics*. CAA, Toronto, Canada, 1–6.
- Mark Boss, Varun Jampani, Raphael Braun, Ce Liu, Jonathan Barron, and Hendrik PA Lensch. 2021. Neural-PIL: Neural Pre-Integrated Lighting for Reflectance Decomposition. In *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman

- Vaughan (Eds.), Vol. 34. Curran Associates, Inc., Red Hook, NY, USA, 10691–10704. https://proceedings.neurips.cc/paper_files/paper/2021/file/58ae749f25eded36f486bc85feb3f0ab-Paper.pdf
- Mario Botsch and Leif Kobbelt. 2004. A Remeshing Approach to Multiresolution Modeling. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing* (Nice, France) (SGP '04). Association for Computing Machinery, New York, NY, USA, 185–192. <https://doi.org/10.1145/1057432.1057457>
- James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. 2018. JAX: composable transformations of Python+NumPy programs. <http://github.com/jax-ml/jax>
- Fabian Brinkmann, Lukas Aspöck, David Ackermann, Rob Opdam, Michael Vorländer, and Stefan Weinzierl. 2021. A benchmark for room acoustical simulation. Concept and database. *Applied Acoustics* 176 (2021), 107867. <https://doi.org/10.1016/j.apacoust.2020.107867>
- Tyson Brochu and Robert Bridson. 2009. Robust topological operations for dynamic explicit surfaces. *SIAM Journal on Scientific Computing* 31, 4 (2009), 2472–2493.
- Charles Burns. 2011. *Mastering Silhouettes: Expert Instruction in the Art of Silhouette Portraiture*. Stackpole Books, Mechanicsburg, PA, USA.
- Chunxiao Cao, Zhong Ren, Carl Schissler, Dinesh Manocha, and Kun Zhou. 2016. Interactive Sound Propagation with Bidirectional Path Tracing. *ACM Transactions on Graphics* 35, 6 (Nov. 2016), 1–11. <https://doi.org/10.1145/2980179.2982431>
- A. Celarek, G. Kopanas, G. Drettakis, M. Wimmer, and B. Kerbl. 2025. Does 3D Gaussian Splatting Need Accurate Volumetric Rendering? *Computer Graphics Forum* 44, 2 (2025), e70032. <https://doi.org/10.1111/cgf.70032> arXiv:<https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.70032>
- Wesley Chang, Venkataram Sivaram, Derek Nowrouzezahrai, Toshiya Hachisuka, Ravi Ramamoorthi, and Tzu-Mao Li. 2023. Parameter-space ReSTIR for Differentiable and Inverse Rendering. In *ACM SIGGRAPH 2023 Conference Proceedings* (Los Angeles, CA, USA) (SIGGRAPH '23). Association for Computing Machinery, New York, NY, USA, Article 18, 10 pages. <https://doi.org/10.1145/3588432.3591512>
- Wenzheng Chen, Jun Gao, Huan Ling, Edward J. Smith, Jaakko Lehtinen, Alec Jacobson, and Sanja Fidler. 2019. Learning to predict 3D objects with an interpolation-based differentiable renderer. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Curran Associates Inc., Red Hook, NY, USA, Article 862, 11 pages.
- Wenzheng Chen, Joey Litalien, Jun Gao, Zian Wang, Clement Fuji Tsang, Sameh Khamis, Or Litany, and Sanja Fidler. 2021. DIB-R++: learning to predict lighting and material

- with a hybrid differentiable renderer. In *Proceedings of the 35th International Conference on Neural Information Processing Systems (NIPS '21)*. Curran Associates Inc., Red Hook, NY, USA, Article 1749, 15 pages.
- Kiseok Choi, Inchul Kim, Dongyoung Choi, Julio Marco, Diego Gutierrez, and Min H. Kim. 2023. Self-Calibrating, Fully Differentiable NLOS Inverse Rendering. In *SIGGRAPH Asia 2023 Conference Papers* (<conf-loc>, <city>Sydney</city>, <state>NSW</state>, <country>Australia</country>, </conf-loc>) (SA '23). Association for Computing Machinery, New York, NY, USA, Article 108, 11 pages. <https://doi.org/10.1145/3610548.3618140>
- Claus L Christensen, George Koutsouris, and Jens H Rindel. 2014. Estimating Absorption of Materials to Match Room Model against Existing Room Using a Genetic Algorithm. <https://doi.org/10.13140/2.1.1588.8647>
- Forrester Cole, Kyle Genova, Avneesh Sud, Daniel Vlasic, and Zhoutong Zhang. 2021. Differentiable Surface Rendering via Non-Differentiable Sampling. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE Computer Society, Los Alamitos, CA, USA, 6068–6077. <https://doi.org/10.1109/ICCV48922.2021.00603>
- Alvaro Collet, Ming Chuang, Pat Sweeney, Don Gillett, Dennis Evseev, David Calabrese, Hugues Hoppe, Adam Kirk, and Steve Sullivan. 2015. High-Quality Streamable Free-Viewpoint Video. *ACM Trans. Graph.* 34, 4, Article 69 (July 2015), 13 pages. <https://doi.org/10.1145/2766945>
- Sayantana Datta, Derek Nowrouzezahrai, Christoph Schied, and Zhao Dong. 2022. Neural Shadow Mapping. In *ACM SIGGRAPH 2022 Conference Proceedings* (Vancouver, BC, Canada) (SIGGRAPH '22). Association for Computing Machinery, New York, NY, USA, Article 8, 9 pages. <https://doi.org/10.1145/3528233.3530700>
- Martin de La Gorce, Nikos Paragios, and David J. Fleet. 2008. Model-based hand tracking with texture, shading and self-occlusions. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society, Los Alamitos, CA, USA, 1–8. <https://doi.org/10.1109/CVPR.2008.4587752>
- Michael Deering, Stephanie Winner, Bic Schediwy, Chris Duffy, and Neil Hunt. 1988. The Triangle Processor and Normal Vector Shader: A VLSI System for High Performance Graphics. In *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '88)*. Association for Computing Machinery, New York, NY, USA, 21–30. <https://doi.org/10.1145/54852.378468>
- Cyril Delalandre, Pascal Gautron, Jean-Eudes Marvie, and Guillaume François. 2011. Transmittance function mapping. In *Symposium on Interactive 3D Graphics and Games* (San Francisco, California) (I3D '11). Association for Computing Machinery, New York, NY, USA, 31–38. <https://doi.org/10.1145/1944745.1944751>

- Thomas Deliot, Eric Heitz, and Laurent Belcour. 2024. Transforming a Non-Differentiable Rasterizer into a Differentiable One with Stochastic Gradient Estimation. *Proc. ACM Comput. Graph. Interact. Tech.* 7, 1, Article 3 (may 2024), 16 pages.
- Miles Detrixhe, Frédéric Gibou, and Chohong Min. 2013. A parallel fast sweeping method for the Eikonal equation. *J. Comput. Phys.* 237 (2013), 46–55.
- Tobias Djuren, Ugo Finnendahl, Maximilian Kohlbrenner, Markus Worchel, and Marc Alexa. 2025. Interpolating splines over triangulated surfaces by blending vertex-centric local geometries. *Computers & Graphics* 131 (2025), 104316. <https://doi.org/10.1016/j.cag.2025.104316>
- William Donnelly and Andrew Lauritzen. 2006. Variance Shadow Maps. In *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games* (Redwood City, California) (*I3D '06*). Association for Computing Machinery, New York, NY, USA, 161–165. <https://doi.org/10.1145/1111411.1111440>
- Hang Dou, Yajie Yan, Ethan Kerzner, Zeng Dai, and Chris Wyman. 2014. Adaptive Depth Bias for Shadow Maps. In *Proceedings of the 18th Meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (San Francisco, California) (*I3D '14*). Association for Computing Machinery, New York, NY, USA, 97–102. <https://doi.org/10.1145/2556700.2556706>
- Bernhard Egger, William A. P. Smith, Ayush Tewari, Stefanie Wuhrer, Michael Zollhoefer, Thabo Beeler, Florian Bernard, Timo Bolkart, Adam Kortylewski, Sami Romdhani, Christian Theobalt, Volker Blanz, and Thomas Vetter. 2020. 3D Morphable Face Models—Past, Present, and Future. *ACM Trans. Graph.* 39, 5, Article 157 (jun 2020), 38 pages. <https://doi.org/10.1145/3395208>
- Jean Jacques Embrechts, Dominique Archambeau, and Guy-Bart Stan. 2001. Determination of the Scattering Coefficient of Random Rough Diffusing Surfaces for Room Acoustics Applications. *Acta Acust. united Ac.* 87 (July 2001), 482–494.
- Randima Fernando, Sebastian Fernandez, Kavita Bala, and Donald P. Greenberg. 2001. Adaptive Shadow Maps. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '01)*. Association for Computing Machinery, New York, NY, USA, 387–390.
- Ugo Finnendahl, Markus Worchel, Tobias Jüterbock, Daniel Wujecki, Fabian Brinkmann, Stefan Weinzierl, and Marc Alexa. 2025. Differentiable Geometric Acoustic Path Tracing using Time-Resolved Path Replay Backpropagation. *ACM Trans. Graph.* 44, 4 (aug 2025). <https://doi.org/10.1145/3730900>
- Géza Freud. 1969. *Orthogonale Polynome*. Birkhäuser Basel, Basel.
- Pascal Fua and Yvan G Leclerc. 1995. Object-centered surface reconstruction: Combining multi-image stereo and shading. *International Journal of Computer Vision* 16, 1 (1995), 35–56.

- Clement Fuji Tsang, Maria Shugrina, Jean Francois Lafleche, Towaki Takikawa, Jiehan Wang, Charles Loop, Wenzheng Chen, Krishna Murthy Jatavallabhula, Edward Smith, Artem Rozantsev, Or Perel, Tianchang Shen, Jun Gao, Sanja Fidler, Gavriel State, Jason Gorski, Tommy Xiang, Jianing Li, Michael Li, and Rev Lebedean. 2022. Kaolin: A Pytorch Library for Accelerating 3D Deep Learning Research. <https://github.com/NVIDIAGameWorks/kaolin>.
- Yasutaka Furukawa and Jean Ponce. 2010. Accurate, Dense, and Robust Multiview Stereopsis. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32, 8 (2010), 1362–1376. <https://doi.org/10.1109/TPAMI.2009.161>
- Jun Gao, Tianchang Shen, Zian Wang, Wenzheng Chen, Kangxue Yin, Daiqing Li, Or Litany, Zan Gojcic, and Sanja Fidler. 2022. GET3D: a generative model of high quality 3D textured shapes learned from images. In *Proceedings of the 36th International Conference on Neural Information Processing Systems (New Orleans, LA, USA) (NIPS '22)*. Curran Associates Inc., Red Hook, NY, USA, Article 2308, 14 pages.
- Yaoan Gao, Jiamin Xu, James Tompkin, Qi Wang, Zheng Dong, Hujun Bao, Yujun Shen, Huamin Wang, Changqing Zou, and Weiwei Xu. 2025. Efficient Object Reconstruction with Differentiable Area Light Shading. In *Proceedings of the SIGGRAPH Asia 2025 Conference Papers (SA Conference Papers '25)*. Association for Computing Machinery, New York, NY, USA, Article 65, 12 pages. <https://doi.org/10.1145/3757377.3763865>
- Andreas Griewank and Andrea Walther. 2008. *Evaluating Derivatives* (second ed.). Society for Industrial and Applied Mathematics, Philadelphia, PA. <https://doi.org/10.1137/1.9780898717761> arXiv:<https://pubs.siam.org/doi/pdf/10.1137/1.9780898717761>
- Kaiwen Guo, Peter Lincoln, Philip Davidson, Jay Busch, Xueming Yu, Matt Whalen, Geoff Harvey, Sergio Orts-Escolano, Rohit Pandey, Jason Dourgarian, Danhang Tang, Anastasia Tkach, Adarsh Kowdle, Emily Cooper, Mingsong Dou, Sean Fanello, Graham Fyffe, Christoph Rhemann, Jonathan Taylor, Paul Debevec, and Shahram Izadi. 2019. The Relightables: Volumetric Performance Capture of Humans with Realistic Relighting. *ACM Trans. Graph.* 38, 6, Article 217 (Nov. 2019), 19 pages. <https://doi.org/10.1145/3355089.3356571>
- Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. 2020. Array programming with NumPy. *Nature* 585, 7825 (Sept. 2020), 357–362. <https://doi.org/10.1038/s41586-020-2649-2>

- Jean-Marc Hasenfratz, Marc Lapierre, Nicolas Holzschuch, and François X. Sillion. 2003. A survey of Real-Time Soft Shadows Algorithms. *Computer Graphics Forum* 22, 4 (Dec. 2003), 753–774. <https://doi.org/10.1111/j.1467-8659.2003.00722.x>
- Jon Hasselgren, Nikolai Hofmann, and Jacob Munkberg. 2022. Shape, light, and material decomposition from images using Monte Carlo rendering and denoising. In *Proceedings of the 36th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) (NIPS '22). Curran Associates Inc., Red Hook, NY, USA, Article 1661, 14 pages.
- Wolfgang Heidrich and Hans-Peter Seidel. 1999. Realistic, Hardware-Accelerated Shading and Lighting. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '99)*. ACM Press/Addison-Wesley Publishing Co., USA, 171–178. <https://doi.org/10.1145/311535.311554>
- Nicholas J Higham. 1987. Error analysis of the Björck-Pereyra algorithms for solving Vandermonde systems. *Numer. Math.* 50 (1987), 613–632.
- Kai-Wen Hsiao, Jia-Bin Huang, and Hung-Kuo Chu. 2018. Multi-View Wire Art. *ACM Trans. Graph.* 37, 6, Article 242 (dec 2018), 11 pages. <https://doi.org/10.1145/3272127.3275070>
- Xuejun Hu, Jinfan Lu, and Kun Xu. 2025. Kernel Predicting Neural Shadow Maps. In *Proceedings of the Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers (SIGGRAPH Conference Papers '25)*. Association for Computing Machinery, New York, NY, USA, Article 36, 10 pages. <https://doi.org/10.1145/3721238.3730645>
- IEC 60268-16. 2020. *Sound System Equipment - Part 16: Objective Rating of Speech Intelligibility by Speech Transmission Index*. International Electrotechnical Commission, Geneva, Switzerland.
- ISO 3382-1. 2009. *Measurement of Room Acoustic Parameters – Part 1: Performance Spaces*. International Organization for Standards, Geneva, Switzerland.
- Wenzel Jakob. 2022. nanobind: tiny and efficient C++/Python bindings. <https://github.com/wjakob/nanobind>.
- Wenzel Jakob, Sébastien Speierer, Nicolas Roussel, Merlin Nimier-David, Delio Vicini, Tizian Zeltner, Baptiste Nicolet, Miguel Crespo, Vincent Leroy, and Ziyi Zhang. 2022b. Mitsuba 3 renderer. <https://mitsuba-renderer.org>.
- Wenzel Jakob, Sébastien Speierer, Nicolas Roussel, and Delio Vicini. 2022a. DR.JIT: A Just-in-Time Compiler for Differentiable Rendering. *ACM Trans. Graph.* 41, 4, Article 124 (jul 2022), 19 pages. <https://doi.org/10.1145/3528223.3530099>
- Jon Jansen and Louis Bavoil. 2010. Fourier opacity mapping. In *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (Washington, D.C.) (I3D '10). Association for Computing Machinery, New York, NY, USA, 165–172. <https://doi.org/10.1145/1730804.1730831>

- Rasmus Jensen, Anders Dahl, George Vogiatzis, Engil Tola, and Henrik Aanæs. 2014. Large Scale Multi-view Stereopsis Evaluation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society, Los Alamitos, CA, USA, 406–413. <https://doi.org/10.1109/CVPR.2014.59>
- Yue Jiang, Dantong Ji, Zhizhong Han, and Matthias Zwicker. 2020. SDFDiff: Differentiable Rendering of Signed Distance Fields for 3D Shape Optimization. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Los Alamitos, CA, USA, 1248–1258. <https://doi.org/10.1109/CVPR42600.2020.00133>
- Justin Johnson, Alexandre Alahi, and Li Fei-Fei. 2016. Perceptual Losses for Real-Time Style Transfer and Super-Resolution. In *Computer Vision – ECCV 2016*, Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling (Eds.). Springer International Publishing, Cham, 694–711.
- Tobias Jüterbock, Daniel Wujecki, and Stefan Weinzierl. 2024. Differentiable Acoustic Path Tracing: Full Spectral Rendering. In *Fortschritte Der Akustik - DAGA 2024*. Deutsche Gesellschaft für Akustik e.V. (DEGA), Hannover, 652–655.
- W. Kahan. 2004. On the cost of floating-point computation without extra-precise arithmetic.
- James T. Kajiya. 1986. The Rendering Equation. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '86)*. Association for Computing Machinery, New York, NY, USA, 143–150. <https://doi.org/10.1145/15922.15902>
- James T. Kajiya and Brian P Von Herzen. 1984. Ray Tracing Volume Densities. In *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '84)*. Association for Computing Machinery, New York, NY, USA, 165–174.
- Korrawe Karunratanakul, Sergey Prokudin, Otmar Hilliges, and Siyu Tang. 2023. HARP: Personalized Hand Reconstruction from a Monocular RGB Video. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Los Alamitos, CA, USA, 12802–12813. <https://doi.org/10.1109/CVPR52729.2023.01231>
- Hiroharu Kato, Deniz Beker, Mihai Morariu, Takahiro Ando, Toru Matsuoka, Wadim Kehl, and Adrien Gaidon. 2020. Differentiable Rendering: A Survey. <https://doi.org/10.48550/ARXIV.2006.12057>
- Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. 2018. Neural 3D Mesh Renderer. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society, Los Alamitos, CA, USA, 3907–3916. <https://doi.org/10.1109/CVPR.2018.00411>

- Bernhard Kerbl, Georgios Kopanas, Thomas Leimkuehler, and George Drettakis. 2023. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Trans. Graph.* 42, 4, Article 139 (July 2023), 14 pages. <https://doi.org/10.1145/3592433>
- Sami Kiminki. 2005. *Sound Propagation Theory for Linear Ray Acoustic Modelling*. Ph. D. Dissertation. Helsinki University of Technology.
- Diederik P. Kingma and Jimmy Ba. 2017. Adam: A Method for Stochastic Optimization. arXiv:1412.6980 [cs.LG] <https://arxiv.org/abs/1412.6980>
- Christoph Kirsch and Stephan D. Ewert. 2023. A Universal Filter Approximation of Edge Diffraction for Geometrical Acoustics. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 31 (2023), 1636–1651. <https://doi.org/10.1109/TASLP.2023.3264737>
- Sebastian Koch, Yurii Piadyk, Markus Worchel, Marc Alexa, Cláudio Silva, Denis Zorin, and Daniele Panozzo. 2021. Hardware Design and Accurate Simulation of Structured-Light Scanning for Benchmarking of 3D Reconstruction Algorithms. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*. Curran Associates Inc., Red Hook, NY, USA. <https://openreview.net/forum?id=bNL5V1Tfe3p>
- M. G. Kreĭn and A. A. Nudel'man. 1977. *The Markov Moment Problem and Extremal Problems*. Translations of Mathematical Monographs, Vol. 50. American Mathematical Society, Providence, RI.
- Heinrich Kuttruff. 2000. *Room Acoustics* (4th ed ed.). Spon Press, London, [England] ; New York, NY.
- Kiriakos N. Kutulakos and Steven M. Seitz. 2000. A Theory of Shape by Space Carving. *International Journal of Computer Vision* 38, 3 (2000), 199–218. <https://doi.org/10.1023/A:1008191222954>
- Alexandr Kuznetsov, Xuezheng Wang, Krishna Mullia, Fujun Luan, Zexiang Xu, Milos Hasan, and Ravi Ramamoorthi. 2022. Rendering Neural Materials on Curved Surfaces. In *ACM SIGGRAPH 2022 Conference Proceedings* (Vancouver, BC, Canada) (SIGGRAPH '22). Association for Computing Machinery, New York, NY, USA, Article 9, 9 pages. <https://doi.org/10.1145/3528233.3530721>
- Laura Labia, Louena Shtrepi, and Arianna Astolfi. 2020. Improved Room Acoustics Quality in Meeting Rooms: Investigation on the Optimal Configurations of Sound-Absorptive and Sound-Diffusive Panels. *Acoustics* 2, 3 (2020), 451–473. <https://doi.org/10.3390/acoustics2030025>
- Samuli Laine, Janne Hellsten, Tero Karras, Yeongho Seol, Jaakko Lehtinen, and Timo Aila. 2020. Modular Primitives for High-Performance Differentiable Rendering. *ACM Trans. Graph.* 39, 6, Article 194 (nov 2020), 14 pages. <https://doi.org/10.1145/3414685.3417861>

- Christoph Lassner and Michael Zollhöfer. 2021. Pulsar: Efficient Sphere-based Neural Rendering. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Los Alamitos, CA, USA, 1440–1449. <https://doi.org/10.1109/CVPR46437.2021.00149>
- Alexandros Lattas, Stylianos Moschoglou, Stylianos Ploumpis, Baris Gecer, Abhijeet Ghosh, and Stefanos Zafeiriou. 2022. AvatarMe++: Facial Shape and BRDF Inference With Photorealistic Rendering-Aware GANs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44, 12 (2022), 9269–9284. <https://doi.org/10.1109/TPAMI.2021.3125598>
- A. Laurentini. 1994. The visual hull concept for silhouette-based image understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 16, 2 (1994), 150–162. <https://doi.org/10.1109/34.273735>
- Aaron E. Lefohn, Shubhabrata Sengupta, and John D. Owens. 2007. Resolution-Matched Shadow Maps. *ACM Trans. Graph.* 26, 4 (oct 2007), 20–es.
- Vincent Lepetit and Pascal Fua. 2005. Monocular Model-Based 3D Tracking of Rigid Objects. *Found. Trends. Comput. Graph. Vis.* 1, 1 (jan 2005), 1–89. <https://doi.org/10.1561/06000000001>
- Ruilong Li, Karl Bladin, Yajie Zhao, Chinmay Chinara, Owen Ingraham, Pengda Xiang, Xinglei Ren, Pratusha Prasad, Bipin Kishore, Jun Xing, and Hao Li. 2020. Learning Formation of Physically-Based Face Attributes. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Los Alamitos, CA, USA, 3407–3416. <https://doi.org/10.1109/CVPR42600.2020.00347>
- Tzu-Mao Li et al. 2018b. Redner. <https://github.com/BachiLi/redner>.
- Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. 2018a. Differentiable Monte Carlo ray tracing through edge sampling. *ACM Trans. Graph.* 37, 6, Article 222 (Dec. 2018), 11 pages. <https://doi.org/10.1145/3272127.3275109>
- Zhaoshuo Li, Thomas Müller, Alex Evans, Russell H. Taylor, Mathias Unberath, Ming-Yu Liu, and Chen-Hsuan Lin. 2023. Neuralangelo: High-Fidelity Neural Surface Reconstruction. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Los Alamitos, CA, USA, 8456–8465. <https://doi.org/10.1109/CVPR52729.2023.00817>
- Ruofan Liang, Jiahao Zhang, Haoda Li, Chen Yang, Yushi Guan, and Nandita Vijaykumar. 2022. SPIDR: SDF-based Neural Point Fields for Illumination and Deformation.
- Selena Ling, Nicholas Sharp, and Alec Jacobson. 2022. VectorAdam for rotation equivariant geometry optimization. In *Proceedings of the 36th International Conference on Neural Information Processing Systems (New Orleans, LA, USA) (NIPS '22)*. Curran Associates Inc., Red Hook, NY, USA, Article 297, 12 pages.

- Hsueh-Ti Derek Liu, Michael Tao, and Alec Jacobson. 2018. Paparazzi: surface editing by way of multi-view image processing. *ACM Trans. Graph.* 37, 6, Article 221 (Dec. 2018), 11 pages. <https://doi.org/10.1145/3272127.3275047>
- Ruoshi Liu, Sachit Menon, Chengzhi Mao, Dennis Park, Simon Stent, and Carl Vondrick. 2023. What You Can Reconstruct from a Shadow. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Los Alamitos, CA, USA, 17059–17068. <https://doi.org/10.1109/CVPR52729.2023.01636>
- Shichen Liu, Weikai Chen, Tianye Li, and Hao Li. 2019a. Soft Rasterizer: A Differentiable Renderer for Image-Based 3D Reasoning. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE Computer Society, Los Alamitos, CA, USA, 7707–7716. <https://doi.org/10.1109/ICCV.2019.00780>
- Shichen Liu, Shunsuke Saito, Weikai Chen, and Hao Li. 2019b. Learning to Infer Implicit Surfaces without 3D Supervision. In *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc., Red Hook, NY, USA. https://proceedings.neurips.cc/paper_files/paper/2019/file/bdf3fd65c81469f9b74cedd497f2f9ce-Paper.pdf
- Tom Lokovic and Eric Veach. 2000. Deep Shadow Maps. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '00)*. ACM Press/Addison-Wesley Publishing Co., USA, 385–392.
- Matthew M. Loper and Michael J. Black. 2014. OpenDR: An Approximate Differentiable Renderer. In *Computer Vision – ECCV 2014*, David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars (Eds.). Springer International Publishing, Cham, 154–169.
- Guillaume Loubet, Nicolas Holzschuch, and Wenzel Jakob. 2019. Reparameterizing Discontinuous Integrands for Differentiable Rendering. *ACM Trans. Graph.* 38, 6, Article 228 (Nov. 2019), 14 pages. <https://doi.org/10.1145/3355089.3356510>
- Fujun Luan, Shuang Zhao, Kavita Bala, and Zhao Dong. 2021. Unified Shape and SVBRDF Recovery using Differentiable Monte Carlo Rendering. *Computer Graphics Forum* 40, 4 (2021), 101–113. <https://doi.org/10.1111/cgf.14344>
- Linjie Lyu, Marc Habermann, Lingjie Liu, Mallikarjun B R, Ayush Tewari, and Christian Theobalt. 2021. Efficient and Differentiable Shadow Computation for Inverse Problems. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE Computer Society, Los Alamitos, CA, USA, 13087–13096. <https://doi.org/10.1109/ICCV48922.2021.01286>
- Anagh Malik, Benjamin Attal, Andrew Xie, Matthew O'Toole, and David B Lindell. 2025. Neural inverse rendering from propagating light. In *Proceedings of the Computer Vision and Pattern Recognition Conference*. IEEE Computer Society, Los Alamitos, CA, USA, 10534–10544.

- Morgan McGuire, Mike Mara, Derek Nowrouzezahrai, and David Luebke. 2017. Real-time global illumination using precomputed light field probes. In *Proceedings of the 21st ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (San Francisco, California) (*I3D '17*). Association for Computing Machinery, New York, NY, USA, Article 2, 11 pages. <https://doi.org/10.1145/3023368.3023378>
- Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. 2020. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *Computer Vision – ECCV 2020*, Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm (Eds.). Springer International Publishing, Cham, 405–421.
- Gavin Miller. 1994. Efficient Algorithms for Local and Global Accessibility Shading. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '94)*. Association for Computing Machinery, New York, NY, USA, 319–326. <https://doi.org/10.1145/192161.192244>
- Niloy J. Mitra and Mark Pauly. 2009. Shadow Art. *ACM Trans. Graph.* 28, 5 (dec 2009), 1–7. <https://doi.org/10.1145/1618452.1618502>
- Gregor Mückl and Carsten Dachsbacher. 2020. Real-Time Room Reverb in Large Scenes Using Transport Path Precomputation. *Journal of Computer Graphics Techniques (JCGT)* 9, 1 (March 2020), 33–53. <http://jcgt.org/published/0009/01/03/>
- Thomas Müller, Fabrice Rousselle, Jan Novák, and Alexander Keller. 2021. Real-time neural radiance caching for path tracing. *ACM Trans. Graph.* 40, 4, Article 36 (July 2021), 16 pages. <https://doi.org/10.1145/3450626.3459812>
- Jacob Munkberg, Wenzheng Chen, Jon Hasselgren, Alex Evans, Tianchang Shen, Thomas Müller, Jun Gao, and Sanja Fidler. 2022. Extracting Triangular 3D Models, Materials, and Lighting From Images. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Los Alamitos, CA, USA, 8270–8280. <https://doi.org/10.1109/CVPR52688.2022.00810>
- Cedrick Münstermann, Stefan Krumpen, Reinhard Klein, and Christoph Peters. 2018. Moment-Based Order-Independent Transparency. *Proc. ACM Comput. Graph. Interact. Tech.* 1, 1, Article 7 (jul 2018), 20 pages.
- Jan U. Müller, Robin Tim Landsgesell, Leif Van Holland, Patrick Stotko, and Reinhard Klein. 2025. Moment-Based 3D Gaussian Splatting: Resolving Volumetric Occlusion with Order-Independent Transmittance. arXiv:2512.11800 [cs.CV] <https://arxiv.org/abs/2512.11800>
- O. Nalbach, E. Arabadzhiyska, D. Mehta, H.-P. Seidel, and T. Ritschel. 2017. Deep Shading: Convolutional Neural Networks for Screen Space Shading. *Computer Graphics Forum* 36, 4 (2017), 65–78. <https://doi.org/10.1111/cgf.13225> arXiv:<https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13225>

- Hubert Nguyen. 2007. *Gpu Gems 3* (first ed.). Addison-Wesley Professional, Boston, MA, USA.
- Baptiste Nicolet, Alec Jacobson, and Wenzel Jakob. 2021. Large steps in inverse rendering of geometry. *ACM Trans. Graph.* 40, 6, Article 248 (Dec. 2021), 13 pages. <https://doi.org/10.1145/3478513.3480501>
- Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. 2020. Differentiable Volumetric Rendering: Learning Implicit 3D Representations Without 3D Supervision. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Los Alamitos, CA, USA, 3501–3512. <https://doi.org/10.1109/CVPR42600.2020.00356>
- Merlin Nimier-David, Sébastien Speierer, Benoît Ruiz, and Wenzel Jakob. 2020. Radiative Backpropagation: An Adjoint Method for Lightning-Fast Differentiable Rendering. *ACM Trans. Graph.* 39, 4, Article 146 (aug 2020), 15 pages. <https://doi.org/10.1145/3386569.3392406>
- Merlin Nimier-David, Delio Vicini, Tizian Zeltner, and Wenzel Jakob. 2019. Mitsuba 2: a retargetable forward and inverse renderer. *ACM Trans. Graph.* 38, 6, Article 203 (Nov. 2019), 17 pages. <https://doi.org/10.1145/3355089.3356498>
- Michael Oechsle, Songyou Peng, and Andreas Geiger. 2021. UNISURF: Unifying Neural Implicit Surfaces and Radiance Fields for Multi-View Reconstruction. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE Computer Society, Los Alamitos, CA, USA, 5569–5579. <https://doi.org/10.1109/ICCV48922.2021.00554>
- Werner Palfinger. 2022. Continuous remeshing for inverse rendering. *Computer Animation and Virtual Worlds* 33, 5 (2022), e2101. <https://doi.org/10.1002/cav.2101> arXiv:<https://onlinelibrary.wiley.com/doi/pdf/10.1002/cav.2101>
- Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. 2019. DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Los Alamitos, CA, USA, 165–174. <https://doi.org/10.1109/CVPR.2019.00025>
- Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems* 32, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., Red Hook, NY, USA, 8024–8035. <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>

- Christoph Peters. 2017. Non-linearly quantized moment shadow maps. In *Proceedings of High Performance Graphics* (Los Angeles, California) (HPG '17). Association for Computing Machinery, New York, NY, USA, Article 15, 11 pages.
- Christoph Peters. 2023. Finding Real Polynomial Roots on GPUs. <https://momentsingraphics.de/GPUPolynomialRoots.html>.
- Christoph Peters and Reinhard Klein. 2015. Moment Shadow Mapping. In *Proceedings of the 19th Symposium on Interactive 3D Graphics and Games* (San Francisco, California) (i3D '15). Association for Computing Machinery, New York, NY, USA, 7–14. <https://doi.org/10.1145/2699276.2699277>
- Christoph Peters, Cedrick Munstermann, Nico Wetzstein, and Reinhard Klein. 2016. Beyond hard shadows: moment shadow maps for single scattering, soft shadows and translucent occluders. In *Proceedings of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (Redmond, Washington) (I3D '16). Association for Computing Machinery, New York, NY, USA, 159–170.
- Felix Petersen, Bastian Goldluecke, Christian Borgelt, and Oliver Deussen. 2022. GenDR: A Generalized Differentiable Renderer. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Los Alamitos, CA, USA, 3992–4001. <https://doi.org/10.1109/CVPR52688.2022.00397>
- Matt Pharr, Wenzel Jakob, and Greg Humphreys. 2016. *Physically Based Rendering: From Theory to Implementation* (3rd ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
- Stanislav Pidhorskyi, Tomas Simon, Gabriel Schwartz, He Wen, Yaser Sheikh, and Jason Saragih. 2025. Rasterized Edge Gradients: Handling Discontinuities Differentiably. In *Computer Vision – ECCV 2024*, Aleš Leonardis, Elisa Ricci, Stefan Roth, Olga Russakovsky, Torsten Sattler, and Gül Varol (Eds.). Springer Nature Switzerland, Cham, 335–352.
- Adam Pilch. 2020. Optimization-based method for the calibration of geometrical acoustic models. *Applied Acoustics* 170 (2020), 107495. <https://doi.org/10.1016/j.apacoust.2020.107495>
- Markus Plack, Clara Callenberg, Monika Schneider, and Matthias B. Hullin. 2023. Fast Differentiable Transient Rendering for Non-Line-of-Sight Reconstruction. In *2023 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. IEEE Computer Society, Los Alamitos, CA, USA, 3066–3075. <https://doi.org/10.1109/WACV56688.2023.00308>
- Benjamin Planche and Rajat Vikram Singh. 2021. Physics-based Differentiable Depth Sensor Simulation. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE Computer Society, Los Alamitos, CA, USA, 14367–14377. <https://doi.org/10.1109/ICCV48922.2021.01412>

- Alexander Pohl and Uwe M. Stephenson. 2014. Combining Higher Order Reflections with Diffractions without Explosion of Computation Time: The Sound Particle Radiosity Method. In *Proc. of the EAA Joint Symposium on Auralization and Ambisonics*. Universitätsverlag der TU Berlin, Berlin, Germany, 119–125. <https://doi.org/10.14279/depositonce-20>
- Wentian Qu, Zhaopeng Cui, Yinda Zhang, Chenyu Meng, Cuixia Ma, Xiaoming Deng, and Hongan Wang. 2023. Novel-view Synthesis and Pose Estimation for Hand-Object Interaction from Sparse Views. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE Computer Society, Los Alamitos, CA, USA, 15054–15065. <https://doi.org/10.1109/ICCV51070.2023.01386>
- Michal Radziszewski, Krzysztof Boryczko, and Witold Alda. 2009. An Improved Technique for Full Spectral Rendering. *Journal of WSCG* 17 (Jan. 2009), 9–16.
- Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. 2020. Accelerating 3D Deep Learning with PyTorch3D. arXiv:2007.08501 [cs.CV] <https://arxiv.org/abs/2007.08501>
- William T. Reeves, David H. Salesin, and Robert L. Cook. 1987. Rendering Antialiased Shadows with Depth Maps. *SIGGRAPH Comput. Graph.* 21, 4 (aug 1987), 283–291. <https://doi.org/10.1145/37402.37435>
- Osborne Reynolds, Arthur William Brightmore, and William Henry Moorby. 1903. *The sub-mechanics of the universe*. Vol. 3. University Press, Cambridge.
- Frédéric Riesz. 1911. Sur certains systèmes singuliers d'équations intégrales. *Annales scientifiques de l'École Normale Supérieure* 28 (1911), 33–62. <http://eudml.org/doc/81305>
- Kaustubh Sadekar, Ashish Tiwari, and Shanmuganathan Raman. 2022. Shadow Art Revisited: A Differentiable Rendering Based Approach. In *2022 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. IEEE Computer Society, Los Alamitos, CA, USA, 628–636. <https://doi.org/10.1109/WACV51458.2022.00070>
- M Salvi. 2008. Probabilistic approaches to shadow maps filtering, February.
- Nikolaos Sarafianos, Bogdan Boteanu, Bogdan Ionescu, and Ioannis A. Kakadiaris. 2016. 3D Human pose estimation: A review of the literature and analysis of covariates. *Computer Vision and Image Understanding* 152 (2016), 1–20. <https://doi.org/10.1016/j.cviu.2016.09.002>
- Johannes L. Schönberger, Enliang Zheng, Jan-Michael Frahm, and Marc Pollefeys. 2016. Pixelwise View Selection for Unstructured Multi-View Stereo. In *Computer Vision – ECCV 2016*, Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling (Eds.). Springer International Publishing, Cham, 501–518.
- Oliver Schreer, Ingo Feldmann, Sylvain Renault, Marcus Zepp, Markus Worchel, Peter Eisert, and Peter Kauff. 2019. Capture and 3D Video Processing of Volumetric Video.

- In *2019 IEEE International Conference on Image Processing (ICIP)*. IEEE Computer Society, Los Alamitos, CA, USA, 4310–4314. <https://doi.org/10.1109/ICIP.2019.8803576>
- Oliver Schreer, Ingo Feldmann, Sylvain Renault, Marcus Zepp, Markus Worchel, Peter Eisert, and Peter Kauff. 2019. Capture and 3D Video Processing of Volumetric Video. In *2019 IEEE International Conference on Image Processing (ICIP)*. IEEE Computer Society, Los Alamitos, CA, USA, 4310–4314. <https://doi.org/10.1109/ICIP.2019.8803576>
- Dirk Schröder. 2011. *Physically based real-time auralization of interactive virtual environments*. Ph. D. Dissertation. RWTH Aachen.
- Johannes L. Schönberger and Jan-Michael Frahm. 2016. Structure-from-Motion Revisited. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Los Alamitos, CA, USA, 4104–4113. <https://doi.org/10.1109/CVPR.2016.445>
- Silvia Sellán and Baptiste Nicolet. 2021. Libigl Botsch-Kobbelt Local Remesher. <https://github.com/sgsellan/botsch-kobbelt-remesher-libigl>.
- Steven A Shafer and Takeo Kanade. 1983. Using shadows in finding surface orientations. *Computer Vision, Graphics, and Image Processing* 22, 1 (1983), 145–176. [https://doi.org/10.1016/0734-189X\(83\)90099-3](https://doi.org/10.1016/0734-189X(83)90099-3)
- Brian Sharpe. 2018. Moment transparency. In *Proceedings of the Conference on High-Performance Graphics (Vancouver, British Columbia, Canada) (HPG '18)*. Association for Computing Machinery, New York, NY, USA, Article 8, 4 pages.
- Samuel Siltanen, Tapio Lokki, Sami Kiminki, and Lauri Savioja. 2007. The Room Acoustic Rendering Equation. *The Journal of the Acoustical Society of America* 122, 3 (Sept. 2007), 1624–1635. <https://doi.org/10.1121/1.2766781>
- Vincent Sitzmann, Julien N. P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. 2020. Implicit neural representations with periodic activation functions. In *Proceedings of the 34th International Conference on Neural Information Processing Systems (Vancouver, BC, Canada) (NIPS '20)*. Curran Associates Inc., Red Hook, NY, USA, Article 626, 12 pages.
- Adam Smith, James Skorupski, and James Davis. 2008. Transient Rendering.
- Mariia Soroka, Christoph Peters, and Steve Marschner. 2025. Quadric-Based Silhouette Sampling for Differentiable Rendering. *ACM Trans. Graph.* 44, 4, Article 81 (July 2025), 20 pages. <https://doi.org/10.1145/3731146>
- Jos Stam. 2020. Computing Light Transport Gradients using the Adjoint Method. arXiv:2006.15059 [cs.GR] <https://arxiv.org/abs/2006.15059>

- Antonio Stanziola, Simon R. Arridge, Ben T. Cox, and Bradley E. Treeby. 2023. j-Wave: An open-source differentiable wave simulator. *SoftwareX* 22 (2023), 101338. <https://doi.org/10.1016/j.softx.2023.101338>
- J. Starck and A. Hilton. 2007. Surface Capture for Performance-Based Animation. *IEEE Computer Graphics and Applications* 27, 3 (2007), 21–31. <https://doi.org/10.1109/MCG.2007.68>
- Shlomi Steinberg, Ravi Ramamoorthi, Benedikt Bitterli, Arshiya Mollazainali, Eugene D'Eon, and Matt Pharr. 2024. A Free-Space Diffraction BSDF. *ACM Transactions on Graphics* 43, 4 (July 2024), 1–15. <https://doi.org/10.1145/3658166>
- Gilbert Strang. 2007. *Computational Science and Engineering*. Wellesley-Cambridge Press, Philadelphia, PA. arXiv:<https://epubs.siam.org/doi/pdf/10.1137/1.9780961408817> <https://epubs.siam.org/doi/abs/10.1137/1.9780961408817>
- Tanli Su and Ioannis Gkioulekas. 2024. Path Sampling Methods for Differentiable Rendering. In *Eurographics Symposium on Rendering*, Eric Haines and Elena Garces (Eds.). The Eurographics Association, Saarbrücken, Germany. <https://doi.org/10.2312/sr.20241148>
- Qilin Sun, Congli Wang, Qiang Fu, Xiong Dun, and Wolfgang Heidrich. 2021. End-to-End Complex Lens Design with Differentiate Ray Tracing. *ACM Trans. Graph.* 40, 4, Article 71 (jul 2021), 13 pages. <https://doi.org/10.1145/3450626.3459674>
- U. Peter Svensson and Paul T. Calamia. 2006. Edge-Diffraction Impulse Responses near Specular-Zone and Shadow-Zone Boundaries. *Acta Acust. united Ac.* 92 (2006), 501–512.
- G. Szegő. 1975. *Orthogonal Polynomials*. American Mathematical Society, Providence, RI.
- Alejandro Sztrajman, Gilles Rainer, Tobias Ritschel, and Tim Weyrich. 2021. Neural BRDF Representation and Importance Sampling. *Computer Graphics Forum* 40, 6 (2021), 332–346. <https://doi.org/10.1111/cgf.14335> arXiv:<https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.14335>
- Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. 2020. Fourier features let networks learn high frequency functions in low dimensional domains. In *Proceedings of the 34th International Conference on Neural Information Processing Systems* (Vancouver, BC, Canada) (*NIPS '20*). Curran Associates Inc., Red Hook, NY, USA, Article 632, 11 pages.
- Jiaxiang Tang, Hang Zhou, Xiaokang Chen, Tianshu Hu, Errui Ding, Jingdong Wang, and Gang Zeng. 2023. Delicate Textured Mesh Recovery from NeRF via Adaptive Surface Refinement. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE Computer Society, Los Alamitos, CA, USA, 17693–17703. <https://doi.org/10.1109/ICCV51070.2023.01626>

- Árpád Tari. 2005. *Moments based bounds in stochastic models*. Ph.D. dissertation. Budapest University of Technology and Economics.
- Árpád Tari, Miklós Telek, and Peter Buchholz. 2005. A Unified Approach to the Moments Based Distribution Estimation – Unbounded Support. In *Formal Techniques for Computer Systems and Business Processes*, Mario Bravetti, Leïla Kloul, and Gianluigi Zavattaro (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 79–93.
- Arjun Teh, Delio Vicini, Bernd Bickel, Ioannis Gkioulekas, and Matthew O’Toole. 2025. Automated design of compound lenses with discrete-continuous optimization. In *Proceedings of the SIGGRAPH Asia 2025 Conference Papers (SA Conference Papers ’25)*. Association for Computing Machinery, New York, NY, USA, Article 47, 11 pages. <https://doi.org/10.1145/3757377.3763850>
- Justus Thies, Michael Zollhöfer, and Matthias Nießner. 2019. Deferred Neural Rendering: Image Synthesis Using Neural Textures. *ACM Trans. Graph.* 38, 4, Article 66 (July 2019), 12 pages. <https://doi.org/10.1145/3306346.3323035>
- Kushagra Tiwary, Tzofi Klinghoffer, and Ramesh Raskar. 2022. Towards Learning Neural Representations from Shadows. In *Computer Vision – ECCV 2022*, Shai Avidan, Gabriel Brostow, Moustapha Cissé, Giovanni Maria Farinella, and Tal Hassner (Eds.). Springer Nature Switzerland, Cham, 300–316.
- Heang K. Tuy and Lee Tan Tuy. 1984. Direct 2-D display of 3-D objects. *IEEE Computer Graphics and Applications* 4, 10 (1984), 29–34. <https://doi.org/10.1109/MCG.1984.6429333>
- Eric Veach. 1998. *Robust Monte Carlo Methods for Light Transport Simulation*. Ph.D. Dissertation. Stanford University, Stanford, CA, USA. Advisor(s) Guibas, Leonidas J.
- Delio Vicini, Sébastien Speierer, and Wenzel Jakob. 2021. Path Replay Backpropagation: Differentiating Light Paths Using Constant Memory and Linear Time. *ACM Trans. Graph.* 40, 4, Article 108 (jul 2021), 14 pages. <https://doi.org/10.1145/3450626.3459804>
- Delio Vicini, Sébastien Speierer, and Wenzel Jakob. 2022. Differentiable Signed Distance Function Rendering. *ACM Trans. Graph.* 41, 4, Article 125 (jul 2022), 18 pages. <https://doi.org/10.1145/3528223.3530139>
- Daniel Vlasic, Pieter Peers, Ilya Baran, Paul Debevec, Jovan Popović, Szymon Rusinkiewicz, and Wojciech Matusik. 2009. Dynamic Shape Capture Using Multi-View Photometric Stereo. In *ACM SIGGRAPH Asia 2009 Papers (Yokohama, Japan) (SIGGRAPH Asia ’09)*. Association for Computing Machinery, New York, NY, USA, Article 174, 11 pages. <https://doi.org/10.1145/1661412.1618520>
- Michael Vorländer. 2020. *Auralization: Fundamentals of Acoustics, Modelling, Simulation, Algorithms and Acoustic Virtual Reality*. Springer International Publishing, Cham. <https://doi.org/10.1007/978-3-030-51202-6>

- Thomas Walker, Octave Mariotti, Amir Vaxman, and Hakan Bilen. 2024. Explicit neural surfaces: learning continuous geometry with deformation fields. In *Proceedings of the 2nd NeurIPS Workshop on Symmetry and Geometry in Neural Representations (Proceedings of Machine Learning Research, Vol. 228)*, Sophia Sanborn, Christian Shewmake, Simone Azeglio, and Nina Miolane (Eds.). PMLR, Cambridge, MA, 324–345. <https://proceedings.mlr.press/v228/walker24a.html>
- Bruce Walter, Stephen R. Marschner, Hongsong Li, and Kenneth E. Torrance. 2007. Microfacet Models for Refraction through Rough Surfaces. In *Rendering Techniques (EGSR'07)*, Jan Kautz and Sumanta Pattanaik (Eds.). Eurographics Association, Saarbrücken, Germany, 195–206.
- Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. 2021. NeuS: learning neural implicit surfaces by volume rendering for multi-view reconstruction. In *Proceedings of the 35th International Conference on Neural Information Processing Systems (NIPS '21)*. Curran Associates Inc., Red Hook, NY, USA, Article 2081, 13 pages.
- Yifan Wang, Aleksander Holynski, Xiuming Zhang, and Xuaner Zhang. 2023a. Sun-Stage: Portrait Reconstruction and Relighting Using the Sun as a Light Stage. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Los Alamitos, CA, USA, 20792–20802. <https://doi.org/10.1109/CVPR52729.2023.01992>
- Yu-Chen Wang, Chris Wyman, Lifan Wu, and Shuang Zhao. 2023b. Amortizing Samples in Physics-Based Inverse Rendering Using ReSTIR. *ACM Trans. Graph.* 42, 6, Article 214 (Dec. 2023), 17 pages. <https://doi.org/10.1145/3618331>
- R. E. Wengert. 1964. A simple automatic derivative evaluation program. *Commun. ACM* 7, 8 (Aug. 1964), 463–464. <https://doi.org/10.1145/355586.364791>
- Lance Williams. 1978. Casting Curved Shadows on Curved Surfaces. *SIGGRAPH Comput. Graph.* 12, 3 (aug 1978), 270–274. <https://doi.org/10.1145/965139.807402>
- Markus Worchel and Marc Alexa. 2023a. Differentiable Rendering of Parametric Geometry. *ACM Trans. Graph.* 42, 6, Article 232 (Dec. 2023), 18 pages. <https://doi.org/10.1145/3618387>
- Markus Worchel and Marc Alexa. 2023b. Differentiable Shadow Mapping for Efficient Inverse Graphics. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Los Alamitos, CA, USA, 142–153.
- Markus Worchel and Marc Alexa. 2025. Moment Bounds are Differentiable: Efficiently Approximating Measures in Inverse Rendering. *ACM Trans. Graph.* 44, 4, Article 80 (July 2025), 21 pages. <https://doi.org/10.1145/3730899>
- Markus Worchel, Rodrigo Diaz, Weiwen Hu, Oliver Schreer, Ingo Feldmann, and Peter Eisert. 2022. Multi-View Mesh Reconstruction with Neural Deferred Shading. In *2022*

- IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Los Alamitos, CA, USA, 6177–6187. <https://doi.org/10.1109/CVPR52688.2022.00609>
- Markus Worchel, Ugo Finnendahl, and Marc Alexa. 2025. Radiative Backpropagation with Non-Static Geometry. In *Eurographics Symposium on Rendering*, Beibei Wang and Alexander Wilkie (Eds.). The Eurographics Association, Saarbrücken, Germany, 1–10. <https://doi.org/10.2312/sr.20251198>
- Markus Worchel, Marcus Zepp, Weiwen Hu, Oliver Schreer, Ingo Feldmann, and Peter Eisert. 2020. Ernst Grube: A Contemporary Witness and His Memories Preserved with Volumetric Video. In *Eurographics Workshop on Graphics and Cultural Heritage*, Michela Spagnuolo and Francisco Javier Melero (Eds.). The Eurographics Association, Saarbrücken, Germany. <https://doi.org/10.2312/gch.20201300>
- Lifan Wu, Guangyan Cai, Ravi Ramamoorthi, and Shuang Zhao. 2021. Differentiable time-gated rendering. *ACM Trans. Graph.* 40, 6, Article 287 (dec 2021), 16 pages. <https://doi.org/10.1145/3478513.3480489>
- Lifan Wu, Nathan Morrical, Sai Praveen Bangaru, Rohan Sawhney, Shuang Zhao, Chris Wyman, Ravi Ramamoorthi, and Aaron Lefohn. 2025. Unbiased Differential Visibility Using Fixed-Step Walk-on-Spherical-Caps And Closest Silhouettes. *ACM Trans. Graph.* 44, 4, Article 79 (July 2025), 16 pages. <https://doi.org/10.1145/3731174>
- Peiyu Xu, Sai Bangaru, Tzu-Mao Li, and Shuang Zhao. 2023. Warped-Area Reparameterization of Differential Path Integrals. *ACM Trans. Graph.* 42, 6, Article 213 (Dec. 2023), 18 pages. <https://doi.org/10.1145/3618330>
- Kai Yan, Cheng Zhang, Sébastien Speierer, Guangyan Cai, Yufeng Zhu, Zhao Dong, and Shuang Zhao. 2025. Image-space Adaptive Sampling for Fast Inverse Rendering. In *Proceedings of the Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers (SIGGRAPH Conference Papers '25)*. Association for Computing Machinery, New York, NY, USA, Article 66, 11 pages. <https://doi.org/10.1145/3721238.3730627>
- Kai Yan, Shuang Zhao, et al. 2020. PSDR-JIT. <https://github.com/andyankai/psdr-jit>.
- Lior Yariv, Jiatao Gu, Yoni Kasten, and Yaron Lipman. 2021. Volume rendering of neural implicit surfaces. In *Proceedings of the 35th International Conference on Neural Information Processing Systems (NIPS '21)*. Curran Associates Inc., Red Hook, NY, USA, Article 367, 11 pages.
- Lior Yariv, Yoni Kasten, Dror Moran, Meirav Galun, Matan Atzmon, Ronen Basri, and Yaron Lipman. 2020. Multiview neural surface reconstruction by disentangling geometry and appearance. In *Proceedings of the 34th International Conference on Neural Information Processing Systems (Vancouver, BC, Canada) (NIPS '20)*. Curran Associates Inc., Red Hook, NY, USA, Article 210, 11 pages.

- Shinyoung Yi, Donggun Kim, Kiseok Choi, Adrian Jarabo, Diego Gutierrez, and Min H. Kim. 2021. Differentiable transient rendering. *ACM Trans. Graph.* 40, 6, Article 286 (dec 2021), 11 pages. <https://doi.org/10.1145/3478513.3480498>
- Cem Yuksel. 2022. High-performance polynomial root finding for graphics. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 5, 3 (2022), 1–15.
- Tizian Zeltner, Fabrice Rousselle, Andrea Weidlich, Petrik Clarberg, Jan Novák, Benedikt Bitterli, Alex Evans, Tomáš Davidovič, Simon Kallweit, and Aaron Lefohn. 2024. Real-time Neural Appearance Models. *ACM Trans. Graph.* 43, 3, Article 33 (June 2024), 17 pages. <https://doi.org/10.1145/3659577>
- Tizian Zeltner, Sébastien Speierer, Iliyan Georgiev, and Wenzel Jakob. 2021. Monte Carlo Estimators for Differential Light Transport. *ACM Trans. Graph.* 40, 4, Article 78 (jul 2021), 16 pages. <https://doi.org/10.1145/3450626.3459807>
- Chong Zeng, Yue Dong, Pieter Peers, Hongzhi Wu, and Xin Tong. 2025b. RenderFormer: Transformer-based Neural Rendering of Triangle Meshes with Global Illumination. In *Proceedings of the Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers (SIGGRAPH Conference Papers '25)*. Association for Computing Machinery, New York, NY, USA, Article 48, 11 pages. <https://doi.org/10.1145/3721238.3730595>
- Yunfan Zeng, Guangyan Cai, and Shuang Zhao. 2025a. A Survey on Physics-based Differentiable Rendering. arXiv:2504.01402 [cs.GR] <https://arxiv.org/abs/2504.01402>
- Cheng Zhang, Bailey Miller, Kai Yan, Ioannis Gkioulekas, and Shuang Zhao. 2020. Path-Space Differentiable Rendering. *ACM Trans. Graph.* 39, 4, Article 143 (July 2020), 19 pages. <https://doi.org/10.1145/3386569.3392383>
- Cheng Zhang, Lifan Wu, Changxi Zheng, Ioannis Gkioulekas, Ravi Ramamoorthi, and Shuang Zhao. 2019. A differential theory of radiative transfer. *ACM Trans. Graph.* 38, 6, Article 227 (Nov. 2019), 16 pages. <https://doi.org/10.1145/3355089.3356522>
- Kai Zhang, Fujun Luan, Zhengqi Li, and Noah Snavely. 2022. IRON: Inverse Rendering by Optimizing Neural SDFs and Materials from Photometric Images. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Los Alamitos, CA, USA, 5555–5564. <https://doi.org/10.1109/CVPR52688.2022.00548>
- Xiuming Zhang, Pratul P. Srinivasan, Boyang Deng, Paul Debevec, William T. Freeman, and Jonathan T. Barron. 2021. NeRFactor: Neural Factorization of Shape and Reflectance under an Unknown Illumination. *ACM Trans. Graph.* 40, 6, Article 237 (dec 2021), 18 pages. <https://doi.org/10.1145/3478513.3480496>
- Ziyi Zhang, Nicolas Roussel, and Wenzel Jakob. 2023. Projective Sampling for Differentiable Rendering of Geometry. *ACM Trans. Graph.* 42, 6, Article 212 (Dec. 2023), 14 pages. <https://doi.org/10.1145/3618385>

- Bowen Zhi, Alisha Sharma, Dmitry N. Zotkin, and Ramani Duraiswami. 2023. A Differentiable Image Source Model for Room Acoustics Optimization. In *2023 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*. IEEE Computer Society, Los Alamitos, CA, USA, 1–5. <https://doi.org/10.1109/WASPAA58266.2023.10248140>