FORWARD AND INVERSE RENDERING WITH
GRADIENT-BASED OPTIMIZATION

A Dissertation
Presented to the Faculty of the Graduate School
of Cornell University
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy

by
Fujun Luan
August 2021
Photorealistic rendering is a fundamental component of computer graphics. To render photorealistic 2D images that faithfully represent our 3D world, efforts have been devoted to developing compact scientific models and efficient algorithms that are capable of simulating lighting and physics to capture appearance using the geometry, reflectance and illumination. Inverse Rendering, on the other hand, focuses on recovering unknown scene attributes from photograph measurements by optimization, typically through analysis-by-synthesis technique\(^1\), or more recently, differentiable rendering. In this thesis, we propose solutions to a few such challenging forward and inverse graphics problems, as described below.

The first work is a forward rendering (SIGGRAPH 2020) approach to speeding up Markov chain Monte Carlo (MCMC) rendering with derivatives from recent rendering engines that support end-to-end differentiation. In this work, we build upon Langevin dynamics and propose a suite of Langevin Monte Carlo algorithms with gradient-based adaptation for efficient photorealistic rendering of scenes with complex light transport effects, such as caustics, interreflections, and occlusions.

In inverse rendering, we develop multiple solutions for recovering material and shape. In our second work (ICCP 2020) we focus on learning-based inverse

\(^1\)Analysis by synthesis is the process that aims to analyze a signal or image by reproducing it using a model.
subsurface scattering. Given images of translucent objects, of unknown shape and lighting, we aim to use learning to infer the optical parameters controlling subsurface scattering of light inside the objects. With physics-based priors in the learning framework, we obtain strong improvements in both parameter estimation accuracy and appearance reproduction compared to traditional networks.

Our third work is a unified shape and appearance reconstruction framework (EGSR 2021) using differentiable rendering for 3D object scanning. We tackle this problem by introducing a new analysis-by-synthesis technique capable of producing high-quality reconstructions through robust coarse-to-fine optimization and physics-based differentiable rendering. We demonstrate the effectiveness of our method on real-world objects captured with handheld cameras that outperforms previous state-of-the-art approaches.

In Appendix A, we tackle the challenge of automatically generating procedural representation of fiber-based yarn models for cloth rendering (SIGGRAPH 2016, EGSR 2017). Appendix B has slightly deviated from rendering and focuses on image style transfer techniques (CVPR 2017, EGSR 2018).

With recent advances in physics-based differentiable rendering, we have taken first steps to speeding up MCMC forward rendering with first-order gradients, improving learning-based inverse subsurface scattering, and introducing an end-to-end differentiable rendering pipeline for high-quality handheld object scanning. As for next steps, a promising future direction for computer graphics would be learning-based neural rendering methods that aim for AR/VR-based applications (e.g., NeRF [147]). Exploring differentiable rendering for improving traditional 3D reconstruction would be another promising topic, thanks to the theoretical and practical breakthroughs in the corresponding field.
Fujun Luan was born on September 7th, 1992 in Fushun, China. After graduating from Tsinghua University in 2015 with a bachelor degree in computer science, he moved to Ithaca, New York and has been studying for his doctorate degree in computer science at Cornell University since then.
ACKNOWLEDGEMENTS

First, and most importantly, I would like to thank my advisor Kavita Bala. Six years ago, she gave me the opportunity to join the Graphics and Vision group at Cornell, and I am deeply grateful for her supervision, responsiveness and support through my entire Ph.D. life. Through the years, she has taught me how to think as a scientific researcher, collaborate with people, and stay on track. It has been a great privilege to work with KB. I truly appreciate and am indebted to her continuous support and patient guidance.

I would like to thank Shuang Zhao, who has provided research suggestions and guidance for almost every piece of work in this thesis. I am sincerely grateful for his support.

I have been very fortunate to work many great collaborators. This thesis would not have been possible without their efforts. I want to thank all my collaborators: Kavita Bala, Shuang Zhao, Ioannis Gkioulekas, Sylvain Paris, Eli Shechtman, Zhao Dong, Noah Snavely, Kevin Matzen, Chengqian Che, Kai Zhang, and Qianqian Wang. Special thanks go to Ioannis and Sylvain for their technical insights and guidance for two of my most satisfied research projects by far. I would also like to thank my undergraduate research advisors, Kun Xu, Shi-Min Hu and Pradeep Sen, for introducing me into computer graphics.

I would like to thank the other members of my Special Committee — Steve Marschner, David Bindel, and Kilian Weinberger — for their time and constructive suggestions.

I was also lucky to have a couple of internships during the summers. I am grateful for my excellent internship mentors: Sylvain Paris and Eli Shechtman at Adobe Research for guiding me through the two style transfer projects. Kevin Matzen at Facebook Research for guiding me in the multi-view stereo project,
and Zhao Dong at Facebook Reality Labs for guiding me in the 3D capture using differentiable rendering.

I want to thank all my lab mates and friends at Cornell. I owe thanks to Pramook Khungurn and Bruce Walter for their help in my early cloth projects. I’d love to thank Xinyue Joy Zhang for accompanying me with so many unforgettable moments and mental support in my Ph.D. life. I also enjoyed playing Xbox/PS4 games with Rundong Wu or eating delicious food with Nathan Yan, Yan Wang, and Lequn Wang during spare time. They have created a stimulating environment and enriched my experience at Cornell. I want to express my sincere thanks to them for their support in both my life and work.

Last but not least, I am deeply indebted to my mother, Guiying Wang, and maternal grandmother, Sulai Lu, for their unconditional love and unaltering support through all these years. I would not have been able to make it this far without their support throughout my life. I also miss the old days my father and I played Super Mario Bros, Contra and Battle City on NES together. This thesis is dedicated to them.
TABLE OF CONTENTS

Biographical Sketch .......................................................... iii
Acknowledgements .............................................................. iv
Table of Contents .............................................................. vi
List of Tables ................................................................. viii
List of Figures ................................................................. ix

1 Introduction ................................................................. 1
1.1 Contributions ............................................................ 3
1.2 Outline of the Dissertation .............................................. 5

2 Background ................................................................. 7
2.1 The Rendering Equation ............................................... 7
2.2 Radiative Transfer Equation .......................................... 8
2.3 Monte Carlo Integration ............................................... 10
2.4 Markov chain Monte Carlo Integration ............................. 12

3 Previous Work ............................................................. 14
3.1 Differentiable Rendering .............................................. 14
3.2 Inverse Subsurface Scattering ........................................ 16
3.3 Shape and Reflectance Reconstruction .............................. 17

4 Langevin Monte Carlo Rendering with Gradient-based Adaptation 21
4.1 Introduction .............................................................. 21
4.2 Overview ................................................................. 24
4.3 Langevin Monte Carlo ................................................ 27
4.4 Online Adaptation ...................................................... 32
4.4.1 Ensuring ergodicity ................................................ 37
4.5 Cache-Driven and Hybrid Adaptation ............................... 42
4.6 Implementation Details ............................................... 47
4.7 Experiments ............................................................. 49
4.8 Conclusion and Discussion ............................................. 53

5 Learning-based Inverse Subsurface Scattering .......................... 57
5.1 Introduction .............................................................. 58
5.2 Overview ................................................................. 60
5.2.1 Problem Setting .................................................... 61
5.2.2 Analysis by Synthesis ............................................. 63
5.3 Inverse Transport Networks ......................................... 65
5.3.1 Relationship to Prior Work ...................................... 67
5.3.2 Training ITNs ........................................................ 69
5.3.3 Differentiable Monte Carlo Volume Rendering .............. 70
5.3.4 Stochastic Optimization ........................................... 72
5.3.5 Post-Learning Refinement ....................................... 73
## 5.4 Overcoming Similarity Relations ................................................. 74
## 5.5 Experiments ................................................................................. 76
## 5.6 Conclusion .................................................................................. 86

### 6 Shape and Reflectance Recovery using Differentiable Rendering 89

#### 6.1 Introduction ............................................................................. 89
#### 6.2 Overview .................................................................................. 90
#### 6.3 Our Method ............................................................................... 92
    6.3.1 Forward and Differentiable Rendering ................................ 92
    6.3.2 Analysis-by-Synthesis Optimization .................................... 96
    6.3.3 Regularization ....................................................................... 98
    6.3.4 Improving Robustness .......................................................... 100
#### 6.4 Results ..................................................................................... 102
    6.4.1 Evaluations and Comparisons .............................................. 103
    6.4.2 Reconstruction Results ....................................................... 108
#### 6.5 Conclusion and Discussion ....................................................... 109

### 7 Conclusion and Future Work 110

### A Fitting and Rendering of Procedural Yarn Models 113

#### A.1 Introduction ............................................................................ 113
#### A.2 Overview ............................................................................... 116
    A.2.1 Fabric Micro-Appearance Modeling .................................... 117
    A.2.2 Procedural Modeling of Yarn Geometry ............................... 118
#### A.3 Model Fitting .......................................................................... 122
    A.3.1 Input ................................................................................... 124
    A.3.2 Ply Twisting and Cross Section Estimation .......................... 125
    A.3.3 Fiber Classification .............................................................. 127
    A.3.4 Fitting Fiber Distribution and Migration .............................. 129
#### A.4 Our Flyaway Fiber Model ......................................................... 132
    A.4.1 Model Description .............................................................. 132
    A.4.2 Model Fitting ....................................................................... 138
#### A.5 Results ..................................................................................... 139
#### A.6 Realization-Minimizing Rendering of Procedural Textiles ......... 145
#### A.7 Conclusion ............................................................................... 145

### B Image Style Transfer for Photos and Paintings 148

#### B.1 Introduction ............................................................................ 148
#### B.2 Method .................................................................................... 150
#### B.3 Deep Photo Style Transfer ....................................................... 153
#### B.4 Deep Painting Harmonization .................................................. 154
#### B.5 Conclusions ............................................................................. 155

### Bibliography 159
### LIST OF TABLES

5.1 **Network performances**: Average MSE for individual scattering parameters, as well as $L_2$ and $1-\text{MS-SSIM}$ image appearance errors, for five different networks. 81

5.2 **Network performances**: Average $L_2$ and $1-\text{MS-SSIM}$ image appearance errors, for five different networks, on the novel scene used to emphasize generalization performance. 83

5.3 **Network performances on initialization of analysis by synthesis**: Average MSE for individual scattering parameters over iterations for five different networks. 86

6.1 **Rendering performance.** We report the rendering cost in seconds (averaged across 100 times) of each differentiable renderer in resolution $512 \times 512$ and 4 samples per pixel on a Titan RTX graphics card. 102

A.1 **List of parameters** used by the procedural yarn model. Parameters shown in purple are per-ply attributes. Those shown in blue are specified for every fiber by independently sampling the corresponding statistical distributions. In particular, per-fiber flyaway parameters (i.e., $R_{\text{loop}}^{\text{max}}, R_{\text{min}}^{\text{hair}}, R_{\text{span}}^{\text{hair}},$ and $\theta_{\text{span}}^{\text{hair}}$) are sampled from normal distributions (see §A.4 and Table A.3) while others (i.e., $R_i, \theta_i, \text{and } \theta_i^{(0)}$) are drawn uniformly (§A.2.2). 124

A.2 **Fiber scattering parameters** using the model introduced by Khungurn et al. [102] for generating all renderings in Figure A.15. We manually adjusted the parameter values since finding optimal ones automatically is orthogonal to our current work. For better appearance matching, we recommended using advanced inverse rendering techniques such as [102]. 138

A.3 **Our fitted parameter values** for all results shown in Figure A.15. Positive and negative $\alpha$ values indicate counter-clockwise and clockwise (ply/fiber) twisting, respectively. We assume, without loss of generality, that $R_{\text{max}} = 1$. For flyaway distribution parameters, paired numbers in parentheses indicate the mean and standard deviation of the corresponding normal distributions. When generating yarns using Algorithm 1, all parameters not mentioned here (e.g., $R_i, \theta_i, \theta_i^{(0)}$) are drawn uniformly from their domains. 143

A.4 **Model size and rendering times** for all our results. The render times are measured by rendering a 1280 pixels wide image, at 128 samples per pixel, on an Intel server with four Xeon X7560 eight-core CPUs. 145
### LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>A very complex outdoor <em>landscape</em> scene rendered using PBRT [179], featuring 23,241 unique plant models. Scene courtesy of Jan-Walter Schliep, Burak Kahraman, and Timm Dapper.</td>
<td>1</td>
</tr>
<tr>
<td>1.2</td>
<td>An illustration figure of <em>forward rendering</em> and <em>inverse rendering</em> [1]. Scene courtesy: &quot;bed classic&quot; from jiraniano.</td>
<td>2</td>
</tr>
<tr>
<td>2.1</td>
<td>Examples of volumetric rendering for liquids and fabrics [60, 135].</td>
<td>9</td>
</tr>
<tr>
<td>2.2</td>
<td>With a symmetric proposal function ( \pi ), the “uphill” samples ( z ) are always accepted as next sample, while the “downhill” jumps are accepted with probability ( a = \frac{f(z)}{f(x^{(t-1)})} ).</td>
<td>12</td>
</tr>
<tr>
<td>4.1</td>
<td>Equal-time (20 minutes) comparison between MEMLT, MMLT, RJMLT, H2MC and two variants of our methods. The scene presents complex glossy and specular interreflections with difficult visibilities. The reference (Ref.) is rendered by BDPT in roughly a day. MEMLT suffers from correlated noise in the glass bottle region due to insufficient local exploration. MMLT and RJMLT both have severe noise because they sometimes get trapped in regions with hard-to-find features using Kelemen-style isotropic mutations. H2MC is more efficient with the help of anisotropic Gaussian mutations, but the computational overhead from Hessian computations and dense matrix operations results in a low sample budget and insufficient exploration. Our Langevin Monte Carlo methods efficiently address these challenges by exploiting first-order gradient information to robustly balance the tradeoff between adaptation and cost.</td>
<td>22</td>
</tr>
<tr>
<td>4.2</td>
<td><strong>2D sampling example:</strong> We compare various MCMC sampling algorithms on a simple two-dimensional anisotropic geometry, typical of those encountered in the primary sample space near caustics. Starting at the same initial position (top of the curve), we show 2048 samples and acceptance ratios produced by each algorithm. “Mmt.” refers to momentum, and “diag.” and “full” to diagonal and full preconditioning, respectively.</td>
<td>33</td>
</tr>
<tr>
<td>4.3</td>
<td><strong>Ablation study:</strong> We use the <em>door</em> scene to compare different versions of our proposed MALA with online adaptation. In (a-f), we show equal-sample (64 samples-per-pixel) renderings produced using the small-step perturbation of Kelemen et al. [101], and MALA with different combinations of momentum (“mmt.”) and diagonal (“diag.”) or full preconditioning. In (g-h), we show converged (20,000 samples-per-pixel) renderings produced using MALA with online adaptation, with and without diminishing adaptation (“DA”). All images include error maps with respect to a reference rendering in (i) produced using BDPT.</td>
<td>35</td>
</tr>
</tbody>
</table>
4.4 **2D Gaussian mixture model**: We compare the performance of MALA with online and hybrid adaptation, on a two-dimensional disconnected geometry, typical of the isolated “islands” of primary sample space. Both algorithms use full preconditioning and large-step mutation probability $T_{\text{global}} = 0.1$. Each row shows sample distributions when running the algorithm for (from left to right) 0.5K, 1K, 2K, 4K, 8K and 16K samples. In the bottom row, the insets show the evolution of the anisotropic transition kernels at two different points in space. Please zoom in to better compare the results.

4.5 **Equal-time comparisons**: We compare MEMLT [84], MMLT [66], RJMLT [21], H2MC [120] and two of our algorithms, across several scenes with complex illumination and occlusion, glossy caustics and interreflections.

4.6 **Equal-time comparisons**: We compare MEMLT [84], MMLT [66], RJMLT [21], H2MC [120] and two of our algorithms, across several scenes with complex illumination and occlusion, glossy caustics and interreflections.

4.7 **Motion blur effects**: Equal-time comparison of H2MC and our hybrid adaptation on the *cars* (3 minutes) and *necklace* (7 minutes) scenes.

5.1 **Inverse transport networks**: (a) Traditional autoencoders use two networks, encoder and decoder, to learn to predict parameters from images. (b) Inverse transport networks replace the decoder with a differentiable Monte Carlo renderer, to improve the generalization and physical accuracy of the predictions. During training, the renderer is provided with the material parameter output by the encoder network, as well as with groundtruth geometry and illumination, to perform forward and backward evaluations of an additional appearance-matching regularization term used to learn the network weights. During testing, the encoder network is used on its own, without the renderer. It takes as input a single, fully uncalibrated (unknown geometry and illumination) image, and produces as output a set of material parameters.
5.2 **Monte Carlo rendering:** (a) Monte Carlo forward rendering estimates radiometric measurements by randomly sampling light paths and aggregating their radiance contributions. By evaluating additional terms for each path, we can use the same procedure to estimate derivatives of measurements with respect to physical scene parameters. (b) Example renderings of the derivatives of a scene with subsurface scattering with respect to different material parameters. The top row shows derivatives estimated by our differentiable renderer, and the bottom row shows derivatives estimated using finite differences. 

5.3 **Weight maps:** We use per-pixel weights equal to each pixel’s distance from the nearest image edge, in order to emphasize image pixels where similarity relations are violated.

5.4 **Datasets and architecture:** (a) We render 30,000 images for the task of homogeneous inverse scattering. These are split into training and testing sets of different shapes, illuminations, and materials. Rendered grayscale images under the same illumination and shape but with different material parameters are combined to form color images. (b) We use these datasets to train and evaluate regressor and inverse transport networks of the same architecture.

5.5 **Images rendered with predicted material parameters:** Each column corresponds to a different input image drawn from our synthetic testing set. The last column shows images rendered under the novel scene used to emphasize generalization performance. For each image, different rows compare the groundtruth (row 1) to images rendered using the parameters predicted by the five networks we evaluate (rows 2-6).

5.6 **Post-learning refinement:** We use predictions from two ITNs to initialize analysis by synthesis optimization. The top row shows how parameters change during gradient descent, in the \( \{ \sigma_s, \sigma_a, \bar{c} \} \) parameter space, and in its 2D projection \( \{ \sigma_s, \bar{c} \} \) colored by reduced scattering coefficient value \( \sigma' \). The bottom row compares groundtruth to renderings using the initial and final parameter estimations from the green optimization trajectory.

5.7 **Real photographs:** We take photographs of two translucent objects, a silicone cube (top) and a soap bar (bottom), under unknown shape and illumination (left column). We use three ITNs to predict scattering parameters for the objects. We evaluate the predicted parameters by rendering images for an approximate reconstruction of the original scenes (middle and right column).
6.1 **Differentiable rendering:** (a) To properly differentiate the intensity $I_j$ of a pixel $P_j$ (illustrated as red squares) with respect to object geometry, a boundary integral over $\Delta P_j$ (illustrated as the orange curve) needs to be calculated. (b) We perform Monte Carlo edge sampling [118, 252] by (i) sampling points (illustrated as small discs) from pre-computed discontinuity curves $\Delta P$, and (ii) accumulating their contributions in the corresponding pixels (e.g., the orange samples contribute to the pixel $P_j$).

6.2 **Comparison** with COLMAP [207] and Kinect Fusion [161] using synthetic inputs. The COLMAP results (c) are generated using 50 RGB images (a) with exact camera poses; the *KF-High* (d1) and *KF-Low* (d2) results are created using 50 ground-truth depth images (b) and low-resolution noisy ones, respectively. Our method (e), when initialized with *KF-Low* (d2) and using RGB inputs (a), produces much more accurate geometries than COLMAP and *KF-High*. Similar to Figure 6.5, the numbers indicate average point-to-mesh distances.

6.3 **Ablation study** on our mesh loss of Eq. (6.9) and coarse-to-fine framework. Using the identical initializations and optimization settings, we show geometries (rendered under a novel view) optimized with (i) various components of the mesh loss; and (ii) the coarse-to-fine process disabled. Similar to Figure 6.5, the number below each result indicates the average point-to-mesh distance.

6.4 **Ablation study** on our material loss of Eq. (6.10). Using identical initial reflectance maps and optimization configurations, models optimized with various components of the material loss neglected are rendered under a novel environmental illumination. On the right of each reconstruction result, we show the optimized reflectance maps (from top to bottom: diffuse albedo, specular albedo, and roughness).

6.5 **Comparison** with a few baselines. We render all reconstructed geometries using Phong shading and visualize depth errors (wrt. the ground-truth geometry). The number below each result indicates the average point-to-mesh distance capturing the Euclidean accuracy [87] of the reconstructed geometry (normalized to have a unit bounding box).

6.6 **Reconstruction results** using synthetic inputs: We obtain the initial meshes (in the left column) using Kinect Fusion [161] with low-resolution and noisy depths. Our method successfully recovers both geometric and reflectance details, producing high-fidelity results under novel rendering conditions.
6.7 **Reconstruction results** of real-world objects: Similar to the synthetic examples in Figure A.15, our technique recovers detailed object geometries and reflectance details, producing high-quality results under novel viewing and illumination conditions.  

6.8 **Applications**: The high-quality models generated by our technique can be used for 3D digital modeling (top) and object insertion in augmented reality (bottom).  

A.1 We present a new technique to automatically generate procedural representations of yarn geometry. Based on geometric measurements of physical yarn samples (a), our approach fits statistical representations of fiber geometry that closely match reality (b). The four yarns in (a, b) from top to bottom are cotton, rayon, silk, and polyester. Our fitted models can populate realistic fiber-level details into yarn-based fabric models to significantly improve the quality of the rendered fabrics (c-top vs. c-middle (ours)). Our procedural models carry high-level synthetic information (e.g., twisting and hairiness) which offers easy editability (c-bottom).  

A.2 **Rendered fabrics** with identical yarn-level geometry but varying fiber-level properties: (a) original; (b) modified fiber twisting; (c) modified fiber distribution; (d) increased flyaway variance. These fiber-level variations change appearance at much larger scales.  

A.3 **Fabric structure**: A yarn normally consists of multiple sub-stands called plies. Each ply in turn contains tens to hundreds of micron-diameter fibers.  

A.4 **Our parameter fitting pipeline**. This pipeline is approximately in the opposite order of the procedural yarn generation algorithm (Algorithm 1) since it is fitting parameters.  

A.5 **Acquisition of yarn geometries**. (a) We stack multiple yarns and CT scan them. (b) Volumetric density information for one of the scanned yarns. (c) Extracted fiber-based representation of the yarn. (d) Tracked ply centers.  

A.6 **Fitting ply cross-sections** (3-ply example). (left) We place an ellipse at each ply center. The short axis of this ellipse points toward the center of a polygon (shown in gray with dashed boundaries) formed by all ply centers. (right) We then rotate and translate all plies (in plane) with the corresponding fiber points. These transformed 2D fiber locations will then be used to fit the ply cross-sections (Figure A.7).
A.7 Fitting ply cross-sections (cont’d). A density visualization of transformed fiber centers (Figure A.6) from a sequence of cross-sectional planes. We use this density information to determine the shape of ply cross-sections.

A.8 Fiber classification: we classify all input fibers into two categories: regular and flyaway (with alien ones removed). (a) Scatter plot of \((d_{\min}, d_{\max})\) for each fiber. (b, c, d) 2D fiber visualizations where the horizontal and vertical axes show \(z_i\) and \(\|\langle x_i, y_i\rangle\|_2\) for each fiber vertex, respectively. The dashed line indicates the threshold beyond which a fiber is considered flyaway.

A.9 The two types of flyaway fibers in our model.

A.10 Comparison of loop-type flyaway fiber models: (a) the migration-based approach [208] lacks the generality to represent occasionally occurring loops; (b) our model has sufficient representative power to capture such fiber geometry.

A.11 Generation of loop-type flyaway fibers. (a) We randomly pick a vertex maximizing the distance to the ply center (indicated with an orange circle). (b) We then select all vertices belonging to the same migration cycle (shown in green) and scale their radii so that the selected vertex has radius \(R_{\text{loop}}\) afterwards.

A.12 Comparison of flyaway fiber models (hair-type): (a) measured yarn geometry (ground truth); (b) procedural yarn with no flyaway fibers; (c) procedural yarn with Perlin noise-based flyaways [208]; (d) procedural with flyaways generated with our model. Fiber migration is increased in (c) for simulating loop-type effects.

A.13 Simplifications made by our hair-type flyaway fiber model: (top) our hair-type flyaways all have arc shapes; (bottom) real hair-type fibers have more complex trajectories.

A.14 Processing Flyaway Fiber. (a) One fiber can contain both hair and loop components. (b) We split each flyaway fiber into multiple hair and loop segments.

A.15 Parameter fitting results. (a) CT-measured yarn geometries. (b) Procedural yarn geometries fitted using our approach. (c) Photographs of real-world yarns. (d) Full renderings of our procedural yarns under similar lighting/viewing conditions. Our fitted parameter values are shown in Table A.3.
A.16 Full textiles with fiber-level geometries added procedurally using our method. The original parameters used in the three rows are: “Silk 1” + “Silk 2” (first), “Rayon 3” + “Rayon 4” (second) and “Polyester 1” (third). The edited silk example is obtained by multiplying ply and fiber twisting (i.e., $\alpha_{\text{ply}}$ and $\alpha$) by $10\times$. The modified rayon model results from tripling flyaway densities $\rho_{\text{loop}}$ and $\rho_{\text{hair}}$ as well as doubling $R_{\text{loop}}^{\text{max}}$ and $R_{\text{span}}^{\text{hair}}$. The altered polyester example is created by halving $e_Y$ (the short axis of the ellipse) while doubling $e_X$ (the long axis).

A.17 Rendered results (woven and knitted): Our method enables physically-based rendering of these large textiles with fiber-level details. Please see Table A.4 for data size and performance statistics.

B.1 Given a reference style image (a) and an input image (b), we seek to create an output image of the same scene as the input, but with the style of the reference image. The Neural Style algorithm [52] (c) successfully transfers colors, but also introduces distortions that make the output look like a painting, which is undesirable in the context of photo style transfer. In comparison, our result (d) transfers the color of the reference style image equally well while preserving the photorealism of the output. On the right (e), we show 3 insets of (b), (c), and (d) (in that order). Zoom in to compare results.

B.2 Comparison of our method against Neural Style and CNNMRF. Both Neural Style and CNNMRF produce strong distortions in their synthesized images. Neural Style also entirely ignores the semantic context for style transfer. CNNMRF tends to ignore most of the texture in the reference style image since it uses nearest neighbor search. Our approach is free of distortions and matches texture semantically.

B.3 Comparison of our method against Reinhard et al. [191] and Pitié [181]. Our method provides more flexibility in transferring spatially-variant color changes, yielding better results than previous techniques.

B.4 Example painting harmonization results with insets on proposed composite for unadjusted cut-and-paste, four state-of-the-art methods and our results. We show that our method captures both spatial and inter-scale color and texture and produces harmonized results on paintings with various styles. Zoom in for details.
CHAPTER 1
INTRODUCTION

Rendering is a fundamental component of computer graphics. At the highest level of abstraction, rendering is the process that maps 3D scene information to 2D images through raytracing algorithms, typically path tracing [89] and its variants. Photorealistic rendering involves physically based light transport techniques that attempt to simulate reality of our 3D world; that is, they use principles of physical laws to describe and model light interactions on surfaces or in participating media. As a result, modern graphics renderers can often create an image of a sophisticated 3D scene with sufficient amount of geometric and shading details that is almost indistinguishable from a photograph of the same scene. See below Fig. 1.1 as an example for such realistic renderings.

Figure 1.1: A very complex outdoor landscape scene rendered using PBRT [179], featuring 23,241 unique plant models. Scene courtesy of Jan-Walter Schliep, Burak Kahraman, and Timm Dapper.
To achieve realistic *forward rendering* of a 3D scene, it is crucial that one can acquire, model and computationally reproduce the shape and appearance of scene objects. However, it is often a heavy and time-consuming task for digital artists to manually create and model these high-quality 3D objects even with modern graphics creative tools. Ideally, we would like to design an algorithm or a pipeline, that is capable of automatically and accurately reconstructing 3D scene objects from a set of real-world captures (e.g., 2D photo measurements), recovering, both its geometric details and appearance properties.

This is where *inverse rendering* comes into play\(^1\). Inverse rendering, is the problem of estimating one or more of illumination, material properties and geometry from observed measurements (e.g., 2D images, or CT data).

![Diagram](image)

**Figure 1.2:** An illustration figure of *forward rendering* and *inverse rendering* [1]. Scene courtesy: “bed classic” from jiraniano.

As shown in Fig. 1.2, if we think of *forward rendering* as the process that takes 3D scene information as input \(x\), and renders an output 2D image \(y = f(x)\) through a black-box rendering function \(f\). Then *inverse rendering* is the reverse of this process — taking 2D photos as input \(y\), and optimizing for the unknown

\(^1\)It is also sometimes called the *analysis-by-synthesis* technique, or more recently, *differentiable rendering* in related literature.
3D scene parameters \( x = f^{-1}(y) \). This was often achieved through the *analysis-by-synthesis* technique, which is the process that aims to analyze a signal or image by reproducing it using a (typically parametric) model. The objective is then to find the value of the model parameters that synthesize the closest image possible in the span of the model. This results in an optimization problem that defines an objective/cost function and a searching algorithm for the procedure. When the rendering function \( f \) becomes *differentiable*, a new door opens to us with a family of gradient-based optimization techniques to search the parameter space more efficiently using algorithms such as Adam [105]. In fact, *differentiable rendering* has gradually become a popular research field and there are many novel topics in computer graphics and computer vision [1] that allow the gradients of 3D objects to be calculated and propagated for various applications like 3D data collection and annotation. We also made use of differentiable rendering in a few pieces of work in this thesis.

In summary, we focus on improving both *forward rendering* and *inverse rendering* in this thesis, and the key to these work is to take advantage of the gradients available in the differentiable renderers. Specifically, differentiable rendering has bridged the gap between these two parts (which used to be largely separated), making them connected and more tightly coupled than in the past.

### 1.1 Contributions

This thesis focuses on forward & inverse rendering applications and presents solutions to a few open problems. Our contributions are as follows:

**Contribution 1 (Chapter 4, forward rendering, SIGGRAPH 2020)** focuses
on speeding up forward rendering in the Markov chain Monte Carlo (MCMC) rendering framework using differentiable rendering. With the recent proliferation of rendering engines that support end-to-end differentiation, we revisit the use of gradient-based MCMC for physics-based rendering. Inspired by Langevin dynamics, we propose adaptation mechanisms that improve the exploration of path space to a suite of Langevin Monte Carlo rendering algorithms. We use theoretical results from controlled MCMC processes to ensure the ergodicity and correctness of these adaptations.

Contribution 2 (Chapter 5, inverse rendering, ICCP 2020) focuses on inferring the material optical parameters of translucent objects with inverse rendering. We begin by proposing a physics-aware learning pipeline that aims to combine the computational efficiency of learning-based approaches with the generality of analysis-by-synthesis approaches for inverse scattering. To train the neural networks efficiently, we pair them with a new efficient and physically-accurate Monte Carlo differentiable rendering engine. We demonstrate the rendering layer helps improving the generalization significantly, resulting in both higher parameter estimation accuracy and closer appearance reproduction.

Contribution 3 (Chapter 6, inverse rendering, EGSR 2021, CVPR 2021) tackles the problem of 3D object reconstruction, including both the geometric details as well as spatially-varying reflectance properties. With recent advances in physics-based differentiable rendering, we can calculate unbiased derivatives of the rendering loss function with respect to geometry and material parameters. In this work, we develop a robust coarse-to-fine optimization strategy with several shape and material priors, and demonstrate that our technique can produce reconstructions with higher quality than previous methods such as COLMAP or
Kinect Fusion.

Appendix A (cloth rendering, SIGGRAPH 2016, EGSR 2017) proposes addresses the realism of cloth rendering. We present a new end-to-end fitting pipeline that automatically generates the procedural representations of yarn geometry that can accurately match our CT measurements with fiber-level geometric details.

Appendix B (style transfer, CVPR 2017, EGSR 2018)) is not directly relevant to rendering, which focuses on image editing with different exposure, contrast and brush strokes through learning-based priors.

1.2 Outline of the Dissertation

This thesis is organized as follows:

- **Chapter 2** focuses our background material on forward and inverse rendering that is common to all the following chapters — including the theory of physically based light transport, Monte Carlo integration and Markov chain Monte Carlo integration. We start with the rendering equation [89] that describes the physical laws of light transport in a scene, followed by radiative transfer equation that extends the rendering framework to participating media (such as fog, fire and cloud). Lastly, to solve for these rendering equations, we briefly review the Monte Carlo integration and Markov chain Monte Carlo techniques.

- **Chapter 3** reviews related work and surveys previous approaches on
Markov chain Monte Carlo rendering, subsurface scattering, and 3D reconstruction for geometry and reflectance.

- **Chapter 4 (SIGGRAPH 2020)** discusses the possibilities of differentiable renderers for speeding up forward image synthesis, by introducing a suite of Langevin Monte Carlo rendering algorithms with gradient-based adaptation and demonstrating its improvements on a large variety of complex scenes.

- **Chapter 5 (ICCP 2020)** introduces our learning-based framework for inverse subsurface scattering. We discuss ways for making networks robust to ambiguities and demonstrate significant improvements on generalization.

- **Chapter 6 (EGSR 2021)** describes our 3D object reconstruction project using handheld mobile devices. It builds upon the Monte Carlo differentiable rendering pipeline and jointly optimizes for geometry mesh and spatially-varying BRDF textures from a set of uncalibrated flashlight photos.

- **Chapter 7** presents our conclusion and suggestions for future research directions.

- **Appendix A (SIGGRAPH 2016, EGSR 2017)** presents our cloth project for fitting procedural yarn models from CT scan data.

- **Appendix B (CVPR 2017, EGSR 2018)** presents our image style transfer projects for photos and paintings.
In this chapter, we review relevant background material on light transport simulation, Monte Carlo integration and subsurface scattering.

2.1 The Rendering Equation

The light transport equation (LTE), also called the rendering equation \[89\], is the governing equation that describes the equilibrium distribution of radiance in a scene. The rendering equation is given by:

\[
L_o(x, \omega_o) = L_e(x, \omega_o) + \int_{\Omega} f_r(x, \omega_o, \omega_i) \, L_i(x, \omega_i) \cos \theta_i \, d\omega_i.
\] (2.1)

The equation tells us that, the outgoing radiance \( L_o(x, \omega_o) \) from a surface point \( x \) in direction \( \omega_o \) can be separated into two components: the emitted radiance (if any) \( L_e(x, \omega_o) \) and the reflected radiance, which we denote as \( L_r(x, \omega_o) \). The latter \( L_r(x, \omega_o) \) term is an integral over the hemisphere \( \Omega \) of the incoming radiance \( L_i(x, \omega_i) \) weighted by a function \( f_r(x, \omega_o, \omega_i) \), called the bidirectional reflectance distribution function (BRDF), and the cosine of the angle between the surface normal at point \( x \) and \( \omega_i \).

Radiance is the light power per unit area per unit solid angle, with its unit watt per meter square per steradian \( \frac{W}{m^2\text{sr}} \). This is a directional quantity, and it can be shown that under geometric optics, radiance is conserved along a ray in vacuum space. That is, \( L(x, \omega) = L(x', \omega) \) if \( x' \) lies along the ray \( (x, \omega) \).
property holds for scenes that only contain surfaces, and we will discuss the case when there is participating media presented in a later section.

The bidirectional reflectance distribution function (BRDF) gives a formula for describing reflection from a surface. If we define irradiance $E$ as the power of incoming light energy from all directions to a point $x$, which has the unit of power per unit area $\frac{W}{m^2}$, we can show that the reflected differential outgoing radiance along a given direction $\omega_o$ is proportional to the irradiance

$$dL_o(x, \omega_o) \propto dE(x, \omega_o). \quad (2.2)$$

This means that under the linearity assumption from geometric optics we have

$$f_r(x, \omega_o, \omega_i) = \frac{dL_o(x, \omega_o)}{dE(x, \omega_o)} = \frac{dL_o(x, \omega_o)}{L_i(x, \omega_i) \cos \theta_i \, d\omega_i}. \quad (2.3)$$

The above equation is due to the Lambert’s cosine law:

$$dE(x, \omega_i) = L_i(x, \omega_i) \cos \theta_i \, d\omega_i. \quad (2.4)$$

Physically based BRDFs will satisfy two important properties:

- **Reciprocity**: For all pairs of $\omega_i$ and $\omega_o$, $f_r(x, \omega_o, \omega_i) = f_r(x, \omega_i, \omega_o)$.

- **Energy conservation**: The total energy reflected by the surface is less than or equal to the energy of incident light. For all directions $\omega_o$, we have

$$\int_{\Omega} f_r(x, \omega_o, \omega') \cos \theta' \, d\omega' \leq 1.$$  

### 2.2 Radiative Transfer Equation

When the scene presents not only surfaces, but also participating media such as translucent solids (jade, wax, skin), liquids (milk, wine, juice) or medium (fog,
Figure 2.1: Examples of volumetric rendering for liquids and fabrics [60, 135].

smoke, cloud, fire), light will also be absorbed or scattered along the ray. This is governed by the radiative transfer equation (RTE). This equation, in the form often used in computer graphics, is

\[
\frac{dL_o(x, \omega_o)}{ds} = -\sigma_a(x, \omega_o)L_o(x, \omega_o) - \sigma_s(x, \omega_o)L_o(x, \omega_o) \\
+ \sigma_s(x, \omega_o) \int_{S^2} f_p(x, \omega_i, \omega_o)L_i(x, \omega_i) \, d\omega_i + Q(x, \omega_o) \\
= -\sigma_t(x, \omega_o)L_o(x, \omega_o) + \sigma_s(x, \omega_o) \int_{S^2} f_p(x, \omega_i, \omega_o)L_i(x, \omega_i) \, d\omega_i + Q(x, \omega_o).
\]  

(2.5)

Here, \(\sigma_a\) is the absorption coefficient, \(\sigma_s\) is the scattering coefficient, and \(\sigma_t = \sigma_s + \sigma_a\) is the extinction coefficient. The absorption coefficient denotes the amount of light absorbed along the ray, the scattering coefficient denotes the amount of light scattered into other directions along the ray. \(f_p\) is called the phase function, and \(Q\) is the volume source term representing emitted radiance from the medium.

We further make a number of simplifications. In our later chapters we focus on non-emissive medium, effectively dropping the volumetric light source term \(Q\) and setting it to zero. In addition, we set \(\alpha := \sigma_s/\sigma_t\), which is often termed the single-scattering albedo. Intuitively, the extinction coefficient specifies the volume density, and the scattering albedo denotes its color. When these coefficients
remains constant across the entire medium, we call it \textit{homogeneous medium}, and \textit{heterogeneous medium} otherwise.

To evaluate $L_0(x, \omega_0)$ at some location $x$ inside a medium, we can integrate both sides of Eq. 2.5 along a line segment of length $s$ with its endpoints $x$ and $y := x - s\omega$, yielding the \textit{volumetric rendering equation}:

$$L_0(x, \omega_0) = \int_y^x \tau(x', x) \left( \sigma_t(x') \int_{S^2} f_p(x', \omega_i, \omega_0) L_i(x', \omega_i) \, d\omega_i \right) \, dx' + \tau(y, x) L_0(y, \omega_0),$$

(2.6)

where

$$\tau(x_1, x_2) := \exp \left( - \int_{x_1}^{x_2} \sigma_t(x) \, dx \right)$$

(2.7)

captures the \textit{transmittance} between $x_1$ and $x_2$.

\section{2.3 Monte Carlo Integration}

The above rendering equations (Eq. 2.1 and Eq. 2.5) are often very difficult to solve analytically, due to many factors in light transport such as the visibility occlusions, glossy interreflections and caustics. Therefore, Monte Carlo techniques are usually used to compute the integral numerically.

The idea of Monte Carlo integration is probabilistically estimating the value of a given integral

$$I = \int_{\Omega} f(x) \, dx.$$  

(2.8)

To solve for this integral, samples are generated according to a probability density function (PDF) $p(x)$. And the expectation of random variable
\[ F_N = \frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i)}{p(X_i)} \] would be

\[
\mathbb{E}[F_N] = \mathbb{E} \left[ \frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i)}{p(X_i)} \right] \\
= \frac{1}{N} \mathbb{E} \left[ \sum_{i=1}^{N} \frac{f(X_i)}{p(X_i)} \right] \\
= \frac{1}{N} \sum_{i=1}^{N} \int_{\Omega} \frac{f(x)}{p(x)} p(x) \, dx \\
= \frac{1}{N} \sum_{i=1}^{N} \int_{\Omega} f(x) \, dx \\
= \int_{\Omega} f(x) \, dx = I. \tag{2.9}
\]

Hence, this is an unbiased estimator of the integral \( I \). The strong law of large numbers guarantees that \( F_N \) converges to the true value \( I \) almost surely when \( N \to \infty \). However, the random variable \( F_N \) might have a high variance, and it often requires many samples (i.e., a large \( N \)) to produce a reasonable estimate.

As a result, the choice of PDF \( p(x) \) has significant influence on the variance of the estimator. If \( p(x) \propto f(x) \) for every \( x \), then we will have a perfect zero-variance estimator and one sample is enough \( F_1 = f(X_1)/p(X_1) = I \). Unfortunately, sampling w.r.t. this perfect PDF \( p(x) \) is often not feasible, and *importance sampling* is a technique addressing this problem.

Importance sampling, to put it simply, is to approximate the perfect PDF \( p(x) \) with some another PDF \( q(x) \), for which we have prior knowledge and can design it by construction (glossy BRDF sampling, next-event estimation, etc.), yielding a low-variance estimator for our problem in practice.
2.4 Markov chain Monte Carlo Integration

Another similar approach for importance sampling is called *metropolis sampling*, which falls into the category of Markov chain Monte Carlo (MCMC) methods. This technique generates samples by mutating/perturbing previous samples and probabilistically accepts or rejects the new samples based on their contribution, resulting in a Markov chain. An example is shown in Fig. 2.2.

Figure 2.2: With a symmetric proposal function $\pi$, the “uphill” samples $z$ are always accepted as next sample, while the “downhill” jumps are accepted with probability $\alpha = \frac{f(z)}{f(x(t-1))}$.

Again, given the integral $I = \int_{\Omega} f(x) \, dx$, MCMC methods based on the Metropolis-Hastings algorithm [68, 146] will use a proposal function $\pi : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}_\geq$, and first samples a proposed sample $z \sim \pi(x(t-1) \rightarrow \cdot)$. Then, the proposed sample is accepted with an accept probability, defined as:

$$\alpha \left( x^{(t-1)} \rightarrow z \right) := \min \left\{ 1, \frac{f(z) \pi(z \rightarrow x^{(t-1)})}{f(x^{(t-1)}) \pi(x^{(t-1)} \rightarrow z)} \right\}. \tag{2.10}$$

If the proposed sample $z$ is accepted, then the current sample is updated to $x^{(t)} = z$, otherwise it stays with the last sample $x^{(t)} = x^{(t-1)}$. Each new sample is used to update the Monte Carlo estimation of the integral, with a PDF $p(x^{(t)}) = f(x^{(t)})/L$, where $L$ is a constant normalization term.
This technique was first introduced to solve the light transport problem by Veach et al. [231], often termed the *Metropolis light transport* (MLT) algorithm, which typically has better convergence rate than standard path tracing, especially for difficult light paths such as specular-diffuse-specular (SDS) paths or caustics.
The previous chapter discusses the fundamental theory of light transport simulation for physically based rendering. This chapter summarizes related previous work on differentiable rendering, subsurface scattering, and 3D reconstruction for shape and reflectance.

3.1 Differentiable Rendering

Differentiable rendering. Analysis by synthesis requires formulating a new forward model, as well as analytically computing its derivatives, specifically for each reconstruction problem. Differentiable renderers such as OpenDR [132] have been proposed to remove this obstacle, by providing a general-purpose framework that can be differentiated with respect to arbitrary scene parameters. To ensure analytical differentiability, these approaches use approximate forward models, ignoring complex light transport effects such as inter-reflections and subsurface scattering. This makes these methods inapplicable to situations where these effects are dominant. Differentiable Monte Carlo rendering algorithms overcome this limitation, by estimating derivatives of images while accounting for all light transport effects. These algorithms were first introduced in the context of differentiation with respect to scattering parameters, and used for accurate inverse scattering [60, 259, 103, 56, 114]. Since then, they have been extended to allow differentiation with respect to, and recovery of, arbitrary scene parameters, including surface reflectance [131, 10], geometry [226], and visibility and pose [119, 253, 164].
**Differentiable rendering of meshes.** We now briefly review differentiable rendering techniques closely related to our work. For a more comprehensive summary, please see the survey by Kato et al. [96].

Specialized differentiable renderers have long existed in computer graphics and vision [60, 59, 227, 11, 29]. Recently, several general-purpose ones [118, 164] have been developed.

A key technical challenge in differentiable rendering is to estimate gradients with respect to object geometry (e.g., positions of mesh vertices). To this end, several approximated methods (e.g., [133, 128, 189]) have been proposed. Unfortunately, inaccuracies introduced by these techniques can lead to degraded result quality. On the contrary, Monte Carlo edge sampling [118, 252], which we use for our differentiable renderer, provides unbiased gradient estimates capable of producing higher-quality reconstructions.

**Neural rendering.** The success of neural rendering [223, 147, 248, 163, 217, 216, 225] has generated significant excitement. In particular, NeRF [147] enables photo-realistic novel view synthesis by representing scenes as radiance fields via multi-layer-perceptrons (MLPs) and fitting these to a collection of input views. While NeRF represents scenes as volumetric opacity fields, other recent methods like DVR [163] and IDR [248] are surface-based. In these three works, appearance is represented by a single MLP that takes a 3D point (and a view direction), and outputs a color. Hence, their appearance model is essentially a surface light field [242] that treats objects as light sources. Such an approach works well for novel view synthesis, but does not disentangle material and lighting, and hence is not suitable for physics-based relighting and
material editing. Other approaches learn an appearance space [145, 122, 139] from Internet photos of landmarks captured under diverse lighting, but are not physics-based and cannot generalize to arbitrary new lighting.

3.2 Inverse Subsurface Scattering

**Subsurface scattering.** Volume rendering algorithms in computer vision and computer graphics are predominantly based on the radiative transfer framework [148]. Existing techniques include Monte Carlo volume rendering and photon mapping algorithms [167]. Inverse scattering techniques include approaches relying on single-scattering approximations [70, 156, 47, 62], which are appropriate for optically thin media; as well as diffusion-based approaches [86, 237, 173, 38, 80, 153, 218, 151], suitable for optically thick media. Intermediate cases, so-called turbid scattering, can be tackled using techniques based on combinations of analysis by synthesis and differentiable Monte Carlo rendering, as discussed below. Recently, deep learning techniques have been used to accelerate forward scattering simulation [92, 233]. To the best of our knowledge, we are the first to consider deep learning techniques for the inverse scattering problem.

**Analysis by synthesis in physics-based vision.** Analysis by synthesis is a core methodology for recovering physical scene parameters from images, which conceptually comprises three steps: (i) formulate an approximate image formation (or forward rendering) model as a function of the scene parameters; (ii) analytically derive an expression for the derivative of the forward model with respect to those parameters; (iii) use gradient-based optimization to solve an analysis-
by-synthesis objective comparing measured and synthesized images. This approach has been used to recover shape [51, 34], material [201, 150, 165], and illumination [142], independently or jointly [138, 14, 249, 130, 170].

Combining deep learning with rendering. Recently, a number of works have proposed using renderers not for analysis-by-synthesis, but as parts of learning architectures. The most popular approach is to replace the decoder network in an auto-encoder pipeline [234, 106] with a rendering layer that takes as input the parameters predicted by the encoder and produces as output synthesized images. This encoder-renderer architecture was first proposed by Wu et al. [245], who used a non-photorealistic renderer to achieve categorical interpretability. The same conceptual architecture was later used, together with approximate (direct lighting) physics-renderers for inference of physical scene parameters such as surface normals, illumination, and reflectance [143, 212, 126, 224, 97, 36, 214, 3, 53, 124, 162, 211]. Inspired from these works, we apply the encoder-renderer architecture to the problem of inverse scattering, using for the first time a physically-accurate Monte Carlo differentiable renderer instead of an approximate one. Finally, differentiable Monte Carlo rendering has also been combined with neural networks in the context of discovering adversarial example scenes for classification tasks [119].

3.3 Shape and Reflectance Reconstruction

Shape reconstruction. Reconstructing object geometry has been a long-standing problem in computer vision.
Multi-view Stereo (MVS) recovers the 3D geometry of sufficiently textured objects using multiple images of an object by matching feature correspondences across views and optimizing photo-consistency (e.g., [210, 236, 48, 206]).

Shape from Shading (SfS) relates surface normals to image intensities [74, 79, 186, 187, 137, 67]. Unfortunately, these methods have difficulties handling illumination changes, non-diffuse reflectance, and textureless surfaces.

Photometric Stereo (PS) takes three or more images captured with a static camera and varying illumination or object pose, and directly estimate surface normals from measurements [243, 72, 262, 228, 172, 184, 185]. These methods typically do not recover reflectance properties beyond diffuse albedo.

Reflectance reconstruction. Real-world objects exhibit richly diverse reflectance that can be described with spatially-varying bidirectional reflectance distribution functions (SVBRDFs). Traditional SVBRDF acquisition techniques rely on dense input images measured using light stages or gantry (e.g, [141, 112, 73, 40, 30, 41, 93]). To democratize the acquisition, some recent works exploit the structure (e.g., sparsity) of SVBRDF parameter spaces to allow reconstructions using fewer input images (e.g., [250, 39, 244, 263, 104, 176]). Additionally, a few recent works have been introduced to produce plausible SVBRDF estimations for flat objects using a small number of input images (e.g., [4, 3, 76, 49, 37, 63]). Despite their ease of use, these techniques cannot be easily generalized to handle more complex shapes.

Joint estimation of shape and reflectance. Several prior works jointly estimate object shape and reflectance. Higo et al. [71] presented a plane-sweeping
method for albedo, normal and depth estimation. Xia et al. [247] optimized an apparent normal field with corresponding reflectance. Nam et al. [154] proposed a technique that alternates between material-, normal-, and geometry-optimization stages. Schmitt et al. [204] perform joint estimation using a hand-held sensor rig with and 12 point light sources. Bi et al. [20] use six images and optimize object geometry and reflectance in two separate stages.

All these methods either rely on MVS for geometry reconstruction or perform alternative optimization of shape and reflectance, offering little to no guarantee on qualities of the reconstruction results. In contrast, as shown in the later chapters, we formulate the problem as a unified analysis-by-synthesis optimization, ensuring locally optimal results.

**Material and environment estimation.** To estimate material properties, most prior works require scenes to be captured under varying illumination [18, 19, 20, 205, 154, 247, 39, 64]. They either place the object of interest on a mechanical turntable and capture it with a fixed camera [247, 39], or move a camera with co-located flashlight to capture a static object from multiple viewpoints [18, 19, 20, 205, 154]. The varying illumination yields rich cues for inferring material properties and geometry [2]. For environment estimation from multi-view images, prior works [175, 131] factorize scene appearance into diffuse image and surface reflectance map given high-quality geometry from RGBD sensors. The surface reflectance map entangles the material and lighting, because it represents the distant environmental illumination convolved with an object’s specular BRDF, hence preventing relighting. In contrast to a global environment map, Azinovic et al. [11] model lighting as surface emissions, and use a Monte Carlo differentiable renderer to jointly estimate material properties.
and surface emissions from multi-view images conditioned on scanned geometry and object segmentation masks. Other work seeks to predict illumination, materials and shape from a single image via learning-based priors [13, 123].

**Joint post refinement.** Given the initial geometry and appearance from RGBD sensors, Maier et al. [137] and Zollhofer et al. [265] jointly refine geometry and appearance by assuming Lambertian BRDF and incorporating shading cues. They pre-compute a lighting model based on spherical harmonics [188], then fix it while optimizing the shape and diffuse albedo. They adopt voxelized SDFs as their geometric representation. Assuming known illumination, Oxholm and Nishino [170] also exploit reflectance cues to refine geometry computed via visual hulls. In contrast to these prior works, our method does not require scanned geometry or a known environment map. Instead, we estimate material and lighting parameters, as well as geometry and surface normals in an end-to-end fashion.
CHAPTER 4
LANGEVIN MONTE CARLO RENDERING WITH GRADIENT-BASED ADAPTATION

In this chapter, we explore the possibilities of improving forward rendering with the gradients available in recent differentiable renderers. We demonstrate that differentiable rendering can benefit physics-based light transport algorithms using gradient-based MCMC.

4.1 Introduction

The development of general-purpose and efficient global illumination algorithms is one of the foundational problems in computer graphics. Physics-based Monte Carlo rendering algorithms [44, 180] can accurately simulate complicated light transport effects such as caustics, strong interreflections, subsurface scattering, and motion blur. They achieve this by aggregating intensity contributions from a large number of randomly generated light paths, representing the different ways in which light can propagate through the scene that is being simulated. As the complexity of the underlying transport effects increases, retaining efficiency requires using Markov chain Monte Carlo (MCMC) techniques [232]: unlike traditional Monte Carlo algorithms, which generate independent paths through local importance sampling, MCMC algorithms use Markov chains to create statistically-correlated paths. This allows them to efficiently perform both a global exploration of the space of possible paths, searching for clusters of paths with high contributions to the image, and a local exploration of such newly discovered clusters.
Figure 4.1: Equal-time (20 minutes) comparison between MEMLT, MMLT, RJMLT, H2MC and two variants of our methods. The scene presents complex glossy and specular interreflections with difficult visibilities. The reference (Ref.) is rendered by BDPT in roughly a day. MEMLT suffers from correlated noise in the glass bottle region due to insufficient local exploration. MMLT and RJMLT both have severe noise because they sometimes get trapped in regions with hard-to-find features using Kelemen-style isotropic mutations. H2MC is more efficient with the help of anisotropic Gaussian mutations, but the computational overhead from Hessian computations and dense matrix operations results in a low sample budget and insufficient exploration. Our Langevin Monte Carlo methods efficiently address these challenges by exploiting first-order gradient information to robustly balance the tradeoff between adaptation and cost.

The success of MCMC algorithms depends critically on the design of proposal distributions for producing new path samples given previously sampled ones. Whereas so-called path space MCMC rendering algorithms directly modify the path within the scene, primary sample space algorithms [101] modify the random numbers provided as input to a black-box path tracing algorithm. Operating in the mathematically-tractable space of real random numbers provides great flexibility for designing proposal distributions, and allows incorporating general-purpose proposals that have found success in other application areas of MCMC inference. This includes state-of-the-art gradient-based proposal distributions such as Langevin Monte Carlo (LMC) [200] and Hamiltonian Monte Carlo (HMC) [42]: essentially, these work by performing one (LMC) or multiple (HMC) steps of a noisy gradient ascent procedure, that guides the sampling pro-
cess towards parts of the primary sample space corresponding to local maxima of the path contribution function.

The use of gradient-based MCMC techniques for physics-based rendering has been hindered by two main factors: First, generating path samples requires the ability to differentiate complicated path-tracing algorithms, corresponding to sequences of reflection, transmission, and volume scattering events. Second, producing high-quality samples requires additional expensive computation, such as computing, factorizing, and inverting the Hessian matrix of the path contribution function for each new path sample [120]. Analogous to the use of the Hessian in Newton’s method, this helps the gradient ascent procedure underlying the sampling process adapt to the local geometry of the primary sampling space.

The recent proliferation of rendering engines that support end-to-end differentiation [6, 118, 253, 164, 28] helps overcome the first of these two factors, and motivates us to revisit the use of gradient-based MCMC for physics-based rendering. Our focus is on developing proposal distributions based on LMC, coupled with adaptation mechanisms to improve the exploration of the primary sampling space. To make these LMC with adaptation processes efficient enough for rendering, we make the following contributions:

- Inspired by state-of-the-art stochastic gradient descent (SGD) algorithms [105], we propose adaptation mechanisms that introduce minimal computational overhead compared to standard LMC. These include a Hessian approximation and a momentum vector that can be computed by performing only scalar operations on first-order gradients, sidestepping the need for expensive second-order differentiation and matrix operations.
• We combine these adaptation mechanisms with a new caching scheme, that stores previously computed gradients, and uses them to guide the sampling process in a way that facilitates both local and global exploration.
• We use theoretical results on controlled MCMC processes [7], to ensure the ergodicity and correctness of these combinations.

These contributions result in three MCMC rendering algorithms that significantly outperform state-of-the-art techniques, resulting in an average MSE improvement of $3 \times$ on equal-time comparisons across a large variety of challenging scenes, and $7 \times$ for scenes with motion blur.

4.2 Overview

We begin by introducing the problem setting, mathematical notation, and algorithmic concepts for Markov Chain Monte Carlo rendering. These will serve as background for developing our contributions.

**Primary sample space.** Our focus will be on physics-based Monte Carlo rendering algorithms that use the primary sample space formulation of light transport, first proposed by Kelemen et al. [101]. This assumes that we have available a path generation algorithm, such as path tracing [90], particle tracing [8, 45], or bidirectional path tracing (BDPT) [230, 108]. Such an algorithm deterministically transforms a sequence of $r(B)$ uniform random variables into a light path of length $B \in [2, \ldots \infty)$, that is, an ordered sequence $\bar{x} = x_1 \rightarrow \cdots \rightarrow x_B$ of three-dimensional points on the surfaces or in the participating media that make up
the scene. We can then define the space of all light paths as the path space \( \mathcal{P} \), the union of hypercubes needed to generate all of the path space as the primary sample space \( \mathcal{U} = \bigcup_{B=0}^{\infty} [0, 1]^{r(B)} \), and the path generation algorithm as a map \( S : \mathcal{U} \rightarrow \mathcal{P} \) between the two spaces.

With this notation at hand, Kelemen et al. [101] proposed expressing radiometric measurements as integrals of the form:

\[
I = \int_{\mathcal{U}} f(S(\bar{u})) \left| \frac{d\mu(\bar{u})}{d\bar{u}} \right| d\bar{u} = \int_{\mathcal{U}} \tilde{f}(\bar{u}) d\bar{u},
\]

(4.1)

where the measurement contribution function \( f \) describes the radiance that flows through the light path \( \bar{x} = S(\bar{u}) \), and \( \tilde{f}(\bar{u}) \equiv f(S(\bar{u})) \left| \frac{d\mu(\bar{u})}{d\bar{u}} \right| \). This integral is a re-parameterization of the path-space integral formulation of light transport [229, 232], with the Jacobian term in Equation (4.1) accounting for the change of variables from path space to primary sample space. Under technical conditions that are typically satisfied by transformations \( S \) corresponding to standard path tracing algorithms, we can write \( \tilde{f}(\bar{u}) = f(\bar{x}) / p(\bar{x}) \), where \( p(\bar{x}) \) is the probability density of the path \( \bar{x} = S(\bar{u}) \) [6, 101]. The primary sample space has been extended to account for multiple importance sampling [66] and mixed-space techniques [21, 169, 171].

Monte Carlo rendering algorithms approximate \( I \) by aggregating weighted contributions from randomly generated sequences \( \bar{u}^{(i)} \):

\[
\langle I \rangle = \frac{1}{T} \sum_{i=1}^{T} \tilde{f}(\bar{u}^{(i)}) p(\bar{u}^{(i)}). \tag{4.2}
\]

Typically, sampling in primary sample space is done using MCMC techniques, though independent sampling techniques also exist [152, 260, 190].
**Markov chain Monte Carlo rendering.** MCMC rendering algorithms, first introduced by Veach and Guibas [232], deviate from traditional Monte Carlo rendering in terms of how they generate samples for the estimator of Equation (4.2): instead of generating each sample independently, they use Markov chains to create correlated sequences of samples. In the following, we describe MCMC rendering in primary sample space [101].

Most commonly, MCMC rendering techniques are based on the Metropolis-Hastings algorithm [146, 69]. This assumes that we have available a proposal distribution $T : \mathcal{U} \times \mathcal{U} \rightarrow \mathbb{R}_\geq$. Given the current primary sample sequence $\bar{u}^{(t)}$, we first sample a proposed sequence $\tilde{v} \sim T \left( \bar{u}^{(t)} \rightarrow \cdot \right)$. Then, the proposed sequence is accepted with an acceptance probability:

$$
\alpha \left( \bar{u}^{(t)} \rightarrow \tilde{v} \right) = \min \left\{ 1, \frac{\tilde{f}(\bar{v}) T \left( \tilde{v} \rightarrow \bar{u}^{(t)} \right)}{\tilde{f}(\bar{u}^{(t)}) T \left( \bar{u}^{(t)} \rightarrow \tilde{v} \right)} \right\}.
$$

If the proposed sequence is accepted, then the current sequence is updated as $\bar{u}^{(t+1)} = \tilde{v}$, otherwise $\bar{u}^{(t+1)} = \bar{u}^{(t)}$. Each new sequence $\bar{u}^{(t+1)}$ is used to update the Monte Carlo estimate of Equation (4.2), with a probability distribution $\hat{p}(\bar{u}^{(t+1)}) = \tilde{f}(\bar{u}^{(t+1)})/L$, where $L$ is a constant estimated using independent Monte Carlo rendering.

A Monte Carlo estimate (Equation (4.2)) formed using sequences sampled with the above Markov chain procedure will be unbiased and consistent if the Markov chain satisfies two conditions: First, it has a stationary distribution that is proportional to the function $\tilde{f}$. Second, it is ergodic, meaning that it will converge to this stationary distribution from all initial sequences $\bar{u}_0$. As a consequence of the use of the Metropolis-Hastings rule of Equation (4.3), both

---

$^1$We denote by $T (\cdot)$ the probability distribution a sample $\tilde{v}$ is drawn from, and by $T (\tilde{v})$ the corresponding probability density function evaluated at $\tilde{v}$. 

26
conditions will be satisfied if we select a proposal distribution that satisfies:
\[ \mathcal{T}(\bar{u} \to \bar{v}) > 0, \text{ for all } \bar{u} \text{ and } \bar{v} \text{ such that } \tilde{f}(\bar{u}) > 0 \text{ and } \tilde{f}(\bar{v}) > 0. \]

The selection of the proposal distribution \( \mathcal{T} \) critically affects the performance of MCMC rendering techniques. As observed by Kelemen et al. [101], it is important to combine two types of proposals: first, large-step mutation proposals \( \mathcal{T}_{\text{global}} \) that can easily reach any sequence from any other sequence; second, small-step perturbation proposals \( \mathcal{T}_{\text{local}} \) that generate a proposed sequence by only making small modifications to the previous sequence. Concretely:

\[
\mathcal{T}(\bar{u} \to \bar{v}) = \pi_{\text{global}} \mathcal{T}_{\text{global}}(\bar{u} \to \bar{v}) + \pi_{\text{local}} \mathcal{T}_{\text{local}}(\bar{u} \to \bar{v}), \tag{4.4}
\]

where \( \pi_{\text{global}}, \pi_{\text{local}} > 0, \pi_{\text{global}} + \pi_{\text{local}} = 1 \). Such combinations guarantee ergodicity, and can efficiently explore the target space, which typically consists of many disjointed subsets (informally termed “islands” by Kelemen et al. [101]): The large-step mutation enables global exploration by jumping from one island to another, and the small-step perturbation enables local exploration by moving within one island. The large-step mutation proposal is typically implemented by uniformly sampling a sequence independently of the previous one: \( \mathcal{T}_{\text{global}}(\bar{u} \to \bar{v}) = \mathcal{T}_{\text{global}}(\bar{v}) \). We use Langevin Monte Carlo to develop a new class of small-step perturbation proposals.

### 4.3 Langevin Monte Carlo

At the basis of our proposed rendering algorithms is a class of MCMC techniques known as *Langevin Monte Carlo* (LMC) [200]. Even though LMC techniques have become popular in other application areas of MCMC, they remain
relatively unexplored in physics-based rendering—we are aware of one exception, the H2MC algorithm [120], which however is derived from the related Hamiltonian Monte Carlo (HMC) [42, 158, 17] framework. To keep our paper self-contained, we provide a brief review of LMC.

LMC techniques take their name from the Langevin diffusion process [111]. This is a continuous-time Markov process \( \vec{u}(t), t \in [0, \infty) \), that, given a positive function \( f : \mathcal{U} \to \mathbb{R}_+ \geq \), satisfies the stochastic differential equation [168]:

\[
d\vec{u}(t) = \frac{1}{2} \nabla \vec{u} \tilde{L}\left(\vec{u}(t)\right) \, dt + \sqrt{\epsilon} \vec{w},
\]

(4.5)

The evolution of \( \vec{u}(t) \) is controlled by a deterministic drift term proportional to the gradient of \( \tilde{L}(\vec{u}(t)) \equiv \log f(\vec{u}(t)) \), and the random Brownian motion term \( \vec{w}(t), t \in [0, \infty) \). This Markov process is ergodic, with a stationary distribution proportional to \( f(\vec{u}(t)) \).

In practice, to simulate the Langevin diffusion, it is necessary to use a discrete approximation, such as the Euler-Maruyama discretization [140]. Given a temporal step-size \( \epsilon > 0 \), this corresponds to the following discrete-time Markov chain:

\[
\vec{u}^{(t+1)} = \vec{u}^{(t)} + \frac{1}{2} \epsilon \nabla \vec{u} \tilde{L}\left(\vec{u}^{(t)}\right) + \sqrt{\epsilon} \vec{w},
\]

(4.6)

where \( \vec{w} \sim \mathcal{N}(\cdot; 0, I) \) is a normally-distributed random variable. Langevin Monte Carlo algorithms use this discrete Markov chain, which is equivalent to a gradient ascent procedure with injected Gaussian noise: the gradient drift term drives the chain towards points of \( \mathcal{U} \) where \( \tilde{L} \) has high density, whereas the injected noise prevents the chain from collapsing to just the (local) maximum.

Due to discretization error, the Markov chain of Equation (4.6) is not guaranteed to converge to the same stationary distribution as the continuous process.

28
This can be corrected by using the Metropolis-Hasting rule of Equation (4.3) to accept or reject states of the chain. This approach, known as the Metropolis-adjusted Langevin algorithm (MALA), corresponds to using a Gaussian proposal distribution with mean $\bar{u}^{(t)} + \frac{1}{2}\epsilon\nabla_a \tilde{L}(\bar{u}^{(t)})$ and covariance matrix $\epsilon I$:

$$T_{\text{MALA}}(\bar{u}^{(t)} \rightarrow \bar{v}) = \mathcal{N}\left(\bar{v}; \bar{u}^{(t)} + \frac{1}{2}\epsilon\nabla_a \tilde{L}(\bar{u}^{(t)}), \epsilon I\right).$$  \hspace{1cm} (4.7)

**Adaptation.** The Markov chain of Equation (4.6) implicitly assumes that the coordinates of $\bar{u}$ are uncorrelated and have approximately equal variance; that is, that $\tilde{L}$ is locally isotropic. This can result in slow convergence when these conditions are violated. This behavior can be ameliorated by modifying Equation (4.6) to use a positive-definite *preconditioning matrix* $M(\bar{u}^{(t)})$ [199]:

$$\bar{u}^{(t+1)} = \bar{u}^{(t)} + \frac{1}{2}\epsilon M(\bar{u}^{(t)}) \nabla_a \tilde{L}(\bar{u}^{(t)}) + \sqrt{\epsilon} \sqrt{M(\bar{u}^{(t)})} \bar{w}. \hspace{1cm} (4.8)$$

The matrix $\sqrt{M(\bar{u}^{(t)})}$ can be computed as the Cholesky decomposition of $M(\bar{u}^{(t)})$. The corresponding proposal distribution becomes:

$$T_{\text{MALA}}(\bar{u}^{(t)} \rightarrow \bar{v}) = \mathcal{N}\left(\bar{v}; \bar{u}^{(t)} + \frac{1}{2}\epsilon M(\bar{u}^{(t)}) \nabla_a \tilde{L}(\bar{u}^{(t)}), \epsilon M(\bar{u}^{(t)})\right). \hspace{1cm} (4.9)$$

As the notation suggests, the preconditioning matrix can vary with $\bar{u}$. We focus on this case, which we refer to as *adaptive preconditioning*, and to the corresponding algorithm as *MALA with adaptation*.

The preconditioning matrix should be selected to be representative of the correlations that exist between the dimensions of the sampling domain $\mathcal{U}$. In gradient-based optimization, this matrix is ideally set equal to the inverse of the Hessian matrix $H \equiv \left[\frac{\partial^2 L}{\partial u_i \partial u_j}\right]$, as in Newton’s method [166]. However, the Hessian matrix cannot be used for preconditioning in MALA, as it is not guaranteed to
be positive-definite. Approaches to circumvent this include using the expectation of the Hessian to form the Fisher information matrix [55], or applying transformations to the Hessian’s eigenvalues to produce a positive definite matrix [120, 16]. Unfortunately, these Hessian-based approaches introduce a significant computational overhead compared to standard LMC: first, they require computing second-order derivatives of $\tilde{L}$, instead of just first-order derivatives; second, they require performing expensive matrix operations (e.g., eigendecomposition of the Hessian). We note additionally that, even when an eigendecomposition is not required, it will be necessary to compute a factorization of the preconditioning matrix to sample the Gaussian proposal of Equation (4.9). As all of these computations need to be performed for every sampled path, they can become prohibitively expensive for MCMC rendering.

**Our contributions.** To alleviate these performance considerations, in this paper we will introduce three variants of MALA with adaptation that satisfy the following set of properties:

**P.1** They provide adaptation using a *diagonal* adaptive preconditioning matrix, as well as an *adaptive drift vector* that will not necessarily be parallel to the gradient $\nabla_0 \tilde{L}(\bar{u}^{(t)})$.

**P.2** Both of these adaptation mechanisms will require minimal processing of only previously-computed first-order gradients.

**P.3** The resulting MALA processes will remain ergodic, with the same stationary distribution as standard MALA.

The combination of properties **P.1** and **P.2** will ensure that our algorithms are efficient enough to facilitate MCMC rendering, avoiding second-order differentia-
tion and expensive matrix factorizations. As we will show, this can be achieved without compromising on the quality of the sampled paths. Our three algorithms will take the general form of MALA algorithm, with the computation of the preconditioning matrix and drift vector taking place in an online fashion (Section 4.4), a cache-driven fashion, or a hybrid of the two (Section 4.5). We note that MALA algorithm incorporates the large-step mutation of Equation (4.4), for which we will follow prior work (though see also Section 4.8). Additionally, the algorithm uses a modified version of the update of Equation (4.8), to emphasize the computational simplicity of diagonal preconditioning: $\otimes$, $\oslash$, and $\oslash^\frac{1}{2}$ are element-wise multiplication, division, and square root, respectively, $\mathbf{1}$ is a vector of all ones, and we overload notation to write $\mathbf{M}^{(t)}$ as a vector.

**Hamiltonian Monte Carlo.** We briefly discuss the relationship of LMC to another class of techniques known as Hamiltonian Monte Carlo (HMC) [42, 158, 17]. HMC uses Hamilton’s equations to evolve a continuous-time Markov process, corresponding to a dynamical system with potential energy determined by the target function $\tilde{L}$, and kinetic energy determined by a custom mass matrix. In practice, the continuous evolution equations are approximated using multiple steps of the leapfrog integrator. This results in a discrete-time Markov chain, which is combined with the Metropolis-Hastings rule. When only one integration step is used between proposals, this Markov chain becomes equivalent to MALA, with the mass matrix used for preconditioning in Equation (4.8) [158, 55]. The use of multiple steps can help reduce sample correlation, but is prohibitively expensive for MCMC rendering, as explained by Li et al. [120]. They instead derive a single-step procedure using a Gaussian approximation to the potential energy; as we discussed above, this is equivalent
to MALA with Hessian-based preconditioning.

4.4 Online Adaptation

Our goal in this section is to develop online adaptation mechanisms for MALA that continuously update the preconditioning matrix and drift vector as new samples are generated, while also satisfying properties P.1-P.3. For adaptation to be beneficial, the preconditioning matrix should adapt to the local geometry of the sampling domain, and the drift vector should accelerate convergence to local modes.

For inspiration on how to achieve these desiderata, we can utilize the similarity between the MALA transition distribution and gradient-based optimization. For example, we could draw an analogy with classical quasi-Newton optimization algorithms, which strike a balance between the fast convergence of the Hessian-based Newton’s method, and the computational efficiency of standard first-order gradient descent. Quasi-Newton algorithms achieve this by using, at each iteration, the accumulated history of first-order gradients, to approximate the Hessian matrix. Unfortunately, naively converting these algorithms to a MALA procedure (e.g., by adding noise to each iteration) will not be successful: First, most common Hessian approximations, such as the BFGS, DFP, and Broyden methods [166], are not guaranteed to be positive definite; enforcing positive-definitiveness requires expensive matrix operations [254, 215], which contradicts our desired property P.2. Second, even if positive definitiveness could be guaranteed, the resulting Markov chain would not necessarily be ergodic, as required by our desired property P.3.
Figure 4.2: 2D sampling example: We compare various MCMC sampling algorithms on a simple two-dimensional anisotropic geometry, typical of those encountered in the primary sample space near caustics. Starting at the same initial position (top of the curve), we show 2048 samples and acceptance ratios produced by each algorithm. “Mmt.” refers to momentum, and “diag.” and “full” to diagonal and full preconditioning, respectively.

We develop our online adaptation procedure by drawing ideas from two distinct areas: First, we take advantage of the similarity between the MALA transition distribution and gradient-based optimization. In particular, we propose to use the Hessian approximations and momentum terms of state-of-the-art stochastic gradient descent (SGD) algorithms [43, 105], as the preconditioning matrix and drift vector, respectively, of MALA with adaptation. Second, we look at the theory of controlled MCMC [7, 198] and use it to modify the online computation of the preconditioning matrix and drift vector in a way that ensures ergodicity. We refer to the resulting algorithm as MALA with online adaptation. As we develop our algorithm, we use Figure 4.2 as a two-dimensional visualization of the effect of each of its components on sampling performance.
Adam preconditioning. Current state-of-the-art SGD algorithms use an adaptive step-size matrix that was originally introduced in the AdaGrad algorithm [43], and was later modified in algorithms such as Adam [105]. Even though these algorithms are popular, e.g., for training deep learning pipelines, they have not previously been used for MCMC rendering. We will describe the Adam algorithm in the context of MALA, where the step-size matrix plays the same role as a preconditioning matrix.

Consider a set of samples \( \bar{u}^{(1)}, \ldots, \bar{u}^{(t)} \) produced by the Markov chain. For each of the samples, we denote by \( \tilde{g}^{(\tau)} \equiv \nabla_{\bar{u}} \tilde{L}(\bar{u}^{(\tau)}) \), \( \tau = 1, \ldots, t \), the gradient of the log-path contribution at that point. These gradients are already computed when generating the MALA proposals. Adam first forms a diagonal accumulation matrix as:

\[
G^{(t)}_O = \sum_{\tau=1}^{t} \beta^{\tau-1} (1 - \beta) \text{diag} \left( \tilde{g}^{(\tau)} \cdot \left( \tilde{g}^{(\tau)} \right)^\top \right),
\]

(4.10)

The constant \( \beta \in [0, 1] \) results in accumulation with exponentially-decaying weights, to emphasize more recent derivatives (likely to be closer to the current location \( \bar{u}^{(t)} \)), and downweight older ones. In practice, the accumulation is performed iteratively, without the need to store previous gradients. Adam uses \( G^{(t)}_O \) to approximate the Hessian and preconditioning matrix at \( \bar{u}^{(t)} \) as:

\[
H^{(t)}_O = \delta I + \sqrt{G^{(t)}_O},
\]

(4.11)

\[
M^{(t)}_O = \left( H^{(t)}_O \right)^{-1},
\]

(4.12)

where \( \delta > 0 \) is a small constant.

We make the following observations for Equations (4.10)-(4.12): First, the computation of the preconditioning matrix \( M^{(t)}_O \) only uses previously computed

\[\text{bias correction}\]

We note that Adam includes a bias correction term in the accumulation matrix. We omit this term, as we found empirically that it results in worse rendering performance.
Figure 4.3: **Ablation study:** We use the *door* scene to compare different versions of our proposed MALA with online adaptation. In (a-f), we show equal-sample (64 samples-per-pixel) renderings produced using the small-step perturbation of Kelemen et al. [101], and MALA with different combinations of momentum (“mmt.”) and diagonal (“diag.”) or full preconditioning. In (g-h), we show converged (20,000 samples-per-pixel) renderings produced using MALA with online adaptation, with and without diminishing adaptation (“DA”). All images include error maps with respect to a reference rendering in (i) produced using BDPT.

First-order gradients. Second, $M_t^{(0)}$ is guaranteed to be positive definite, as required for MALA with adaptation, without the need for additional eigendecomposition operations. Third, both $M_t^{(0)}$ and its factorization can be computed using simple *scalar* inversion and square root operations. From these, we conclude that the preconditioning matrix $M_t^{(0)}$ satisfies properties P.1-P.2. Finally, we refer to the AdaGrad and Adam papers [43, 105] for theoretical arguments justifying the Hessian approximation of Equations (4.10)-(4.12). All of these make $M_t^{(0)}$ a suitable preconditioning matrix for MALA with adaptation.
**Diagonal versus full preconditioning.** The computational advantages of diagonal adaptation come at the cost of potentially worse adaptation: the preconditioning matrix $M^{(t)}_O$ can no longer capture correlations that may exist between the dimensions of the sampling space $\mathcal{U}$. As discussed by Li et al. [120], accounting for these correlations can improve rendering performance for complex scenes.

To quantitatively assess the extent to which adaptation with diagonal preconditioning negatively affects sampling performance, we consider a full-variant of Adam preconditioning [43]. This requires modifying the accumulation matrix as:

$$G^{(t)}_O = \sum_{\tau=1}^t \beta^{t-1} (1 - \beta) \tilde{g}^{(\tau)} \cdot (\tilde{g}^{(\tau)})^T,$$

with Equations (4.11)-(4.12) remaining the same. The resulting matrix $M^{(t)}_O$ remains positive definite, and therefore can be used for preconditioning. Computing this full preconditioning matrix requires factorizing the matrix $G^{(t)}_O$, and inverting the matrix $H^{(t)}_O$. An additional factorization is required to compute $\sqrt{M^{(t)}_O}$ for sampling. We note that the computational overhead of these operations compared to the case of diagonal preconditioning can be alleviated by using efficient rank-two update algorithms [254, 25] to directly compute both $M^{(t)}_O$ and its factorization.

In Figure 4.3(e-f), we compare equal-sample renderings produced using diagonal and full preconditioning. We observed that full preconditioning provides a small improvement over diagonal preconditioning. This small improvement comes with a considerable computational overhead, resulting in more than double the runtime compared to diagonal preconditioning. In Section 4.7, we show additional experiments, including comparisons to the Hessian-based
preconditioning of Li et al. [120], supporting these observations. Therefore, we focus on the diagonal case in the rest of the paper.

**Adam momentum.** In addition to a Hessian approximation, Adam uses *momentum* [183, 220] to accelerate optimization convergence. We adopt the same as an adaptive drift vector in MALA. Concretely, we compute the momentum vector as:

\[
\mathbf{m}^{(t)}_O = \sum_{\tau=1}^{t} \alpha^{\tau-1} (1 - \alpha) \mathbf{g}^{(\tau)}. \tag{4.14}
\]

where \(\alpha \in [0, 1]\) controls the exponential decay of older gradients. As in Equation (4.10), in practice gradients are accumulated iteratively. In the next section, we describe how we modify MALA to use this momentum vector and the preconditioning matrix of Equation (4.12).

### 4.4.1 Ensuring ergodicity

A naive way to combine the Adam preconditioning and momentum schemes with MALA would be to modify Equation (4.8) as:

\[
\hat{\mathbf{u}}^{(t+1)} = \hat{\mathbf{u}}^{(t)} + \frac{1}{2} \varepsilon \mathbf{M}^{(t)}_O \mathbf{m}^{(t)}_O + \sqrt{\varepsilon} \sqrt{\mathbf{M}^{(t)}_O} \hat{\mathbf{w}}. \tag{4.15}
\]

Unfortunately, using this equation to generate samples is likely to result in incorrect estimates, as the resulting stochastic process is not guaranteed to be ergodic, even when combined with Metropolis-Hastings. This is because the proposal distribution implied by Equation (4.15) is no longer *time-homogeneous*: Due to the use of the preconditioning matrix \(\mathbf{M}^{(t)}_O\) and momentum vector \(\mathbf{m}^{(t)}_O\), the proposal will be different each time the process returns to a specific state \(\hat{\mathbf{u}}\).
(Equivalently, the proposal is a function of not just \( \bar{u} \), but also the time \( t \).) Consequently, the conditions for ergodicity we discussed in Section 4.2 are no longer sufficient. To address this problem, we first present some technical background on controlled MCMC techniques, then use this theory to derive an ergodic version of Equation (4.15).

**Controlled MCMC.** We provide a brief review, referring to Andrieu and Thoms [7] for more details, and to Hachisuka and Jensen [65] for a previous application of the controlled MCMC framework to rendering in the context of photon mapping.\(^3\)

Controlled MCMC algorithms consider, in addition to the sampling domain \( \mathcal{U} \), a parameter domain \( \Theta \). Parameter vectors \( \theta \in \Theta \) control the Markov chain’s transition distribution \( P : \mathcal{U} \times \mathcal{U} \times \Theta \to \mathbb{R}_+ \). Each parameter vector effectively maps to a different transition distribution \( P(\bar{u} \rightarrow \cdot ; \theta) \), which can be used to sample new states for the chain. The parameter vectors to be used are determined using a time-varying control function \( F \) that maps tuples of states to parameters. At time \( t + 1 \), given the history of samples \( \bar{u}^{(1)}, \ldots, \bar{u}^{(t)} \) and an initial parameter vector \( \theta^{(0)} \), sampling proceeds as:

\[
\begin{align*}
\text{Sample } & \bar{u}^{(t+1)} \sim P(\bar{u} \rightarrow \cdot ; \theta^{(t)}). \\
\text{Update } & \theta^{(t+1)} = F(t+1, \theta^{(0)}, \bar{u}^{(1)}, \ldots, \bar{u}^{(t+1)}).
\end{align*}
\]

In our derivation, we will use a set of three conditions that are sufficient for creating an ergodic controlled MCMC process, which we simplify and summarize as follows.

\(^3\)Controlled MCMC is often also referred to as adaptive MCMC. We do not use this term, to avoid confusion with MALA with adaptation.
**C.1 Simultaneous ergodicity:** For all parameter vectors $\theta \in \Theta$, the transition distribution $P(\bar{u} \to \cdot; \theta)$ must be ergodic.

**C.2 Bounded convergence:** The space $\mathcal{U} \times \Theta$ is compact.

**C.3 Diminishing adaptation:** The time-varying transition distribution $P(\bar{u} \to \cdot; \theta(t))$ converges to some fixed distribution $P(\bar{u} \to \cdot; \theta^*)$ as $t \to \infty$.

These conditions were derived by Roberts and Rosenthal [197], and we refer to their paper for their technical formulation. Whereas conditions C.1 and C.2 are typically trivially satisfied, the diminishing adaptation condition C.3 requires careful design of the controlled MCMC process. Effectively, this condition ensures that, as $t \to \infty$, the proposal distribution becomes time-homogeneous.

**Enforcing diminishing adaptation.** We now revisit Equation (4.15), which we can interpret as a controlled MCMC process: its parameter vector is the tuple $\theta(i) = (m(i)_O, M(i)_O)$, and the control function $F$ corresponds to Equations (4.14) and (4.10)-(4.12). Finally, the transition distribution $P$ is the combination of a modified Equation (4.9):

$$T_{\text{online}}(\bar{u} \to \bar{v}; \theta(i)) = \mathcal{N}\left(\bar{v}; \bar{u} + \frac{1}{2} \epsilon M_O^{(i)} m_O^{(i)}, \epsilon M_O^{(i)}\right),$$

with the Metropolis-Hastings acceptance rule of Equation (4.3).

This controlled MCMC process satisfies the simultaneous ergodicity condition C.1: For each $\theta$, the proposal distribution of Equation (4.18) is equivalent to MALA with a fixed preconditioning matrix and mean shift that, when combined with Metropolis-Hastings, is ergodic. It also satisfies the bounded convergence condition C.2, because $\mathcal{U}$ is bounded and closed, and $\Theta$ is in practice finite (floating point numbers have finite state space [65]).
However, this controlled MCMC process does not satisfy the diminishing adaptation condition C.3: As $t \to \infty$, $M_O^{(t)}$ and $m_O^{(t)}$ do not converge to a limit that is either fixed or a function of only $\bar{u}^{(t)}$. To address this, we propose to modify Equations (4.11) and (4.14) as:

$$H_O^{(t)} = \delta I + \frac{1}{c_1} \sqrt{G_O^{(t)}}, \quad (4.19)$$

$$m_O^{(t)} = \bar{g}^{(t)} + \frac{1}{c_2} \sum_{\tau=1}^{t} \alpha^{\tau-1} (1 - \alpha) \bar{g}^{(\tau)}, \quad (4.20)$$

where $c_1, c_2 > 0$ are positive exponents. With this modification, as $t \to \infty$, $M_O^{(t)}$ converges to $\delta I$, and $m_O^{(t)}$ converges to the gradient $\bar{g}^{(t)}$. Therefore, the modified controlled MCMC process satisfies the diminishing adaptation condition C.3, and is thus ergodic. We confirm this and demonstrate the need for diminishing adaptation in Figure 4.3(g-i): when rendering converged images with and without diminishing adaptation, the former matches the reference (rendered with BDPT), whereas the latter results in strong artifacts. We term this modified controlled MCMC process MALA with online adaptation.

Discussion. In Figures 4.3(a-e), we show equal-sample comparisons of different versions of our algorithm against two baselines, the small-step perturbation of Kelemen et al. [101], and standard MALA without adaptation. We observe that both of the adaptation mechanisms we introduced, adaptive drift and (especially) preconditioning, result in significantly improved rendering performance. Their combined use in MALA with online adaptation reduces error by almost 80% compared to standard MALA. This improvement comes at a negligible computational overhead—the runtimes for Figures 4.3(b) and (e) differ by approximately 8%. Additionally, in Section 4.7, we perform equal-time comparisons of our algorithm with state-of-the-art MCMC rendering algorithms (in-
cluding a version of MALA with Hessian-based preconditioning [120]). Despite its simplicity, our algorithm results in $2 - 4 \times$ error reduction across a variety of complex scenes.

At the same time, we can identify two shortcomings of MALA with online adaptation. First, the diminishing adaptation scheme of Equations (4.19)-(4.20) has the effect that, as $t \to \infty$, our algorithm converges to standard MALA without preconditioning or momentum (Equation (4.7)). This can be problematic for complicated scenes where rendering low-noise images requires a very large number of samples, as later samples will not benefit from preconditioning.

Second, as discussed in Section 4.2, in practice it is necessary to combine the MALA-based small-step perturbation with a large-step mutation. When a large step is performed and the Markov chain moves to a new neighborhood in $U$, the accumulated preconditioning matrix $M_O^{(t)}$ and momentum vector $m_O^{(t)}$ will not be good approximations of the local geometry until a few more small steps are performed within this neighborhood. As a result, we expect that our adaptive preconditioning will not be effective immediately after the occurrence of large steps, resulting in many rejected samples. Even though we can mitigate this effect by reducing the probability $\pi_{\text{global}}$ of a large step, this would result in slower global exploration. In the next section, we build upon the techniques we developed in this section to propose a version of MALA with adaptation that overcomes both of the shortcomings discussed above, without hindering global exploration.
4.5 Cache-Driven and Hybrid Adaptation

In this section, we show how to modify MALA with online adaptation to achieve better asymptotic performance as $t \to \infty$ and reduce disruption by large steps, while still producing an algorithm that satisfies properties P1-P3. Our main insight is as follows: introducing diminishing adaptation in the previous section was necessary because Equations (4.10) and (4.14) accumulate gradients using exponentially-decaying weights. This accumulation scheme serves as a computationally efficient proxy for using only past gradients that were computed in the proximity of the current sample. If we can afford to store and query all previously-computed gradients, then we can modify these rules to only use nearby previous gradients, without the need for exponentially-decaying accumulation.

We formalize this insight by making two contributions: First, we combine a new gradient caching scheme with the adaptation rules of the previous section to propose a new algorithm that we call MALA with cache-driven adaptation, and which remains ergodic without the need for diminishing adaptation. Second, we develop a hybrid between online and cache-driven adaptation, that combines their complementary advantages. The resulting hybrid algorithm will be the main algorithm we use in our rendering experiments.

**Gradient caching.** At any given time $t$, we denote by

$$C^{(t)} \equiv \left\{ \left( \bar{\mathbf{u}}^{(\tau)}, \mathbf{g}^{(\tau)} \right), \tau = 1, \ldots, t \right\}, \quad (4.21)$$

the set of tuples of sequences $\bar{\mathbf{u}}^{(\tau)}$ and corresponding gradient values $\mathbf{g}^{(\tau)}$ that the Markov chain has visited. We refer to this set as a gradient cache, hinting at its
eventual implementation in Section 4.6.

Given a sequence \( \tilde{v} \) and a radius \( r > 0 \), we denote by

\[
Q\left( \tilde{v}; C^{(t)}, r \right) \equiv \left\{ \tau : (\tilde{u}^{(\tau)}, g^{(\tau)}) \in C^{(t)} \text{ and } \| \tilde{u}^{(\tau)} - \tilde{v} \| < r \right\},
\]

the set of integer indices for tuples in \( C^{(t)} \) whose sequence \( \tilde{u}^{(\tau)} \) is within Euclidean distance \( r \) of \( \tilde{v} \). Continuing the cache analogy, we refer to this set as the query of cache \( C^{(t)} \) for point \( \tilde{v} \).

**Cache-driven preconditioning and momentum.** We can now update the definitions of preconditioning and momentum in Section 4.4 so that they use gradient caching. In particular, we define diagonal preconditioning with gradient caching as:

\[
G^{(t)}_C = \text{diag} \left( \frac{1}{\left| Q\left( \tilde{v}; C^{(t)}, r \right) \right| \sum_{\tau \in Q\left( \tilde{u}^{(t)}; C^{(t)}, r \right)} g^{(\tau)} \cdot (g^{(\tau)})^\top} \right),
\]

\[
H^{(t)}_C = \delta I + \sqrt{G^{(t)}_C},
\]

\[
M^{(t)}_C = \left( H^{(t)}_C \right)^{-1},
\]

where \( \left| Q\left( \tilde{v}; C^{(t)}, r \right) \right| \) is the size of the set \( Q\left( \tilde{v}; C^{(t)}, r \right) \). Similarly, we define momentum with gradient caching as:

\[
m^{(t)}_C = \frac{1}{\left| Q\left( \tilde{v}; C^{(t)}, r \right) \right| \sum_{\tau \in Q\left( \tilde{u}^{(t)}; C^{(t)}, r \right)} g^{(\tau)}}.
\]

Finally, with these definitions at hand, we modify the MALA update rule of Equation (4.8) to take the form:

\[
\tilde{u}^{(t+1)} = \tilde{u}^{(t)} + \frac{1}{2} \varepsilon M^{(t)}_C m^{(t)}_C + \sqrt{\varepsilon} \sqrt{M^{(t)}_C} \tilde{w},
\]
Equivalently, given a cache $C(t)$, we can define a controlled variant of the MALA proposal distribution of Equation (4.9) as:

$$T_{\text{cache}}(\bar{u} \rightarrow \bar{v}; C(t)) = \mathcal{N}\left(\bar{v}; \bar{u} + \frac{1}{2} \epsilon M_{C}^{(i)} m_{C}^{(i)}, \epsilon M_{C}^{(i)}\right).$$  \hspace{1cm} (4.28)

The combination of this proposal with the Metropolis-Hastings acceptance rule is our proposed MALA with cache-driven adaptation.

**Ergodicity.** Analogously to Section 4.4.1, MALA with cache-driven adaptation is a controlled MCMC procedure: the parameter vector is the gradient cache, $\theta^{(i)} \equiv C^{(i)}$, and the control function $T$ corresponds to Equations (4.23)-(4.26). In Section 4.4.1, we had to modify the control equations for online preconditioning and momentum, to ensure that the diminishing adaptation condition $C.3$ is satisfied. By contrast, this is not necessary for the cache-driven variants we defined above, as ergodicity can be achieved in an alternative way.

In particular, we can define some threshold $H$ on the maximum number of entries $|C^{(i)}|$ we will allow the cache to store. Once this threshold is reached, we stop caching, while continuing to run MALA with the preconditioning matrix and momentum of Equations (4.23)-(4.26). Consequently, the time-varying transition distribution of Equation (4.28) reaches a fixed distribution when the threshold $H$ is reached, and therefore satisfies the diminishing adaptation condition $C.3$. We conclude that MALA with cache-driven adaptation is ergodic with the correct stationary distribution. This mechanism for ensuring ergodicity is known as *finite adaptation* [197], and remains effective for any finite value of $H$. 

44
Figure 4.4: 2D Gaussian mixture model: We compare the performance of MALA with online and hybrid adaptation, on a two-dimensional disconnected geometry, typical of the isolated “islands” of primary sample space. Both algorithms use full preconditioning and large-step mutation probability $T_{\text{global}} = 0.1$. Each row shows sample distributions when running the algorithm for (from left to right) 0.5K, 1K, 2K, 4K, 8K and 16K samples. In the bottom row, the insets show the evolution of the anisotropic transition kernels at two different points in space. Please zoom in to better compare the results.

Comparing online and cache-driven adaptation. We have now defined two versions of MALA with adaptation, one in Section 4.4 using an online procedure inspired from SGD, and one in this section driven by a gradient cache. The cache-driven version overcomes the two limitations of the online version discussed at the end of Section 4.4: in particular, given that it does not require introducing explicit diminishing adaptation, the cache-driven version converges to MALA with a non-unit preconditioning matrix—approximately the one that would result from an MCMC process using the online scheme without large-step mutations or exponentially-decaying summation. Additionally, the cache-driven version adapts to the occurrence of large-step mutations seamlessly, as Equations (4.23)-(4.26) allow it to immediately switch to computing preconditioning matrices and momentum vectors using any available gradients in the new neighborhood it moves to. We show a two-dimensional visualization of this improved behavior in Figure 4.4.
Conversely, these advantages come at the cost of maintaining the gradient cache $C(t)$, and performing potentially expensive cache queries $Q(\cdot; C(t), r)$ for each sampled path. This cost increases as rendering progresses and the cache becomes larger, making the addition of new entries and queries more expensive. Another downside of the cache-driven scheme is that at the initial stages of the rendering process, when the cache is sparsely populated, it may result in no preconditioning even in cases where the online scheme would produce a non-identity preconditioning matrix. This can happen, for example, after a sequence of small-step perturbations that are all greater than the radius $r$. These often occur immediately after a large step mutation, when the Markov chain lands at a new neighborhood and starts quickly converging towards the local mode: Whereas the online scheme would use the resulting sequence of gradients to form a preconditioning matrix, the cache-driven scheme would reject them when querying for nearby gradients.

Finally, the effectiveness of cache-driven adaptation will decrease as the path length $B$, and thus dimensionality $r(B)$ of the cached gradients, increases. This is because of the *curse of dimensionality*, which states that the number of samples that need to be cached for nearest-neighbor estimation to be accurate increases exponentially with dimensionality. As we discuss in Section 4.6, in practice we circumvent this issue by disabling adaptation for very long paths, as is commonly done in many MCMC rendering techniques [152, 260, 190].

**Hybrid adaptation.** We propose a hybrid between the two adaptation schemes that combines their complementary advantages. Our hybrid operates in two stages. The first stage continues while the number of entries in the cache, $|C(t)|$, is smaller than some predefined threshold $H$. During this stage, we use
online adaptation, but without diminishing adaptation \((c_1, c_2 = 0)\), as ergodicity will be guaranteed when we switch to the second stage. Additionally, we continue to expand the cache with any new gradient evaluation.

The second stage starts once the number of entries in the case reaches the threshold, \(|C^{[0]}| \geq H\). During this stage, we disable online adaptation and use exclusively the cache-driven version. We additionally stop adding new entries to the cache, which ensures ergodicity and helps reduce the cost of subsequent cache queries.

### 4.6 Implementation Details

We implement our proposed algorithms within the open-source reference implementation [117] of H2MC [120]. This is a primary-sample-space MCMC renderer that, critically, uses automatic differentiation [15] to compute derivatives of the measurement contribution function. We modify the automatic differentiation code to disable the computation of Hessian matrices, which are needed by H2MC but not by our algorithms.

**Perturbation and mutation strategies.** We use the same large-step mutation and small-step perturbation strategies as H2MC, including their proposed parameterization change and mixing weights. We only apply MALA-based sampling to the small-step mutation, as suggested by H2MC. We also follow H2MC and disable MALA with adaptation for paths of length \(B \geq 8\) (primary-sample-space sequences of length \(r(B) \geq 16\)), reverting to Gaussian mutations [101] for longer paths. We note that we did not implement lens perturbations in our al-
algorithm, which we also disabled when comparing with H2MC. This lets us perform fair comparisons with other primary-sample-space MLT methods [21, 66] that do not support lens perturbations.

**Gradient caching.** We implement the gradient cache with the nanoflann library [22], which uses KD-tree structures to support efficient nearest-neighbor searches even in high-dimensional spaces (such as the primary sample space). To further reduce the overhead of cache queries, we use a hybrid of nearest-neighbor and radius queries: We terminate the KD-tree traversal once there are $K$ neighbors found within the search radius $r$.

**Truncated gradients.** Livingstone and Zanella [129] recently analyzed the convergence behavior of gradient-based MCMC (e.g., Langevin and Hamiltonian Monte Carlo), and showed that very large variations in the magnitude of the gradients can significantly deteriorate convergence. This observation is also supported by older literature, which suggests truncating gradients when using either standard MALA [200] or MALA with adaptation [9]. We follow this suggestion and truncate all dimensions of our gradients to a maximum value of $g = 100$, which helps ensure robust convergence across a large variety of scenes.

**Parameter settings.** The proposed algorithms use a few parameters, namely: exponential-decay rates ($\alpha, \beta$), diminishing adaptation exponents ($c_1, c_2$), cache query radius ($r$), nearest neighbors ($K$), cache size ($H$), uniform preconditioning weight ($\delta$), and step size ($\epsilon$). We set $\alpha = 0.9$, $\beta = 0.999$, and $\delta = 0.001$, which are the values recommended by Kingma and Ba [105]. We also select values $\epsilon = 0.01$, $r = 0.01$, $K = 30$, and $H = 10,000$ for the remaining parameters. We use
these parameter values for all the renderings shown in this project.

### 4.7 Experiments

We run all experiments on a machine with an Intel Xeon E5-2630 v3 processor. We render all reference images by running BDPT for more than 12 hours per scene.

**Equal-time comparisons.** We perform equal-time comparisons of the online and hybrid versions of MALA with adaptation, with BDPT and five state-of-the-art MCMC rendering algorithms: the path-space MEMLT algorithm [84], and the primary-sample-space RJMLT [21], MMLT [66], and H2MC [120] algorithms. We use a set of 17 scenes with complex illumination, occlusions, caustics, and interreflections. These characteristics make these scenes challenging for non-MCMC rendering algorithms such as BDPT. To verify the consistency of all the MCMC algorithms we test, we confirmed that in all scenes they match the reference when run until convergence. On all scenes, MALA with hybrid adaptation was the best performing algorithm, and MALA with online adaptation performed second best. Relative performance among the other MCMC algorithms varied from scene to scene. These results suggest that, despite its simplicity, MALA with online adaptation can be expected to have robust and efficient performance across a large variety of scenes. Additional performance gains can be obtained by using MALA with hybrid adaptation, which however has higher implementation complexity. Overall, our two algorithms resulted in an average MSE improvement of 3× compared to the third best. Finally,
Figure 4.5: Equal-time comparisons: We compare MEMLT [84], MMLT [66], RJMLT [21], H2MC [120] and two of our algorithms, across several scenes with complex illumination and occlusion, glossy caustics and interreflections.
Figure 4.6: **Equal-time comparisons:** We compare MEMLT [84], MMLT [66], RJMLT [21], H2MC [120] and two of our algorithms, across several scenes with complex illumination and occlusion, glossy caustics and interreflections.

BDPT had the worst average performance, suggesting that the complexity of our scenes makes it necessary to use MCMC rendering for good performance.
Figures 4.1, 5.5 and 4.6 show a representative sample of these renderings for all the MCMC algorithms; we provide the complete set of comparisons, including comparisons to BDPT, in the supplement.

**Motion blur simulations.** We follow Li et al. [120] and simulate motion blur by treating time as another dimension in the primary sample space. We then perform an equal-time comparison of our algorithms with H2MC, on two scenes that include challenging motion blur effects, as shown in Figure 4.7 (we provide full-sized renderings and comparisons in the supplement): the *cars* scene contains a static and a moving car, and the *necklace* scene shows a gemstone rotating and a golden ring moving linearly. In addition to motion blur, both
scenes include other difficult light transport effects, such as specular-diffuse-specular (SDS) paths and complex caustics. Our algorithms significantly outperformed H2MC in both scenes, resulting in $7.9\times$ and $7.4\times$ MSE improvements. This demonstrates the ability of our algorithms to adapt to each dimension of the primary sample space, including time. We note that we did not compare against the other MCMC rendering algorithms we considered in this section: RJMLT does not provide an implementation that supports motion blur, and Li et al. [120] demonstrated that H2MC outperforms MMLT and MEMLT in scenes with motion blur effects.

4.8 Conclusion and Discussion

We discuss some important aspects of our algorithms, which suggest limitations and directions for future exploration.

**Differentiation in rendering.** Compared to a standard primary-sample-space MCMC renderer, the main additional requirement of our adaptation algorithms is the ability to compute first-order gradients of path tracing algorithms. This differentiation functionality is becoming standard in modern rendering engines [6, 120, 118, 253, 164, 28], which incorporate automatic differentiation [61] to facilitate both forward and inverse rendering. There is additionally active research [83] on designing automatic differentiation libraries that are better suited to the computational graphs typical of rendering, compared to general-purpose libraries such as the one in our implementation [15]. These developments suggest that our online adaptation algorithm can be readily incorporated within
modern rendering software, and potentially offer even bigger performance improvements over gradient-free algorithms than those reported in Section 4.7.

**Gradient-based sampling in path space.** Our algorithms are currently designed for use only in the primary sample space. This is in large part due to the mathematical convenience this space provides, making it amenable to differentiation. Considering the complementary advantages of sampling techniques operating in the two spaces, it would be interesting to explore the use of MALA with adaptation in path space, where gradients have already found some use for sampling near-specular paths [84, 94]. As discussed above, this exploration can benefit from the availability of end-to-end differentiable rendering systems.

**Global exploration.** We focused on developing small-step perturbation proposals for local exploration, and relied on prior work [101] for the large-step mutation proposals needed for global exploration. As both critically affect rendering performance, in the future it will be important to research global exploration techniques that synergize with the local exploration techniques we developed. One direction is to use the cache of our hybrid method to additionally facilitate global exploration. A straightforward way to do this would be to modify the cache to store both gradients and throughput values $f(\bar{u})$. We could use this information to replace the uniform large-step mutation proposal $T_{\text{global}}$ with one that uses the cache for importance sampling, for example, by forming a kernel density estimate of $f(\bar{u})$. This form of cache-driven adaptation for global exploration can potentially improve rendering performance, at a negligible overhead (the cache is already used for local exploration) and without affecting ergodicity (the same arguments for the ergodicity of cache-driven local exploration apply).
We leave a more detailed investigation, including utilizing learning-based techniques for path guiding in the primary sample space [152, 260, 190], for the future.

**Parameter setting.** As discussed in Section 4.6, our algorithms requires setting a large number of parameters. Empirically, we observed that some of these parameters (for example, the step size $\epsilon$) can result in significant deterioration in performance if set incorrectly. However, we additionally found that selecting parameter values can be done once, without the need for per-scene fine-tuning: We coarsely searched for parameter values that produced good performance on a simple scene (ring scene in the supplement), and then used the same values for all of our experiments. This suggests that users of our implementation should be able to achieve robust high performance, without the need to change the default parameter values we provide. In the supplement, we empirically investigate how changing different parameters impacts rendering performance. An interesting future research direction would be to use controlled MCMC for online adaptation (during the rendering process) of parameters towards their scene-specific optimal values.

We introduced two variants of the Metropolis-adjusted Langevin algorithm with adaptation that are suitable for MCMC rendering. For efficiency, our algorithms mimic SGD and employ diagonal preconditioning and momentum, computed using only past gradients in an online or cache-driven manner. To ensure correctness, our algorithms are designed using the framework of controlled MCMC. Put together, these features allow our algorithms to adapt to the geometry of the primary sample space for local exploration, and to seamlessly combine with complementary sampling techniques for global exploration. We
have demonstrated that our algorithms outperform the state-of-the-art in equal-
time comparisons, resulting in an average MSE improvement of $3\times$ on a variety of scenes, and $7\times$ on scenes with motion blur.
CHAPTER 5
LEARNING-BASED INVERSE SUBSURFACE SCATTERING

In this chapter, we present our learning-based pipeline for inverse subsurface scattering.

Given images of translucent objects, of unknown shape and lighting, we aim to use learning to infer the optical parameters controlling subsurface scattering of light inside the objects. We introduce a new architecture, the inverse transport network (ITN), that aims to improve generalization of an encoder network to unseen scenes, by connecting it with a physically-accurate, differentiable Monte Carlo renderer capable of estimating image derivatives with respect to scattering material parameters. During training, this combination forces the encoder network to predict parameters that not only match groundtruth values, but also reproduce input images. During testing, the encoder network is used alone, without the renderer, to predict material parameters from a single input image. Drawing insights from the physics of radiative transfer, we additionally use material parameterizations that help reduce estimation errors due to ambiguities in the scattering parameter space. Finally, we augment the training loss with pixel-wise weight maps that emphasize the parts of the image most informative about the underlying scattering parameters. We demonstrate that this combination allows neural networks to generalize to scenes with completely unseen geometries and illuminations better than traditional networks, with 38.06% reduced parameter error on average. This work has been published at ICCP 2020 [29].
5.1 Introduction

Translucent materials are everywhere around us, ranging from biological tissues to many industrial chemicals, and from the atmosphere and clouds to minerals. The common cause of the characteristic appearance of all these classes of materials is \textit{subsurface scattering}: As photons reach the surface of a translucent object, they continue traveling in its interior, where they scatter, potentially multiple times, before reemerging outside the object.

The ubiquity of translucency has motivated decades of research across numerous scientific fields on problems relating to subsurface scattering. Broadly speaking, we can break these problems down into two categories. The first category is forward scattering problems, which attempt to predict the appearance of a translucent object, assuming that the optical parameters controlling scattering of light at its interior are known. Computer vision and computer graphics offer an array of algorithms for solving this problem, known in this literature as \textit{volume rendering}, including Monte Carlo rendering algorithms that can reproduce translucent appearance in a physically-accurate way [167].

The second category, and the focus of this work, is inverse scattering problems: Given images of a translucent object, they attempt to predict its underlying scattering parameters. Inverse scattering is an active research topic in many sciences outside of computer vision and computer graphics, including medical imaging, remote sensing, and material science. The fundamental challenge in inverse scattering is the extremely multi-path and multi-bounce nature of light propagation inside scattering volumes. The complexity of volume light transport makes inverse scattering a difficult problem even in the case where an ob-
ject is characterized by a single set of, spatially-constant, material parameters (homogeneous scattering).

Among existing approaches for inverse scattering, many are based on simplifying assumptions about volume light transport, such as single scattering (all photons scatter once) and diffusion (all photons scatter a very large number of times). These assumptions limit the applicability of these methods to very optically-thin and thick materials [156, 86], excluding large classes of important turbid materials. Alternatively, recent years have seen the development of general-purpose inverse scattering techniques, which combine analysis by synthesis and Monte Carlo volume rendering in order to accurately estimate material parameters without the need for simplifications [60, 259, 103, 56, 114]. Despite their broad applicability, these techniques can be prohibitively computationally expensive: processing measurements of a new material often requires performing hundreds of expensive Monte Carlo rendering operations.

We investigate the use of deep learning techniques for inverse scattering problems, as a means to address the computational challenges of analysis by synthesis, while maintaining its broad applicability. We are inspired by recent successes of such techniques in other inverse rendering problems [241], such as inferring shape, reflectance, and illumination from images [143, 212, 126, 97]. Despite these successes, the use of neural networks for inverse scattering remains unexplored, and we take first steps in this direction.

We begin by proposing a physics-aware learning pipeline that we term inverse transport networks (ITN), which aims to combine the computational efficiency of learning-based approaches with the generality of analysis by synthesis approaches for inverse scattering. Taking inspiration from recent work on
combining physics and learning [143, 212, 126, 224, 97], these neural networks are trained to produce output parameters that not only match groundtruth values, but also reproduce the input images when used as input to a forward physics-based renderer. To be able to train these neural networks efficiently, we pair them with a new efficient and physically-accurate Monte Carlo differentiable rendering engine [259, 103, 56, 119]. We further tailor these neural networks towards inverse scattering, by taking into account results from the radiative transfer literature, characterizing the conditions under which different scattering materials can produce similar translucent appearance [246, 255]. We introduce ways for making our networks robust to these ambiguities, including the use of non-linear material parameterizations, and weight maps emphasizing pixels where these ambiguities are weaker. We demonstrate the effectiveness of our networks in experiments on synthetic and real datasets, where we show that our networks can use a single uncalibrated (completely unknown shape and illumination) image input, to produce material parameter estimates that are on average 38.06% more accurate than those produced by baseline regression networks. Furthermore, images rendered with our predictions are on average 53.82% closer to the groundtruth.

5.2 Overview

Physics-based techniques infer unknowns by optimizing for parameter values until they can be used, together with a physics-based image formation model, to reproduce input images. Although this optimization can sometimes be performed analytically (e.g., photometric stereo [78] and direct reflectometry [138]), it typically requires searching the parameter space and performing
multiple evaluations of the forward image formation model, until a set of parameters that can predict the input images is found. This framework is usually termed *analysis-by-synthesis* in computer vision or *inverse rendering* in graphics [132, 60, 259, 103, 56, 114]. By accurately modeling the underlying physics of light transport, physics-based techniques can produce high-fidelity estimates for fully general inputs, at the cost of high, and often prohibitive, computational requirements.

Learning-based techniques, in contrast, use supervised and unsupervised data to create functions that approximately map image measurements directly to physical parameters. In the last few years, these functions predominantly take the form of multi-layer neural networks, which can be efficiently optimized on large training datasets using stochastic optimization [121, 192, 221, 261, 157, 238, 115]. Although these techniques allow for efficient inference, they generally do not guarantee the physical plausibility of predicted parameter values. Additionally, since these methods are purely data-driven and do not attempt to reproduce the underlying physics, they usually generalize poorly to inputs that are underrepresented in the training dataset.

### 5.2.1 Problem Setting

We are interested in the problem of *homogeneous inverse scattering*: Given an image of a translucent object, of potentially unknown shape and lighting, we aim to determine the optical material parameters that control the scattering of light inside this object. These parameters are:

- The *extinction coefficient* $\sigma$, is the scalar optical density of the material, con-
trolling the average distance between consecutive volume events.

- The **volumetric albedo** $\alpha$ is the scalar probability of whether photons are scattered or absorbed at volume events.

- The **phase function** $f_p$ is the spherical probability distribution controlling the direction scattered photons continue to travel towards.

The phase function is typically assumed to be only a function of the inner product between incoming and outgoing directions. The first moment of the phase function, or **average cosine**, $\bar{c} = 2\pi \int_{-1}^{1} c f_p(c) \, dc$, $-1 \leq \bar{c} \leq 1$, is commonly used to characterize a material as predominantly forward-scattering ($\bar{c} > 0$), backward-scattering ($\bar{c} < 0$), or isotropic ($\bar{c} = 0$). From the above parameters, we can also derive the **scattering coefficient** $\sigma_s = \alpha \cdot \sigma_t$ and **absorption coefficient** $\sigma_a = (1 - \alpha) \cdot \sigma_t$, which describe how much light is scattered and absorbed, respectively, at each scattering event. In general, all these parameters can be spatially varying, but in our setting we assume they are constant everywhere inside the object (homogeneous scattering). Throughout the chapter, we will be using different subsets of the above parameters, as well as certain non-linear functionals, to characterize scattering. We will be denoting each material as $\pi$, with the corresponding parameterization inferred from context. We discuss specific parameterizations in Section 5.4.

The primary difficulty of the inverse scattering problem lies in the complexity of the underlying **volumetric light transport** physics: Each photon propagating inside a scattering medium undergoes a random walk, controlled non-linearly by the medium’s parameters. These random walks, described by the radiative transfer equation [148] typically involve more than one bounce. In turn, a radiometric detector capturing an image of such an object accumulates a large
number of photons, each performing a different random walk. As a consequence of this extremely multi-path and multi-bounce light transport, images of translucent objects are highly non-linear functions of the underlying material parameters. We will represent this complex image formation process using the operator $T(\pi)$, where $\pi$ are the material parameters. We note that $T$ is also a function of other scene parameters, such as shape and illumination; we omit this dependence for notational simplicity, and to focus on the material parameters we are interested in recovering.

In certain cases, we can simplify this image formation model by assuming that each photon only bounces either once or a very large number of times inside the object. These approximations, known as single scattering and diffusion respectively, are of limited applicability, as they are only accurate for very optically thin [156] or thick [86] materials. Additionally, the diffusion approximation cannot be used near thin geometric features such as sharp edges.

5.2.2 Analysis by Synthesis

The shortcomings of these approximations have motivated the development of general-purpose inverse scattering techniques that accurately model the full complexity of volumetric light transport [60, 259, 103, 56, 114]. These techniques operate within the framework of analysis by synthesis, also known in computer graphics as inverse rendering. Given image measurements $I$, we search for parameters $\pi$ that, when used to synthesize images, can closely match the measurements. This approach can be succinctly written as the following optimiza-
tion problem:
\[ \hat{\pi} = \text{argmin}_\pi \| I - T(\pi) \|_2. \]  
(5.1)

This procedure can be used for inverse scattering in objects of arbitrary known shape and lighting. This is thanks to the advent of graphics algorithms that can accurately simulate the full complexity of volumetric light transport. Besides traditional forward rendering algorithms that synthesize images as functions of material parameters \( \pi \) [167], recent years have seen the development of differentiable rendering algorithms that compute image derivatives with respect to these parameters, \( \partial T(\pi) / \partial \pi \) [60, 259, 103, 56, 119]. Differentiable rendering algorithms can greatly accelerate analysis by synthesis, by enabling the use of gradient descent algorithms for solving the optimization problem (5.1).

Despite these advances, performing inverse scattering by analysis by synthesis remains challenging in many situations. First, solving optimization (5.1), even with gradient descent, is computationally intensive, requiring performing hundreds or thousands of expensive rendering operations. Second, the use of gradient descent means that the analysis by synthesis optimization is susceptible to local minima in the loss function of Equation (5.1). This issue is particularly pronounced in inverse scattering, where the highly-nonlinear function \( T(\pi) \) results in large classes of different material parameters \( \pi \) that can produce similar images \( I \). These scattering parameter ambiguities are known as similarity relations [246, 255]. Third and last, performing inverse scattering requires accurate calibration of ancillary scene parameters such as shape, illumination, and camera pose, which is not possible except in controlled lab environments.

We aim to overcome these challenges by investigating the use of data-driven algorithms for the inverse scattering problem. In particular, in Section 5.3, we
discuss how to alter the training procedure of neural networks, to produce networks that, at test time, can use a single uncalibrated input image to produce material estimates $\pi$ that are close to those we would obtain from analysis by synthesis. Then, in Section 5.4, we discuss design choices, inspired from the physics of scattering, that help these networks overcome ambiguities due to similarity relations.

5.3 Inverse Transport Networks

Supervised learning provides an alternative to the analysis by synthesis methodology for inverse rendering problems, and has previously been successful for tasks such as reflectance, illumination, and shape inference [143, 212, 126, 97]. These prior successes motivate us to investigate the use of learning techniques for the inverse scattering problem.

Supervised learning assumes availability of a training set of image measurements $\{I_d\}_{d=1}^D$ and corresponding groundtruth material parameters $\{\pi_d\}_{d=1}^D$. Given a training dataset, learning techniques use empirical risk minimization to train a parametric regression model $N[w]$, e.g., a neural network, that directly maps images to parameters:

$$\hat{w} = \arg\min_w \sum_{d=1}^D \|\pi_d - N[w](I_d)\|^2.$$  \hspace{1cm} (5.2)

The trained network $N[\hat{w}]$ can be used to efficiently obtain parameter estimates $\hat{\pi}$ for new images $I$, through forward pass operations: $\hat{\pi} = N[\hat{w}](I)$. This is in contrast with analysis by synthesis, which requires solving the expensive optimization problem (5.1) for every new input image. Additionally, the trained
network can be used with images where other scene parameters are completely uncalibrated.

These advantages of supervised techniques come with the caveat that it is difficult to guarantee the accuracy of the estimates \( \hat{\pi} \) obtained for images of scenes that are not well represented in the training set. Given the highly non-linear mapping \( T \) from scene to images in the case of subsurface scattering, it is challenging to train networks that generalize well to scenes of, e.g., very different shape or illumination.

In order to combine the complementary advantages of learning and analysis by synthesis, we propose to regularize the training loss function (5.2) with a term that closely resembles the loss function (5.1) of analysis by synthesis:

\[
\hat{w} = \arg\min_w \sum_{d=1}^D \left[ \|\pi_d - N[w](I_d)\|^2 + \lambda \|I_d - T(N[w](I_d))\|^2 \right].
\]  

The regularization term in Equation (5.3) forces the neural network to predict parameters \( \pi_d \) that not only match the groundtruth, but also can be used with forward rendering to reproduce the input images. This has two desirable effects: First, the parameters predicted by the network are likely to be close to what would have been obtained from analysis by synthesis, as the regularization term in Equation (5.3) is equivalent to the analysis by synthesis loss (5.1). Second, the regularization term forces the neural network \( N[\hat{w}] \) to be approximately equal to the inverse of the volumetric light transport operator \( T \), that is, \( N[\hat{w}] \approx T^{-1} \). Given that \( T \) models the physics of subsurface scattering for scenes of arbitrary geometries and illumination, we expect the resulting neural network to generalize well to novel scenes. We refer to networks trained using the loss (5.2) as regressor networks (RN), and to networks trained using (5.3) as inverse transport networks (ITN), based on their above-discussed property.
Figure 5.1: **Inverse transport networks:** (a) Traditional autoencoders use two networks, encoder and decoder, to learn to predict parameters from images. (b) Inverse transport networks replace the decoder with a differentiable Monte Carlo renderer, to improve the generalization and physical accuracy of the predictions. During training, the renderer is provided with the material parameter output by the encoder network, as well as with groundtruth geometry and illumination, to perform forward and backward evaluations of an additional appearance-matching regularization term used to learn the network weights. During testing, the encoder network is used on its own, without the renderer: It takes as input a single, fully uncalibrated (unknown geometry and illumination) image, and produces as output a set of material parameters.

### 5.3.1 Relationship to Prior Work

Regularization similar to Equation (5.3) has previously appeared in two general forms. The first is autoencoder architectures [234, 106] that, in addition to the regressor (encoder) network $N[w]$ mapping images to parameters, use a second
decoder network $D[u]$ that maps the parameters back to images. Then, the regularization term in Equation (5.3) is replaced with $\|I_d - D[u](N[w](I_d))\|^2$, and both the encoder and decoder networks are trained simultaneously, potentially without access to groundtruth parameters (self-supervised learning). These architectures are of great utility when inferring semantic parameters (e.g., a class label) of a scene, where there is generally no analytical model for the forward mapping of these parameters to images. However, when the unknowns $\pi$ are scattering material parameters, autoencoder architectures do not take advantage of the rich knowledge we have about the physics governing the forward operator $T$. Additionally, the forward mapping $D[u]$ may not generalize to novel scenes, as it is specific to the training dataset. Figure 5.1 compares the autoencoder and inverse transport architectures.

There are also networks that use regularization terms where the light transport operator $T$ is replaced with an approximate rendering model [143, 212, 126, 224, 97]. These approximations generally use direct lighting models, where photons are assumed to only interact with the scene once between emission and detection (e.g., direct reflection without interreflections). Unfortunately, these networks have limited applicability to the case of inverse scattering, where the underlying physics are characterized by extremely multi-path, multi-bounce light transport. Inspired by these prior works, our ITNs are physics-aware learning pipelines that can be used even in the presence of these higher-order transport effects that are dominant in inverse scattering.
5.3.2 Training ITNs

The optimization problem (5.3) for ITN training is computationally challenging: Evaluating the operator $\mathcal{T}$ requires solving the radiative transfer equation [148]. In theory, training could be performed using algorithms such as REINFORCE [245], which do not require differentiating the regularization term and only employ graphics rendering algorithms for forward evaluations of $\mathcal{T}$. However, such algorithms are known to suffer from slow convergence.

Instead, we aim to optimize the loss (5.3) with state-of-the-art stochastic gradient descent algorithms [105]. This requires using differentiable rendering algorithms to estimate derivatives of $\mathcal{T}$ with respect to material parameters $\pi$ in an unbiased manner. For this, we rely on prior work [60, 259, 103, 56] that devised Monte Carlo rendering algorithms for simulating these derivatives by simulating the full volumetric light transport in a physically-accurate way. These algorithms have subsequently been generalized to scene parameters such as reflectance [131, 10], geometry [226], and pose [119, 253, 164]. For completeness, we provide below an overview of the differentiable rendering formulation at the basis of our work. We note that, because we optimize over only material parameters, our differentiable rendering formulation is significantly simpler than that required for dealing with global geometry changes, and which has been developed extensively in recent works [119, 253, 164].

Figure 5.1 provides an overview of our pipeline at training and test time: During training, the network is connected to the differentiable renderer. The network takes as input a single, high-dynamic-range image, and produces as output a set of scattering material parameters. During training, the network is connected to the differentiable renderer. The renderer takes as input the param-
eters produced by the network, as groundtruth geometry and illumination, to compute values and gradients of the regularization term in Equation (5.3). As we discuss in Section 5.5, because we train the network using synthetic input images, the geometry and illumination are readily available. During testing, the network is used on its own, without the renderer. As our objective is to use the network on testing images that are completely uncalibrated, no geometry or illumination information is given as input to the network during either training or testing.

5.3.3 Differentiable Monte Carlo Volume Rendering

To keep the paper self-contained, we provide a brief overview of forward and differentiable rendering in the context of subsurface scattering. Our discussion largely follows [56]. The starting point for both types of rendering is the path integral formulation of light transport, which expresses the images captured by a radiometric detector as integrals over the space of possible light paths [167]:

\[
\mathcal{T}(\pi) = \int_{\mathcal{P}} f[\pi](\bar{x}) \, d\bar{x}.
\]  

(5.4)

The above integration is performed over the space \( \mathcal{P} \) of all possible light paths of the form \( \bar{x} \equiv (x_0, x_1, \ldots, x_K) \), for any \( K > 1 \) and with \( x_k \in \mathbb{R}^3 \) (for \( k = 0, 1, \ldots, K \)). For each such path, \( x_0 \) is located on a light source, \( x_K \) on a sensor, and intermediate vertices \( x_k \) light-scene interactions via reflection, refraction, and subsurface scattering. The throughput function \( f[\pi] \) describes the amount of radiance contributed by a path as a function of the scene geometry, material, illumination, and detector.

By differentiating Equation (5.4) and rearranging throughput terms and their
derivatives, we can obtain a similar path integral expression for the derivatives \( \frac{\partial T(\pi)}{\partial \pi} \) of images with respect to the scattering parameters \( \pi \):

\[
\frac{\partial T(\pi)}{\partial \pi} = \int_{\mathbb{P}} f[\pi](\bar{x}) S[\pi](\bar{x}) \, d\bar{x}.
\] (5.5)

Compared to Equation (5.4), the path integral for this case includes the score function \( S[\pi] \), that sums derivatives of the per-vertex throughput with respect to \( \pi \).

Monte Carlo rendering algorithms evaluate the integrals of Equations (5.4) and (5.5) using Monte Carlo integration: (i) paths \( \{\bar{x}_n : n = 1, \ldots, N\} \) are drawn from a probability density \( p \) over the path space \( \mathbb{P} \); (ii) their throughputs \( f_s[\pi] \) are computed; and (iii) unbiased and consistent estimators of Equations (5.4)
and (5.5) are formed as

\[
\langle T(\pi) \rangle = \frac{1}{N} \sum_{n=1}^{N} f[\pi](\bar{x}_n) / p(\bar{x}_n), \tag{5.6}
\]

\[
\langle \partial T(\pi) / \partial \pi \rangle = \frac{1}{N} \sum_{n=1}^{N} \frac{f[\pi](\bar{x}_n) S[\pi](\bar{x}_n)}{p(\bar{x}_n)}. \tag{5.7}
\]

We use our own implementation of differentiable rendering: We integrated the Stan Math Library [26] for automatic differentiation of throughput terms, with the Mitsuba engine [82] for physically accurate Monte Carlo rendering. We use Mitsuba’s volumetric path tracing algorithm to sample paths for forming the estimates of Equations (5.6) and (5.7). Even though our focus is on inverse scattering, our implementation is a general-purpose differentiable renderer that can compute derivatives for scene parameters such as normals, reflectance, and illumination. We verified correctness of our derivatives by comparing derivatives computed using finite differences. An example comparison is shown in Figure 5.2. Note that the finite-difference gradients required more than two million samples-per-pixel, compared to 16384 samples-per-pixel used by the differentiable renderer. This shows the critical performance advantages of using differentiable rendering instead of numerical differentiation, which have also been well-documented in the past [119, 253, 164, 60, 259, 103, 56].

5.3.4 Stochastic Optimization

In addition to physical accuracy, Monte Carlo differentiable rendering provides computational advantages in the context of gradient-based optimization. In particular, training deep neural networks strongly relies on the ability to perform backpropagation in a stochastic manner, by computing derivatives of the loss function (5.2) using random subsets of the training set (minibatches). Changer-
ing the minibatch size allows controlling the tradeoff between the cost of gradient computations and the number of iterations for convergence [23, 107].

Monte Carlo differentiable rendering offers control over a similar capability: We can reduce the number of sampled paths to accelerate derivative computation, at the cost of increased variance. As the Monte Carlo derivative estimates are consistent and unbiased, we can use this to take advantage of the same convergence guarantees and tradeoffs as with stochastic backpropagation. Therefore, our Monte Carlo differentiable rendering engine is particularly well-suited for training of neural networks using state-of-the-art stochastic gradient descent algorithms [105].

5.3.5 Post-Learning Refinement

Our focus is on using inverse transport networks as an inference algorithm that can be used in place of analysis by synthesis optimization when the latter is not possible, e.g., when dealing with uncalibrated scenes of unknown geometry and illumination. We mention though, that inverse transport networks can be useful even when these scene parameters are calibrated and analysis by synthesis can be performed. In particular, the trained network \( N \) can be used to produce a first estimate of the unknown parameters \( \pi \) underlying an input image \( I \). This estimate can be used to \textit{warm-start} subsequent analysis by synthesis optimization, by serving as initialization for the gradient descent minimization of the analysis by synthesis loss (5.1) for the image \( I \). The effect of this warm-starting procedure is that the analysis by synthesis optimization can converge much faster than if we had skipped the network-based estimation stage and used a random
initial point. We expect the ITN architecture to be particularly effective for this kind of analysis by synthesis acceleration, given that the regularization term in its training loss function (5.3) encourages the network to produce estimates that are close to the analysis by synthesis solution. Additionally, the ITN-based initialization can help the analysis by synthesis minimization avoid local minima due to similarity relations.

5.4 Overcoming Similarity Relations

Similarity relations describe classes of material parameters that produce very similar appearance under certain geometry and lighting conditions [246, 255]. These relations are derived from the radiative transfer equation, and are well-studied in the subsurface scattering literature. We focus on first-order similarity relations, which are the most commonly-used class of material ambiguities: Two materials $\pi$ and $\pi'$ are considered similar if they satisfy,

$$
\sigma_a = \sigma'_a, \quad \sigma_s \cdot (1 - \bar{c}) = \sigma'_s \cdot (1 - \bar{c}').
$$

These ambiguities can be problematic for both analysis by synthesis and supervised learning techniques for inverse scattering. In the following, we discuss two strategies for ameliorating the negative effect of similarity relations.

Material parameterization. There are several redundant parameters that are typically used to characterize the space of scattering materials. Prior work has considered parameterizations in terms of non-linear functionals of these parameters that, when used with analysis by synthesis, reduce estimation errors due to similarity relations [255]. We take advantage of this prior work, and use these
parameterizations in the loss functions (5.2) and (5.3). We parameterize the material space as $\pi = \{\sigma_a, \sigma_s, \sigma_r^s\}$, where $\sigma_r^s$ is the reduced scattering coefficient,

$$\sigma_r^s = \sigma_s \cdot (1 - \bar{c}).$$

(5.9)

Intuitively, the robustness of this parameterization is due to the fact that the reduced scattering coefficient exactly matches the second similarity relation equation (5.8). The same parameterization also arises when deriving the reduced scattering properties of diffusion-based subsurface scattering [86]. Throughout the rest of the paper, we refer to this as the similarity-aware parameterization. In Section 5.5, we compare this with other naive parameterizations in the context of supervised learning for inverse scattering.

**Per-pixel weight maps.** Similarity relations are derived under the assumption that photons perform a large number of scattering events inside an object. As a consequence, their accuracy is strongly-dependent on scene conditions such as illumination and shape. Specifically, at thin parts of the object or parts of the surface with sharp geometric features (e.g., geometric edges), two materials will have different appearance even if their scattering parameters satisfy the similarity relations. The importance of thin geometric features for translucent appearance is well-documented in the literature, even beyond similarity considerations. These features has been shown to provide rich information about the scattering parameters of an object [57], and to be important for the perception of translucency by humans [46, 58, 32].

Motivated by the above, we modify the regularization term in the training loss (5.3), to use a per-pixel weight map,

$$\sum_{i,j} \left\| w_{ij}^d (I_d^i - T (N[w] (I_d)))^2 \right\|.$$ 

(5.10)
where the superscript $ij$ indicates indexing an image at pixel coordinates $[i, j]$, and summation is done over all pixels. For each image $I_d$ in the training dataset, the per-pixel weights are selected to emphasize pixels corresponding to parts of the object with thin geometry, where similarity relations are not accurate.

Determining optical thickness, that is, the average distance light travels inside the object, requires knowing the groundtruth shape and illumination. In lieu of these, we use a simple algorithm for generating a weight map from only the input image $I_d$: Considering that, in a textureless homogeneous material, all image-space edges correspond to geometric discontinuities, we first process the image $I_d$ with an edge detector, then assign to each pixel $[i, j]$ a weight $w_{ij}^d$ equal to its distance from the nearest edge. Figure 5.3 shows example weight maps created this way. As this weight map is computed from only the input image without requiring any additional information, we additionally provide it as an input to the network during both training and testing. Despite its simplicity, the figure and the results of Section 5.5 show that our algorithm is robust enough to produce meaningful weight maps resulting in significant performance improvements for a large variety of geometries.

5.5 Experiments

We evaluate the performance of different neural networks through experiments on simulated datasets and real images.

**Network details.** We compare neural networks trained with five different loss functions. First, a regressor network (RN), trained using the purely supervised
Figure 5.3: **Weight maps:** We use per-pixel weights equal to each pixel’s distance from the nearest image edge, in order to emphasize image pixels where similarity relations are violated.

loss (5.2), and the *naive parameterization* $\pi = \{\sigma, \alpha, \bar{c}\}$. Second, an RN using the similarity-aware parameterization of Section 5.4, $\pi = \{\sigma_a, \sigma_r, \sigma_s\}$. Third, an inverse transport network (ITN), trained using the regularized loss (5.3), and the naive parameterization. Fourth, an ITN that uses the regularization (5.3) and the similarity-aware parameterization. Fifth and last, an ITN that uses the weighted regularization (5.10) and the similarity-aware parameterization.

All networks take as input a single high-dynamic-range image. For all networks, we use a state-of-the-art architecture for inverse rendering problems relating to homogeneous reflectance [126, 143]: Each network is composed of seven convolutional layers, and the size of the output channel for each layer is reduced to half the size of its input. The kernel size for the convolutional layer is 3 by 3, with a stride of 2 and padding of 1. Each convolutional layer is followed by a rectified linear unit (ReLU) and a max-pooling layer. A fully-connected layer is used at the end. We visualize this architecture in Figure 5.4. We select this architecture as it reflects the state-of-the-art in the supervised deep material task that is closest to ours: inferring homogeneous BRDF parameters (as
far as we know, there is no prior work on estimating homogeneous subsurface scattering). The use of this architecture ensures that the RN with the naive parameterization can serve as meaningful baselines for evaluating the importance of our various innovations (similarity-aware parameterization, regularization using the differentiable Monte Carlo renderer, and weight map).

Figure 5.4: **Datasets and architecture:** (a) We render 30,000 images for the task of homogeneous inverse scattering. These are split into training and testing sets of different shapes, illuminations, and materials. Rendered grayscale images under the same illumination and shape but with different material parameters are combined to form color images. (b) We use these datasets to train and evaluate regressor and inverse transport networks of the same architecture.

When training ITNs, we use as initialization an RN trained for a few epochs. We set $\lambda$ in Equation (5.3) so that the supervised and regularization terms have...
Figure 5.5: Images rendered with predicted material parameters: Each column corresponds to a different input image drawn from our synthetic testing set. The last column shows images rendered under the novel scene used to emphasize generalization performance. For each image, different rows compare the groundtruth (row 1) to images rendered using the parameters predicted by the five networks we evaluate (rows 2-6).
approximately the same magnitude. All networks are trained using Adam [105] for 50 epochs, with a batch size of 60 and learning rate of $10^{-4}$.

Datasets. For our quantitative comparisons, we use a synthetic dataset containing images of translucent objects with varying geometry, illumination, and material parameters. We use ten different object shapes, selected to have a variety of thin and thick geometric features, each placed under ten different illumination conditions created using the Hošek-Wilkie sun-sky model [75]. For each shape and illumination combination, we render images for different parameters $\pi$ that include $\sigma_t \in [25\;\text{mm}^{-1}, 300\;\text{mm}^{-1}]$, $\alpha \in [0.3, 0.95]$, and Henyey-Greenstein phase functions $f_p$ with parameter $g \in [0, 0.9]$. We use the Mitsuba physics-based renderer [?] to simulate 30,000 high-dynamic range images under these settings.

We focus on evaluating the ability of the different networks to generalize to scenes containing new shapes and illuminations. For this, we separate our rendered images into training and testing sets that do not contain any overlapping shapes or illuminations. In particular, we use the images for six shapes and four illuminations as the training set, and use images under the remaining shape and illumination combinations for testing (Figure 5.4). This yields a training set of 6,000 images and a testing set of 7,000 images (we exclude a few thousand images available in the dataset that mix training illuminations with testing shapes, or vice versa). Our use of a training set that is relatively small compared to testing, and a testing set that contains only completely unseen shapes and illuminations, both reflect our goal to evaluate the generalization properties of the five networks we consider.
Table 5.1: **Network performances**: Average MSE for individual scattering parameters, as well as $L_2$ and $1 - \text{MS-SSIM}$ image appearance errors, for five different networks.

<table>
<thead>
<tr>
<th>network</th>
<th>parameters $\sigma_t$</th>
<th>parameters $\alpha$</th>
<th>parameters $\bar{c}$</th>
<th>$L_2$ error</th>
<th>$1 - \text{MS-SSIM}$ error</th>
</tr>
</thead>
<tbody>
<tr>
<td>RN+naive</td>
<td>90.81</td>
<td>0.180</td>
<td>0.400</td>
<td>0.314</td>
<td>0.069</td>
</tr>
<tr>
<td>RN+similar</td>
<td>77.45</td>
<td>0.141</td>
<td>0.374</td>
<td>0.180</td>
<td>0.036</td>
</tr>
<tr>
<td>ITN+naive</td>
<td>71.23</td>
<td>0.093</td>
<td><strong>0.253</strong></td>
<td>0.222</td>
<td>0.031</td>
</tr>
<tr>
<td>ITN+similar</td>
<td>64.62</td>
<td>0.116</td>
<td>0.273</td>
<td><strong>0.144</strong></td>
<td>0.032</td>
</tr>
<tr>
<td>ITN+similar+map</td>
<td><strong>60.32</strong></td>
<td><strong>0.088</strong></td>
<td>0.282</td>
<td>0.145</td>
<td><strong>0.024</strong></td>
</tr>
</tbody>
</table>

Finally, we note that, even though all networks are trained on grayscale images, they can handle color images by processing each color channel independently. Throughout this paper and in the supplement, we visualize results using color images, synthesized by combining grayscale images from our dataset that have the same illumination and shape conditions, but different material parameters. These color images are processed by the networks in the above-described channel-by-channel manner.

**Quantitative Evaluation.** We evaluate the five networks in two ways: First, we consider how accurately they predict material parameters for images in the testing dataset. Accuracy is quantified using the $L_2$ error between groundtruth and predicted parameters. Second, we examine how well images rendered with the predicted parameters match the appearance of the input images. We compare rendered and input images using $L_2$ error and MS-SSIM [239] (a benchmark perceptually-motivated image similarity metric).
Table 5.1 summarizes the results. The ITN with similarity-aware parameterization and weight maps outperforms the other three networks in most metrics, except for $\bar{c}$ where its performance is a close second. In terms of parameter prediction, it is noteworthy that the similarity-aware parameterization outperforms the naive parameterization for both the RN and ITN cases, despite the fact that errors are computed directly on the parameters optimized by the naive parameterization ($\sigma_t$, $\alpha$, and $\bar{c}$). This highlights the importance of accounting for similarity relations when designing networks for inverse scattering. When comparing ITNs with RNs, the ITN produces strong improvements in both parameter and appearance predictions regardless of what parameterization is used. These improvements provide strong evidence that the regularization term in Equation (5.3) allows the ITN to generalize better to unseen shapes and illuminations. Finally, Figures 5.5 show images rendered with parameters predicted by the five networks, for materials of varying optical thickness, including diffusive, turbid, and very optically thin materials. In all cases, the ITN with similarity-aware parameterization and weight map produces the images that best match the reference.

**Evaluation under novel scene.** The previous results already focus on the generalization performance of the trained networks, considering that the testing images have shape and illumination conditions that are completely absent from the training dataset. To further emphasize generalization performance, we perform an additional set of experiments: We use the networks to predict material parameters for all test images. We then use these parameter estimates, as well as their groundtruth values, to render images for a scene of completely new geometry and lighting, absent from both training and testing datasets. We compare
Table 5.2: **Network performances**: Average $L_2$ and $1-\text{MS-SSIM}$ image appearance errors, for five different networks, on the novel scene used to emphasize generalization performance.

<table>
<thead>
<tr>
<th>network</th>
<th>appearance</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$L_2$</td>
<td>$1 - \text{MS-SSIM}$</td>
</tr>
<tr>
<td>RN+naive</td>
<td>0.250</td>
<td>0.067</td>
</tr>
<tr>
<td>RN+similar</td>
<td>0.144</td>
<td>0.035</td>
</tr>
<tr>
<td>ITN+naive</td>
<td>0.186</td>
<td>0.031</td>
</tr>
<tr>
<td>ITN+similar</td>
<td>0.140</td>
<td>0.035</td>
</tr>
<tr>
<td>ITN+similar+map</td>
<td><strong>0.133</strong></td>
<td><strong>0.021</strong></td>
</tr>
</tbody>
</table>

these renderings using the same image similarity metrics as above. Table 5.2 shows the results, and Figure 5.5 (rightmost column) visualizes a few example images rendered on this novel scene. We see that the ITN with similarity-aware parameterization and weight map significantly outperforms all other networks, and produces images very similar to those rendered with the groundtruth parameters. This provides evidence that this network can infer reliable estimates of the true scattering paremeters underlying a translucent object, from just a single, completely uncalibrated image of that object.

**Initialization of analysis by synthesis.** As we discussed in Section 5.3, our main focus is on using neural networks to predict scattering parameters from completely uncalibrated photographs, where analysis by synthesis optimization is not possible due to lack of information on geometry and illumination. When this information is available, analysis by synthesis will typically produce more accurate material parameters than our method, considering that analysis by synthesis uses significantly more information about the object. However,
even in such cases, our networks can still be useful, as they provide a way to warm-start subsequent analysis by synthesis optimization, accelerating its convergence and improving the quality of the resulting material estimates.

To quantify the performance difference between our networks and analysis by synthesis, as well as the advantage that can be gained from warm-starting, we perform the following experiment: We randomly select a subset of 40 images from the testing set (10 for each shape in that set), and use our five networks to predict material parameters. Then, we use these parameters to initialize analysis by synthesis optimization for each image: We use our differentiable renderer together with the groundtruth scene information (e.g., geometry and illumination) to compute derivatives of the loss of Equation (5.1) with respect to the parameters $\pi$, and use ADAM to optimize the values of these parameters, starting from the values predicted by the networks. Each optimization procedure is run for 150 iterations, and we record the average MSE of individual parameters across all 40 images at every 30 iterations. The results are shown in Table 5.3. We make two observations: First, as expected, analysis by synthesis optimization takes advantage of the additional scene information it has access to, to significantly improve the initial parameter predictions of the networks (average MSE reduction of 43.8%). Second, initializing with our ITN with similarity-aware parameterization and weight map provides more than 2x convergence speedup compared to the baseline RN, resulting in 53.5% better average MSE at the same number of iterations. Our ITN with similarity-aware parameterization and weight map additionally outperforms all other networks.

The improvements observed above are in part due to the fact that a better initialization can help the analysis by synthesis procedure avoid local min-
ima due to similarity relations. Figure 5.6 shows a simple demonstration: The ITN with similarity-aware parameterization and weight map produces an initialization that is closer to the true parameters than the one by the ITN with naive parameterization. Additionally, after 50 gradient descent iterations, the optimization initialized by the ITN with similarity-aware parameterization and weight map has converged to parameters closer to the groundtruth, also producing 7.7% lower image error. From the optimization trajectories at the top of Figure 5.6, we observe that, after a few iterations, the optimization initialized by the ITN with naive parameterization moves along a contour of constant $\sigma'_s$ (visualized by the color map). This indicates that it may be trapped at a local minimum. The bottom row of Figure 5.6 compares the groundtruth image with renderings using the initial and optimized material predictions from the ITN with similarity-aware parameterization and weight map.

We observe that, even though the renderings do not reproduce the appearance of the original objects perfectly, in all cases the parameter predictions produce plausible appearance, especially considering the complete lack of calibration. The appearance errors are in part because of the mismatched geometry and lighting, and the fact that we do not model surface reflectance. In particular, the rendered images reproduce important features of translucent appearance, e.g., the intensity gradients near geometric edges. Qualitatively, the ITN with similarity-aware parameterization and weight map performs the best among the three networks, in terms of both matching the overall intensity of the real objects, and intensity gradients at geometric edges. We consider these results a promising start towards uncalibrated and computationally efficient inverse subsurface scattering on images captured in-the-wild.
Table 5.3: **Network performances on initialization of analysis by synthesis**: Average MSE for individual scattering parameters over iterations for five different networks.

<table>
<thead>
<tr>
<th>number of iterations</th>
<th>1</th>
<th>30</th>
<th>60</th>
<th>90</th>
<th>120</th>
<th>150</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RN+naive</strong></td>
<td>90.40</td>
<td>72.87</td>
<td>60.93</td>
<td>55.58</td>
<td>52.22</td>
<td>49.26</td>
</tr>
<tr>
<td><strong>RN+similar</strong></td>
<td>59.23</td>
<td>43.96</td>
<td>34.36</td>
<td>28.87</td>
<td>25.41</td>
<td>23.67</td>
</tr>
<tr>
<td><strong>σt</strong></td>
<td>60.02</td>
<td>57.82</td>
<td>51.36</td>
<td>47.27</td>
<td>43.69</td>
<td>40.51</td>
</tr>
<tr>
<td><strong>ITN+naive</strong></td>
<td>50.45</td>
<td>39.47</td>
<td>33.09</td>
<td>28.78</td>
<td>25.53</td>
<td>23.27</td>
</tr>
<tr>
<td><strong>ITN+similar</strong></td>
<td>43.98</td>
<td>32.42</td>
<td>26.08</td>
<td>23.71</td>
<td>22.48</td>
<td>21.36</td>
</tr>
<tr>
<td><strong>ITN+similar+map</strong></td>
<td>0.138</td>
<td>0.095</td>
<td>0.096</td>
<td>0.093</td>
<td>0.089</td>
<td>0.084</td>
</tr>
<tr>
<td><strong>α</strong></td>
<td>0.078</td>
<td>0.058</td>
<td>0.053</td>
<td>0.049</td>
<td>0.046</td>
<td>0.044</td>
</tr>
<tr>
<td><strong>ITN+naive</strong></td>
<td>0.102</td>
<td>0.090</td>
<td>0.084</td>
<td>0.077</td>
<td>0.071</td>
<td>0.066</td>
</tr>
<tr>
<td><strong>ITN+similar</strong></td>
<td>0.073</td>
<td>0.057</td>
<td>0.051</td>
<td>0.046</td>
<td>0.043</td>
<td>0.040</td>
</tr>
<tr>
<td><strong>ITN+similar+map</strong></td>
<td>0.057</td>
<td>0.052</td>
<td>0.047</td>
<td>0.043</td>
<td>0.041</td>
<td>0.038</td>
</tr>
<tr>
<td><strong>c</strong></td>
<td>0.304</td>
<td>0.279</td>
<td>0.266</td>
<td>0.268</td>
<td>0.264</td>
<td>0.252</td>
</tr>
<tr>
<td><strong>RN+naive</strong></td>
<td>0.232</td>
<td>0.229</td>
<td>0.202</td>
<td>0.178</td>
<td>0.158</td>
<td>0.145</td>
</tr>
<tr>
<td><strong>ITN+naive</strong></td>
<td>0.284</td>
<td>0.246</td>
<td>0.224</td>
<td>0.207</td>
<td>0.190</td>
<td>0.178</td>
</tr>
<tr>
<td><strong>ITN+similar</strong></td>
<td>0.233</td>
<td>0.234</td>
<td>0.210</td>
<td>0.178</td>
<td>0.158</td>
<td>0.143</td>
</tr>
<tr>
<td><strong>ITN+similar+map</strong></td>
<td>0.240</td>
<td>0.202</td>
<td>0.172</td>
<td>0.151</td>
<td>0.139</td>
<td>0.128</td>
</tr>
</tbody>
</table>

5.6 Conclusion

We have taken first steps towards using deep learning techniques for the problem of homogeneous inverse scattering. Starting from a state-of-the-art regression network architecture as baseline, we made three innovations, informed from the physics of radiative transfer: First, we introduced inverse transport networks as an architecture that can combine the efficiency of neural networks with the generality properties of analysis by synthesis. Second, we used ma-
Figure 5.6: **Post-learning refinement**: We use predictions from two ITNs to initialize analysis by synthesis optimization. The top row shows how parameters change during gradient descent, in the \( \{\sigma_s, \sigma_a, \bar{c}\} \) parameter space, and in its 2D projection \( \{\sigma_s, \bar{c}\} \) colored by reduced scattering coefficient value \( \sigma_r \). The bottom row compares groundtruth to renderings using the initial and final parameter estimations from the green optimization trajectory.

Material parameterizations that can ameliorate ambiguities in the scattering parameter space due to similarity relations. Third, we utilized per-pixel weight maps to emphasize parts of the image that are informative about the underlying scattering parameters. Our experiments show that the combination of these innovations results in networks that can produce convincing scattering material parameter estimates, when provided with a single photograph without any calibration (completely unknown geometry and illumination). Additionally, the performance of our networks shows strong improvements in both parameter estimation accuracy and appearance reproduction compared to the baseline.
Figure 5.7: **Real photographs:** We take photographs of two translucent objects, a silicone cube (top) and a soap bar (bottom), under unknown shape and illumination (left column). We use three ITNs to predict scattering parameters for the objects. We evaluate the predicted parameters by rendering images for an approximate reconstruction of the original scenes (middle and right column).

We hope that these results will motivate follow-up work on using data-driven learning techniques to improve upon and complement existing physics-based approaches for inverse subsurface scattering.

At the core of our approach is the use of Monte Carlo differentiable renderers. The use of physically-accurate rendering allows us to enhance the generalization of neural networks, and the differentiability allows us to efficiently train these networks. Together with other work on combining differentiable rendering with learning [119], our results point towards a new direction of exploration: the investigation of more general learning architectures that intelligently combine neural networks with physics-based light transport simulation.
CHAPTER 6
SHAPE AND REFLECTANCE RECOVERY USING DIFFERENTIABLE RENDERING

6.1 Introduction

Reconstructing the shape and appearance of real-world objects from 2D images has been a long-standing problem in computer vision and graphics. Previously, the acquisition of object geometry and (spatially varying) reflectance has been studied largely independently. For instance, many techniques based on multiview-stereo (MVS) [54, 209, 228, 155, 4, 76, 195, 196] and time-of-flight imaging [161, 81] have been introduced for the reconstruction of 3D shapes. Although these methods can also provide rough estimations of surface reflectance, they usually rely on the assumption of simple (e.g., diffuse-dominated) reflectance and can produce unsatisfactory results for glossy objects. On the other hand, previous approaches that specialized at recovering an object’s spatially varying reflectance [264, 50, 63] typically require object geometries to be predetermined, limiting their practical usage for many applications where such information is unavailable.

In this paper, we demonstrate that detailed geometry and spatially varying reflectance of a real-world object can be recovered using a unified analysis-by-synthesis framework. To this end, we apply gradient-based optimization of the rendering loss (i.e., the difference between rendered and target images) that are affected by both object geometry and reflectance. Although such gradients with respect to appearance are relatively easy to compute, the geometric gradients are known to be much more challenging to compute and, therefore, have been
mostly approximated in the past using techniques like soft rasterization [128] in computer vision. We, on the other hand, leverage recent advances in physics-based differentiable rendering to obtain unbiased and consistent geometric gradients that are crucial for obtaining high-quality reconstructions.

Concretely, our contributions include:

- A Monte Carlo differentiable renderer specialized for collocated configurations. Utilizing edge sampling [118], our renderer produces unbiased and consistent gradient estimates.
- A new analysis-by-synthesis pipeline that enables high-quality reconstruction of spatially varying reflectance and, more importantly, mesh-based object geometry.
- A coarse-to-fine scheme as well as geometry and reflectance priors for ensuring robust reconstructions.

We evaluate various key aspects of our method via several ablation studies. Further, we demonstrate the effectiveness of our technique via both synthetic and real examples.

## 6.2 Overview

We formulate the problem of joint estimation of object geometry and reflectance as an analysis-by-synthesis (aka. inverse-rendering) optimization. Let $\xi$ be some vector that depicts both the geometry and the reflectance of a real-world object. Taking as input a set of images $\tilde{I}$ of this object, we estimate $\xi$ by minimizing a
predefined loss $\mathcal{L}$:

$$
\xi^* = \arg\min\xi \mathcal{L}(I(\xi), \xi; \tilde{I}), \tag{6.1}
$$

where $I(\xi)$ are a set of renderings of the object generated using the geometry and reflectance provided by $\xi$. We further allow the loss $\mathcal{L}$ to directly depend on the parameters $\xi$ for regularization.

**Acquisition setup.** Similar to recent works on reflectance capture [4, 5, 76, 195, 3, 49, 37, 193], we utilize an acquisition setup where the object is illuminated with a point light collocated with the camera. Common examples include a smart phone’s flash and camera as well as a consumer-grade RGBD sensor mounted with an LED light.

**Overview of our method.** Efficiently solving the optimization of Eq. (6.1) requires computing gradient $\frac{\partial \mathcal{L}}{\partial \xi}$ of the loss $\mathcal{L}$ with respect to the geometry and reflectance parameters $\xi$. According to the chain rule, we know that

$$
\frac{d\mathcal{L}}{d\xi} = \frac{\partial \mathcal{L}}{\partial I} \frac{dI}{d\xi} + \frac{\partial \mathcal{L}}{\partial \xi}, \tag{6.2}
$$

where $\frac{\partial \mathcal{L}}{\partial I}$ and $\frac{\partial \mathcal{L}}{\partial \xi}$ can be computed using automatic differentiation [177]. Further, estimating gradients $\frac{dI}{d\xi}$ of rendered images requires performing differentiable rendering. Despite being relatively easy when the parameters $\xi$ only capture reflectance, differentiating the rendering function $I$ becomes much more challenging when $\xi$ also controls object geometry [118]. To this end, we develop a new differentiable renderer that is specific to our acquisition setup and provides unbiased gradient estimates.

In the rest of this chapter, we provide a detailed description of our technique that solves the analysis-by-synthesis optimization (6.1) in an efficient and
Figure 6.1: **Differentiable rendering:** (a) To properly differentiate the intensity $I_j$ of a pixel $P_j$ (illustrated as red squares) with respect to object geometry, a boundary integral over $\Delta P_j$ (illustrated as the orange curve) needs to be calculated. (b) We perform Monte Carlo edge sampling [118, 252] by (i) sampling points (illustrated as small discs) from pre-computed discontinuity curves $\Delta P$, and (ii) accumulating their contributions in the corresponding pixels (e.g., the orange samples contribute to the pixel $P_j$).

robust fashion. In §6.3.1, we detail our forward-rendering model and explain how it can be differentiated. In §6.3.2, we discuss our choice of the loss $\mathcal{L}$ and optimization strategy.

### 6.3 Our Method

#### 6.3.1 Forward and Differentiable Rendering

In what follows, we describe (i) our representation of object geometry and reflectance; and (ii) how we render these representations in a differentiable fashion.
Object geometry and reflectance. We express object geometries using standard triangle meshes. Compared to other representations that are popular in 3D reconstruction, such as SDF volumes [174, 88], occupancy networks [144] or sphere-based clouds [110], triangle meshes can be efficiently rendered and edited with many 3D digital content creation tools. Further, as a widely adopted format, triangle meshes can be easily imported into numerous applications in computer graphics, vision, and augmented/virtual reality (AR/VR).

One of the biggest challenges when using meshes for 3D reconstruction is that topological changes are difficult. We show in the following sections that this can be addressed by using reasonable initial geometries and a coarse-to-fine optimization process.

To depict an object’s spatially varying reflectance, we use the Disney BRDF [95], a parametric model offering a good balance between simplicity and flexibility. This model has also been used by many prior works (e.g., [124, 123, 20]). Using this BRDF model, the spatially varying reflectance of an object is described using four 2D texture maps specifying, respectively, diffuse albedo $a_d$, specular albedo $a_s$, surface roughness $\alpha$, and normal $n$. Thanks to the efficiency of our system (which we will present in the following), we directly use fine meshes to express detailed geometries and do not rely on approximations like bump/normal mapping.

Forward rendering. Given a virtual object depicted using parameters $\xi$, we render one-bounce reflection (aka. direct illumination) of the object. Specifically, assume the point light and the camera are collocated at some $o \in \mathbb{R}^3$. Then, the intensity $I_j$ of the $j$-pixel is given by an area integral over the pixel’s footprint.
\( \mathcal{P}_j \), which is typically a square on the image plane:

\[
I_j = \frac{1}{|\mathcal{P}_j|} \int_{\mathcal{P}_j} I_e \frac{f_r(o \rightarrow y \rightarrow o)}{\|y - o\|^2} \, dA(x),
\]

(6.3)

where \( y \) is the intersection between the object geometry \( M \) and a ray that originates at \( o \) and passes through \( x \) on the image plane. Further, \( I_e \) denotes the intensity of the point light; \( f_r(o \rightarrow y \rightarrow o) \) indicates the cosine-weighted BRDF at \( y \) (evaluated with both the incident and the outgoing directions pointing toward \( o \)); and \( A \) is the surface-area measure. We note that no visibility check is needed in Eq. (6.3) since, under the collocated configuration, any point \( y \in M \) visible to the camera must be also visible to the light source.

We estimate Eq. (6.3) using Monte Carlo integration by uniformly sampling \( N \) locations \( x_1, x_2, \ldots, x_N \in \mathcal{P}_j \) and computing \( I_j \approx \frac{1}{N} \sum_{i=1}^{N} I(x_i) \) where \( I \) is the integrand defined in Eq. (6.3).

**Differentiable rendering.** Computing image gradients \( \frac{dI}{\xi} \) in Eq. (6.2) largely boils down to differentiating pixel intensities Eq. (6.3) with respect to \( \xi \). Although this can sometimes be done by differentiating the integrand \( I \)—that is, by estimating \( \int_{\mathcal{P}_j} (\frac{dI}{\xi}) \, dA \)—doing so is insufficient when computing gradients with respect to object geometry (e.g., vertex positions). Consequently, the gradient \( \frac{dI}{\xi} \) has usually been approximated using soft rasterization [128, 189] or reparameterized integrals [133]. Biased gradient estimates, unfortunately, can reduce the quality of optimization results, which we will demonstrate in §A.5.

On the other hand, a few general-purpose unbiased techniques [118, 252] have been introduced recently. Unfortunately, these methods focus on configurations without point light sources—which is not the case under our collocated
configuration. We, therefore, derive the gradient \( \frac{dI_j}{d\xi} \) utilizing mathematical tools used by these works. Specifically, according to Reynolds transport theorem [194], the gradient involves an interior and a boundary integrals:

\[
\frac{dI_j}{d\xi} = \frac{1}{|P_j|} \left[ \int_{P_j} \frac{dI_j}{d\xi}(x) dA(x) + \int_{\Delta P_j} (n(x') \cdot \frac{dx'}{d\xi}) \Delta I(x') d\ell(x') \right],
\]

where the interior term is simply Eq. (6.3) with its integrand \( I \) differentiated. The boundary one, on the contrary, is over curves \( \Delta P_j := \Delta P \cap P_j \) with \( \Delta P \) comprised of jump discontinuity points of \( I \). In practice, \( \Delta P \) consists of image-plane projections of the object’s silhouettes. Further, \( n(x) \) is the curve normal within the image plane, \( \Delta I(x) \) denotes the difference in \( I \) across discontinuity boundaries, and \( \ell \) is the curve-length measure (see Figure 6.1-a).

Similar to the Monte Carlo estimation of Eq. (6.3), we estimate the interior integral in Eq. (6.4) by uniformly sampling \( x_1, \ldots, x_N \in P_j \). To handle the boundary integral, we precompute the discontinuity curves \( \Delta P \) (as polylines) by projecting the object’s silhouette onto the image plane. At runtime, we draw \( x_1', \ldots, x_M' \in \Delta P \) uniformly. Then,

\[
\frac{dI_j}{d\xi} \approx \frac{1}{N} \sum_{i=1}^{N} \frac{dI_j}{d\xi}(x_i) + \frac{1}{M} \sum_{i=1}^{M} \mathbb{1}[x_i' \in P_j] \left( n(x_i') \cdot \frac{dx_i'}{d\xi} \right) \Delta I(x_i'),
\]

where \( |\Delta P| \) denotes the total length of the discontinuity curves \( \Delta P \), and \( \mathbb{1}[] \) is the indicator function.

In practice, we estimate gradients of pixel intensities via Eq. (6.5) in two rendering passes. In the first pass, we evaluate the interior component indepen-
dently for each pixel. In the second pass, we evaluate the boundary component for each \( x'_i \) in parallel and accumulate the results in the corresponding pixel (see Figure 6.1-b).

### 6.3.2 Analysis-by-Synthesis Optimization

We now present our analysis-by-synthesis optimization pipeline that minimizes Eq. (6.1).

**Object parameters.** As stated in §6.3.1, we depict object geometry using a triangle mesh (which is comprised of per-vertex positions \( p \) and UV coordinates \( u \) as well as per-triangle vertex indices) and reflectance using three 2D texture maps specifying the object’s spatially varying diffuse albedo \( a_d \), specular albedo \( a_s \), and surface roughness \( \alpha \), respectively. In this way, our combined geometry and reflectance parameters are given by \( \xi = (p, u, a_d, a_s, \alpha) \). Note, we do not modify the connectivity of the triangle vertices and rely on additional re-meshing steps, which we will discuss in §6.3.4, to improve mesh topology.

**Loss.** A key ingredient in our analysis-by-synthesis optimization is the loss \( \mathcal{L} \). Let \( \tilde{I} := (\tilde{I}_1, \tilde{I}_2, \ldots) \) be a set of images of some object (with camera location and pose calibrated for each image \( \tilde{I}_k \)). Then, our loss takes the form:

\[
\mathcal{L}(I(\xi), \xi; \tilde{I}) := \mathcal{L}_{\text{rend}}(I(\xi); \tilde{I}) + \mathcal{L}_{\text{reg}}(\xi),
\]

(6.6)

where \( \mathcal{L}_{\text{rend}} \) is the rendering loss that measures the difference between rendered and target object appearances. Specifically, we set

\[
\mathcal{L}_{\text{rend}}(I(\xi); \tilde{I}) := \lambda_{\text{rend}} \sum_k \| \Phi_k(I_k(\xi)) - \Phi_k(\tilde{I}_k) \|_1,
\]

(6.7)
where $\lambda_{\text{rend}} > 0$ is a user-specified weight, $I(\xi) := (I_1(\xi), I_2(\xi), …)$ denotes images rendered using our forward-rendering model of Eq. (6.3) with object geometry and reflectance specified by $\xi$ (under identical camera configurations as the input images), and $\Phi_k$ captures pixel-wise post-processing operations such as tone-mapping and background-removing masking.

We estimate gradients of the rendering loss of Eq. (6.7) with respect to the object parameters $\xi$ using our differentiable rendering method described in §6.3.1. We will demonstrate in §6.4.1 that accurate gradients are crucial to obtain high-quality optimization results.

In Eq. (6.6), $L_{\text{reg}}(\xi)$ is a regularization term for improving the robustness of the optimization, which we will discuss in §6.3.3. Gradients of this term can be obtained easily using automatic differentiation.

**Optimization process.** Like any other analysis-by-synthesis method, our technique takes as input an initial configuration of an object’s geometry and reflectance. In practice, we initialize object geometry using MVS or Kinect Fusion. Our technique is capable of producing high-quality reconstructions using crude initializations (obtained using low-resolution and noisy inputs). For the reflectance maps, we simply initialize them as constant-valued textures.

Provided an initial configuration of the object’s geometry and reflectance, we minimize the loss of Eq. (6.6) using the Adam algorithm [105].

Further, to make the optimization more robust, we leverage a coarse-to-fine approach that periodically performs remeshing and upsamples the reflectance-describing textures. We will provide more details on this process in §6.3.4.
Figure 6.2: **Comparison** with COLMAP [207] and Kinect Fusion [161] using synthetic inputs. The COLMAP results (c) are generated using 50 RGB images (a) with exact camera poses; the KF-High (d1) and KF-Low (d2) results are created using 50 ground-truth depth images (b) and low-resolution noisy ones, respectively. Our method (e), when initialized with KF-Low (d2) and using RGB inputs (a), produces much more accurate geometries than COLMAP and KF-High. Similar to Figure 6.5, the numbers indicate average point-to-mesh distances.

### 6.3.3 Regularization

Using only the rendering loss $L_{\text{rend}}$ expressed in Eq. (6.7) can make the optimization unstable and/or converge to local minima. To address this problem, we regularize the optimization by introducing another loss $L_{\text{reg}}$ that in turn consists of a *material* loss $L_{\text{mat}}$ and a *mesh* one $L_{\text{mesh}}$:

$$L_{\text{reg}}(\xi) := L_{\text{mesh}}(M) + L_{\text{mat}}(a_d, a_s, \alpha), \quad (6.8)$$

which we will discuss in the following.

**Mesh loss.** We encourage our optimization to return “smooth” object geometry by introducing a *mesh loss*:

$$L_{\text{mesh}}(M) := L_{\text{lap}}(M) + L_{\text{normal}}(M) + L_{\text{edge}}(M), \quad (6.9)$$
where the mesh-Laplacian loss $L_{\text{lap}}$ of a mesh with $n$ vertices is given by $L_{\text{lap}}(M) := \lambda_{\text{lap}} \|LV\|^2$ where $V$ is an $n \times 3$ matrix with its $i$-th row storing coordinates of the $i$-th vertex, and $L \in \mathbb{R}^{n \times n}$ is the mesh’s Laplacian matrix [159].

Additionally, we use a normal-consistency loss $L_{\text{normal}}$ to encourage normals of adjacent faces to vary slowly by setting $L_{\text{normal}}(M) := \lambda_{\text{normal}} \sum_{i,j}[1 - (n_i \cdot n_j)]^2$, where the sum is over all pairs $(i, j)$ such that the $i$-th and the $j$-th triangles share a common edge, and $n_i$ and $n_j$ denote the normals of these triangles.

Lastly, we penalize the mesh for having long edges, which usually yield ill-shaped triangles, by utilizing an edge-length loss $L_{\text{edge}} := \lambda_{\text{edge}} \left(\sum_i e_i^2\right)^{1/2}$, where $e_i$ denotes the length of the $i$-th face edge.

**Material loss.** Our material loss $L_{\text{mat}}$ regularizes the reflectance maps representing diffuse albedo $a_{d}$, specular albedo $a_{s}$, and surface roughness $\alpha$. Specifi-
Figure 6.4: Ablation study on our material loss of Eq. (6.10). Using identical initial reflectance maps and optimization configurations, models optimized with various components of the material loss neglected are rendered under a novel environmental illumination. On the right of each reconstruction result, we show the optimized reflectance maps (from top to bottom: diffuse albedo, specular albedo, and roughness).

cally, we set
\begin{equation}
\mathcal{L}_{\text{mat}}(a_d, a_s, \alpha) := \mathcal{L}_{\text{spec}}(a_d, a_s) + \mathcal{L}_{\text{roug}}(\alpha),
\end{equation}

where $\mathcal{L}_{\text{spec}}$ correlates diffuse and specular albedos [204]: assuming nearby pixels with similar diffuse albedos to have similar specular ones, we set
\begin{equation}
\mathcal{L}_{\text{spec}}(a_s, a_d) := \lambda_{\text{spec}} \sum_p \|a_s[p] - (\sum_q a_s[q] \mu_{p,q})/(\sum_q \mu_{p,q})\|_1,
\end{equation}

where  
\begin{equation}
\mu_{p,q} := \exp\left(-\frac{\|p-q\|_2^2}{2\sigma_1^2} - \frac{(a_d[p]-a_d[q])^2}{2\sigma_2^2}\right)
\end{equation}

is the bilateral weight between pixels with indices $p, q \in \mathbb{Z}^2$.

Spatially varying surface roughness is known to be challenging to optimize even when the object geometry is known [49]. To regularize our optimization of surface roughness, we introduce a smoothness term that measures its total variation:  
\begin{equation}
\mathcal{L}_{\text{roug}}(\alpha) := \lambda_{\text{roug}} \sum_{i,j} (|\alpha[i+1, j] - \alpha[i, j]| + |\alpha[i, j+1] - \alpha[i, j]|),
\end{equation}

where $\alpha[i, j]$ indicate the value of the $(i, j)$-th pixel in the roughness map.

6.3.4 Improving Robustness

As described in §6.3.2, when minimizing the loss of Eq. (6.6), we keep the mesh topology unchanged. This, unfortunately, can severely limit the flexibility of
our optimization of object geometry, making the result highly sensitive to the quality of the initial mesh. Additionally, without taking precautions, updating vertex positions can introduce artifacts (e.g., self intersections) to the mesh that cannot be easily fixed by later iterations.

To address these problems, we utilize a few extra steps.

**Coarse-to-fine optimization.** Instead of performing the entire optimization at a single resolution, we utilize a coarse-to-fine process for improved robustness. Similar steps have been taken in several prior works, although typically limited to either geometry [213, 203, 99, 227] or reflectance [39, 195, 76].

Specifically, we start the optimization by using low-resolution meshes and reflectance maps. If the input already has high resolutions, we simplify them via remeshing and image downsampling. Our optimization process then involves multiple stages. During each stage, we iteratively refine the object geometry and reflectance with fixed mesh topology. After each stage (except the final one), we upsample the mesh (using instant meshes [85]) and the texture maps (using simple bilinear interpolation).

**Robust surface evolution.** During optimization, if the vertex positions are updated naïvely (i.e., using simple gradient-based updates with no validation checks), the mesh can have degraded quality and even become non-manifold (i.e., with artifacts like holes and self-intersections). Motivated by other optimization-driven mesh editing algorithms [202, 127], we evolve a mesh using a pipeline implemented in the El Topo library [24]: Given the initial positions of a set of vertices with associated displacements, El Topo moves each
Table 6.1: **Rendering performance.** We report the rendering cost in seconds (averaged across 100 times) of each differentiable renderer in resolution $512 \times 512$ and 4 samples per pixel on a Titan RTX graphics card.

<table>
<thead>
<tr>
<th></th>
<th>SoftRas</th>
<th>PyTorch3D</th>
<th>Mitsuba 2</th>
<th>Nvdiffrast</th>
<th>Redner</th>
<th>Ours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kettle</td>
<td>0.0184</td>
<td>0.0202</td>
<td>0.0877</td>
<td>0.0013</td>
<td>0.1196</td>
<td>0.0143</td>
</tr>
<tr>
<td>Maneki</td>
<td>0.0192</td>
<td>0.0224</td>
<td>0.0863</td>
<td>0.0010</td>
<td>0.1029</td>
<td>0.0146</td>
</tr>
<tr>
<td>Pig</td>
<td>0.0971</td>
<td>0.0772</td>
<td>0.0913</td>
<td>0.0014</td>
<td>0.1336</td>
<td>0.0263</td>
</tr>
<tr>
<td>Kitty</td>
<td>0.0249</td>
<td>0.0334</td>
<td>0.0889</td>
<td>0.0010</td>
<td>0.1190</td>
<td>0.0225</td>
</tr>
</tbody>
</table>

vertex along its displacement vector while ensuring no self-intersection is generated.

### 6.4 Results

We implement our differentiable renderer (§6.3.1) in C++ with CUDA 11 and OptiX 7.1. For vectorized and differentiable computations, we utilize the Enoki library [83], which has been demonstrated to be more efficient than generic ones like Tensorflow and PyTorch for rendering [164].

We implement the rest of our optimization pipeline, including the loss computations, in PyTorch. We use one set of weights for our optimizations: $\lambda_{\text{rend}} = 1$ for the rendering loss of Eq. (6.7); $(\lambda_{\text{lap}}, \lambda_{\text{edge}}, \lambda_{\text{normal}}) = (0.1, 1, 0.01)$ for the mesh loss of Eq. (6.9) and $(\lambda_{\text{spec}}, \lambda_{\text{roug}}) = (0.01, 0.001)$ for the material loss of Eq. (6.10).

In practice, our optimization involves 500–1000 iterations (for all coarse-to-fine stages) and takes 0.5–2 hours per example (see the supplement for more details).
6.4.1 Evaluations and Comparisons

Comparison with differentiable renderers. Our renderer enjoys high performance, thanks to its specialized nature (i.e., focused on the collocated configuration) and the combined efficiency of RTX ray tracing offered by OptiX and GPU-based differentiable computations by Enoki. As demonstrated in Table 6.1, our renderer is faster than SoftRas [128], PyTorch3D [189], and Mitsuba 2 [164] without the need to introduce bias to the gradient estimates. Nvdiffrast [109] is faster than our system but produces approximated gradients. Lastly, compared to Redner [118], another differentiable renderer that produces unbiased gradients, our renderer offers better performance.

To further demonstrate the importance for having accurate gradients, we conduct a synthetic experiment where the shape of an object is optimized (with known diffuse reflectance). Using identical input images, initial configurations, losses, and optimization settings (e.g., learning rate), we ran multiple optimizations using Adam [105] with gradients produced by SoftRas, PyTorch3D, Mitsuba 2, Nvdiffrast, and our technique, respectively. As shown in Figure 6.5, using biased gradients yields various artifacts or blurry geometries in the optimized results.

Effectiveness of shape optimization. To ensure robustness when optimizing the shape of an object, our technique utilizes a mesh loss (§6.3.3) as well as a coarse-to-fine framework (§6.3.4). We conduct another experiment to evaluate the effectiveness of these steps. Specifically, we optimize the shape of the pig model using identical optimization configurations except for (i) having various components of the mesh loss turned off; and (ii) not using the coarse-to-fine
Figure 6.5: Comparison with a few baselines. We render all reconstructed geometries using Phong shading and visualize depth errors (wrt. the ground-truth geometry). The number below each result indicates the average point-to-mesh distance capturing the Euclidean accuracy [87] of the reconstructed geometry (normalized to have a unit bounding box).
Reconstruction results using synthetic inputs: We obtain the initial meshes (in the left column) using Kinect Fusion [161] with low-resolution and noisy depths. Our method successfully recovers both geometric and reflectance details, producing high-fidelity results under novel rendering conditions.
Figure 6.7: **Reconstruction results** of real-world objects: Similar to the synthetic examples in Figure A.15, our technique recovers detailed object geometries and reflectance details, producing high-quality results under novel viewing and illumination conditions.
framework. As shown in Figure 6.3, with the mesh Laplacian loss $L_{\text{lap}}$ neglected (by setting $\lambda_{\text{lap}} = 0$), the resulting geometry becomes “bumpy”; without the normal and edge-length losses $L_{\text{normal}}$ and $L_{\text{edge}}$, the optimized geometry also has artifacts due to sharp normal changes and ill-shaped triangles. Additionally, without performing the optimization in a coarse-to-fine fashion (by directly starting with a subdivided version of the initial mesh), the optimization gets stuck in a local optimum (with all losses enabled).

**Effectiveness of material loss.** Our material loss of Eq. (6.10) is important for obtaining clean reflectance maps that generalize well to novel settings. As shown in Figure 6.4, without correlating diffuse and specular albedos (by having $\lambda_{\text{spec}} = 0$), diffuse colors are “baked” into specular albedo, leading to heavy artifacts under novel environmental illumination. With the roughness smoothness $L_{\text{roug}}$ disabled, the resulting roughness map contains high-frequency noise that leads to artifacts in rendered specular highlights.

**Comparison with previous methods.** To further evaluate the effectiveness of our technique for recovering object geometry, we compare with several previous methods [207, 161].

Figure 6.2 shows comparisons with COLMAP [207] and Kinect Fusion [161] using synthetic inputs. Our technique, when using crude initial geometries (obtained using Kinect Fusion with low-resolution and noisy depth images), produces results with much higher quality than the baselines. COLMAP fails badly for the *kitty* example since the object contains insufficient textures for correspondences to be reliable established.
6.4.2 Reconstruction Results

Figure A.15 shows reconstruction results obtained using synthetic input images and rendered under novel views and illuminations. The initial geometries are obtained using Kinect Fusion with low-resolution noisy depth inputs. Using 50 input images, our technique offers the robustness for recovering both smooth (e.g., the *kitty* example) and detailed (e.g., the *pig* example) geometries and reflectance.

Figure 6.8: **Applications**: The high-quality models generated by our technique can be used for 3D digital modeling (top) and object insertion in augmented reality (bottom).

We show in Figure 6.7 reconstruction results using as input 100 real pho-
tographs per example. The initial geometries are obtained using COLMAP. Our analysis-by-synthesis technique manages to accurately recover the detailed geometry and reflectance of each model.

Please note, in Figures A.15 and 6.7, the detailed geometric structures (e.g., those in the bell, pig, chime, and buddha examples) fully emerge from the mesh-based object geometries: no normal or displacement mapping is used.

Lastly, since our reconstructed models use standard mesh-based representations, they can be used in a broad range of applications (see Figure 6.8).

6.5 Conclusion and Discussion

We investigate new approaches to jointly recover the shape and reflectance of real-world objects with inverse rendering. At the core of our techniques is a unified analysis-by-synthesis pipeline that iteratively refines object geometry and reflectance. Our Monte Carlo differentiable renderer generates unbiased gradients for mesh vertices, which is crucial for high-quality mesh-based reconstruction and refinement. For robust optimization, we leverage a coarse-to-fine framework with a few regularization terms. To refine mesh topology, our technique relies on remeshing steps (between coarse-to-fine stages). How topology can be optimized in a robust and flexible fashion is an important problem for future investigation. Alternatively geometry representations such as SDFs or volume opacity fields (e.g., NeRF [147]) could potentially handle the topology with more flexibilities. Lastly, more advanced regularizations of geometry and/or appearance may enable high-quality reconstructions with fewer input images.
This thesis presented my main research projects during Ph.D. study, spanning across gradient-based MCMC rendering, inverse subsurface scattering, 3D object reconstruction, procedural cloth models, and image style transfer. At the core of these projects, optimization is the crucial ingredient.

Back in 2016, differentiable rendering wasn’t as popular as today. To inverse the parameters of procedural yarn models (see Appendix for the cloth project), we relied on traditional gradient-free optimization techniques (e.g., MLE, Nelder-Mead Simplex method). Although the proposed pipeline was robust on fitting real-world yarns, its alternative and iterative optimization scheme inevitably makes the system complex. With recent advances in physics-based Monte Carlo differentiable rendering, optimization for such tasks would be much simpler and more elegant nowadays.

People have witnessed the explosive growth of deep learning in computer graphics and vision in the last decade. With compact and convenient deep-learning libraries for prototyping and experiments (e.g., PyTorch [178]) and hardware acceleration (e.g., NVIDIA GPU compute), sophisticated neural networks have been proposed and exploited in challenging problems with improved performance compared to traditional methods. More importantly, recent advances in neural rendering [147] have demonstrated that physical priors (e.g., volume rendering) from computer graphics can strongly benefit the forward model, especially for neural network generalization.

Similarly as in Li et al. [120], we demonstrated that derivatives available in
recent differentiable renderers can be beneficial to speeding up forward rendering (Chapter 4), not limited to inverse applications. Starting from Langevin dynamics, we import gradient-based optimization techniques (e.g., Adam) into MCMC rendering, and validated our approach on various of scenes with challenging light transport effects (e.g., glossy interreflections, motion blur).

A key to bridge the gap between traditional methods and deep learning is differentiable rendering, or more broadly, differentiable graphics pipelines. Our inverse subsurface scattering project (Chapter 5) takes a step towards this direction, and showed that using differentiable renderer as a neural network layer can equip physical constraints to the learning process, thus improving the network accuracy on unseen scenes.

We also applied Monte Carlo differentiable rendering on 3D object reconstruction (Chapter 6) problems. Previously, gradients could not be backpropagated through the edge discontinuities, and alternative optimization was required (i.e., optimizing normals, and then run Poisson surface reconstruction [98, 99], and repeat). We showed that with unbiased derivatives wrt. vertex positions, high-quality reconstructions can be optimized in a joint fashion for both geometry and reflectance.

A few future research directions worth exploring are as follows:

**Scene representation.** Due to recent progress in neural rendering, especially its promising fit for AR/VR applications, research on different scene representations deserves more thorough exploration. The triangle mesh has been the dominant geometry format in traditional computer graphics. Hierarchical data structures like the kd tree enable fast ray intersections for rendering. How-
ever, meshes have less flexibility in handling topology changes during optimization, compared to volume opacity fields (e.g., NeRF) or signed distance functions (SDFs). On the other hand, while recent neural representations have shown promising reconstruction results, they are not well supported in graphics pipelines yet. It would be interesting to explore a hybrid representation, with both the flexibility for object capture and scanning, while at the same time remaining efficient in raytracing.

**Physical priors in deep learning.** When we understand the physical laws behind the problem, it would be preferred to encode such prior knowledge into the design of the learning pipeline. This has been shown to be useful, especially for the cases where training data is limited. For example, a rendering layer is usually included in recovering SVBRDF parameters from mobile captures [36, 124, 49, 63]. It would be beneficial to explore more along this direction, and extend these physical priors to more challenging problems in computer graphics and vision.

**Adaptive sampling and reconstruction.** It has been shown that differentiable rendering is powerful for inverse problems, while little has been explored in making use of gradients from the differentiable rendering for other forward problems. H2MC [120] and our LMC [136] have demonstrated it by accelerating MCMC rendering with local adaptations, while more in this space can be further explored. For instance, would it be helpful to improve the design of Monte Carlo denoising architecture, or the global exploration of MLT algorithm, when path derivatives are available?
In this appendix, we present our fitting pipeline for procedural yarn models, and extend it to on-the-fly rendering for very large textiles with fiber-level details.

A fabrics’s appearance depends heavily on how the yarns and fibers are arranged. We model cloth micro geometry with procedural yarn models, which can describe different type of yarns with fiber-level details (both regular fibers and flyaway fibers) with only 22 floating numbers. We fit our procedural model to micro CT data with the proposed automatically fitting pipeline and validate our results with both CT imaging and photographs of real-world yarns. We further extend this framework for rendering very large textiles by generating yarns on the fly, addressing memory limitations of fully realizing the entire textiles. This two projects have been published at ACM SIGGRAPH 2016 [258] and EGSR 2017 [135], respectively.

A.1 Introduction

Fabrics are essential to our daily lives. Designing and modeling them virtually is important for many applications such as online retail, textile design, and in entertainment applications like games and movies. Despite a rich history in computer graphics, accurately modeling fabrics remains challenging; fabrics are structurally and optically very complex, resulting in dramatic variation in their appearance.

Representing the richness of fabrics in virtual models has actively driven re-
Figure A.1: We present a new technique to automatically generate procedural representations of yarn geometry. Based on geometric measurements of physical yarn samples (a), our approach fits statistical representations of fiber geometry that closely match reality (b). The four yarns in (a, b) from top to bottom are cotton, rayon, silk, and polyester. Our fitted models can populate realistic fiber-level details into yarn-based fabric models to significantly improve the quality of the rendered fabrics (c-top vs. c-middle (ours)). Our procedural models carry high-level synthetic information (e.g., twisting and hairiness) which offers easy editability (c-bottom).

search in a variety of fabric appearance models. Recently, *micro-appearance based* fabric models [256, 257, 102, 208] that explicitly capture a fabric’s micro-scale geometry have been introduced. Compared to traditional methods, these models describe fabrics at unprecedented detail, and thus offer superior generality as the complex appearance of a fabric is largely a direct consequence of its small-scale structures.

Since fiber-level details greatly affect large-scale fabric appearance, accurately capturing them is critical for predictive applications such as textile design. Previously, these detailed structures were measured through volume imaging (e.g., computed microtomography), leading to high-resolution 3D volumes [256, 257] or large numbers of unorganized fiber curves [102]. Neither of these representations is easy to manipulate. On the other hand are yarn-based models [91, 33] which are much more amenable to editing and simulation [251],
but do not capture the richly detailed fiber-level structures that are needed to achieve realism. See Figure A.1-c (top), where the yarns appear to be made of plastic because they miss many detailed highlights generated by fiber-level structures. Bridging the gap between convenience and easy modeling on the one hand, and complex appearance on the other, is critical for the adoption of richly detailed fabric models.

We introduce an automated modeling process which takes physical CT measurements (Figure A.1-a) and computes procedural descriptions for fabric yarns (Figure A.1-b). The output can be used to populate realistic fiber-level details into yarn-based models (Figure A.1-c, middle), significantly improving the realism of the final output. Further, the parameterized models are easy to edit, producing significant variations of appearance that match the real-world behavior of yarns (Figure A.1-c, bottom).

Micro-geometry of fabrics has been studied extensively in textile research. Recently, Schröder et al. [208] introduced a procedural yarn model to computer graphics leveraging state-of-the-art results from this field [149, 222, 100]. Their model represents fabric yarns based on statistical distributions which can be edited easily. However, how to set parameter values for this model to closely match real fabrics remains a significant challenge. In [208], creating procedural yarns was a manual process. But this fitting can be tricky. As demonstrated in Figure A.2, changes in fiber-level properties, from twisting to level of hairiness, can greatly affect a fabric’s overall appearance. Thus, properly fitting these parameters is essential to correctly reproduce the visual characteristics of real-world fabrics. But this is very challenging to do manually as fiber-level structures are too small to observe, and setting corresponding parameters properly
requires careful analysis of 3D micro-geometries that are difficult to obtain without volumetric measurements.

We present a novel technique to create procedural yarn models automatically based on physical measurements of real-world yarns. Our contributions include:

- An end-to-end pipeline to automatically build our model from physical measurements acquired using micro CT imaging.
- An improved flyaway fiber model which is not only statistically more meaningful, but also easier to fit to.
- Validation of our fitted results by comparing with CT measurements as well as photographs.

Our tool and data allow fiber-level details to be added to yarn-based cloth designs produced by textile software [182] and simulation tools [91, 33]. The added details can lead to significantly higher rendering quality and greatly benefit the design applications.

A.2 Overview

We discuss prior work in fabric micro-appearance modeling techniques and procedural yarn models.
Figure A.2: **Rendered fabrics** with identical yarn-level geometry but varying fiber-level properties: (a) original; (b) modified fiber twisting; (c) modified fiber distribution; (d) increased flyaway variance. These fiber-level variations change appearance at much larger scales.

### A.2.1 Fabric Micro-Appearance Modeling

Fabrics are constructed by combining multiple yarns via manufacturing technologies such as weaving and knitting. Each yarn, in turn, is created by twisting hundreds of micron-diameter fibers. Real-world yarns usually consist of multiple sub-strands or *plies* (Figure A.3). Recent fiber-based models achieve visual accuracy by representing the real structure of yarns and plies explicitly. These models can be classified into two major categories: volumetric and fiber-based.
Figure A.3: **Fabric structure:** a yarn normally consists of multiple sub-stands called plies. Each ply in turn contains tens to hundreds of micron-diameter fibers.

**Volumetric models**  Volumetric models describe a fabric’s micro-geometry using high-resolution 3D volumes obtained by processing micro CT measurements [256]. At each point in the volume, local fiber density and orientation are stored.

**Fiber-based models.** Fiber-based models treat fabrics as collections of individual fibers described as 1D curves [102, 208]. These fiber curves are then combined with bidirectional curve scattering distribution functions (BCSDFs) to generate rendered images.

Compared to volumetric models, fiber-based models offer a similar level of realism [102] while being more compact and providing more synthetic information. Therefore, in this work, we choose to generate fiber-based representations procedurally.

### A.2.2 Procedural Modeling of Yarn Geometry

In this work, we build on a procedural yarn model proposed by Schröder et al. [208] based on state-of-the-art results from textile research [149,
Algorithm 1: procedural yarn generation

1: for each ply $i$ do
2: generate regular fibers using (A.1), (A.2), (A.3)
3: add flyaway fibers
4: scale all fibers in the ply to obtain elliptical cross-sections
5: end for
6: twist all plies together around the yarn center

222, 100]. This model statistically describes how individual yarns are formed by underlying fibers. The key parameters are:

- For fibers: cross-sectional fiber distribution, fiber twisting, and fiber migration.
- For plies: ply cross section, ply twisting, and fiber count.

Additionally, this model has a separated step, which we improve upon in §A.4, that handles the important-for-realism effect of flyaway fibers. See [208] for more details.

**Cross-sectional fiber distribution.** A key component of the procedural yarn model is a cross-sectional fiber distribution that captures the likelihood of a fiber’s existence given its distance $R \in [0, 1)$ from the ply center. This distribution uses the following (unnormalized) density function:

$$p(R) = (1 - 2\epsilon)\left(\frac{e - e^R}{e - 1}\right)^\beta + \epsilon,$$

(A.1)

which is used with rejection sampling (Algorithm 2) to draw cross-sectional fiber locations. Given a sampled location for the $i$-th fiber $(x_i, y_i)$, the fiber curve
(as a circular helix parameterized by \( \theta \)) can be generated as follows, assuming the ply center to be the Z-axis:

\[
x(\theta) = R_i \cos(\theta + \theta_i), \quad y(\theta) = R_i \sin(\theta + \theta_i), \quad z(\theta) = \frac{\alpha \theta}{2\pi},
\]

where \( R_i := \|(x_i, y_i)\| \), \( \theta_i := \text{atan2}(y_i, x_i) \), and \( \alpha \) is a constant determining the fiber’s twist (i.e., the helix’s pitch).

**Fiber migration.** In (A.2), the distance between a generated fiber and the ply center stays constant. But this is unnatural: fibers typically migrate from such a fixed distance. This *fiber migration* is modeled by allowing the distance to change continuously between two given constants \( R_{\min} \) and \( R_{\max} \). That is, by replacing \( R_i \) in (A.2) with:

\[
R_i(\theta) := R_{\min}R_i + \frac{(R_{\max} - R_{\min})R_i}{2} [\cos(s\theta + \theta_i^{(0)}) + 1],
\]

where \( s \) is a constant controlling the length of a rotation, and \( \theta_i^{(0)} \) is a per-fiber parameter indicating the rotation’s initial “phase”.

**Ply cross section.** Plies generated with (A.1) and (A.3) always have circular cross-sections. A simple generalization is to support elliptical cross-sections by scaling a ply along the X- and Y-directions by factors of \( e_X \) and \( e_Y \) respectively.

**Flyaway fibers.** Real yarns usually contain *flyaway fibers* that do not follow the flow of normal fibers.

These irregularities not only contribute greatly to the realism of yarn appearance, but are also crucial for reproducing fuzzy silhouettes of real-world fabrics.
Previous work generated flyaway fibers in a somewhat simplified manner using 3D perlin noise [208].

We introduce an improved flyaway model which is statistically more meaningful and easier to fit to.

**Ply twisting.** The final step to build a procedural yarn model is twisting the component plies.

For a yarn centered at the Z-axis, each ply is twisted to follow a circularly helical curve

\[ S(z) := (S_x(z), S_y(z), z) \]

with its pitch controlled by \( \alpha_{\text{ply}} \):

\[
S_x(z) = R_{\text{ply}} \cos(2\pi z/\alpha_{\text{ply}} + \theta_{\text{ply}}), \\
S_y(z) = R_{\text{ply}} \sin(2\pi z/\alpha_{\text{ply}} + \theta_{\text{ply}}).
\] (A.4)

Besides the cross section and twisting information, each ply has an integer \( m \) associated with it that specifies the number of component fibers. The entire pipeline for procedural generation of yarn geometry is summarized in Algorithm 1.

**Challenges in parameter fitting.** Although the fiber generation process (Algorithm 1) is straight-forward given the model parameters, its inverse problem of parameter fitting is far from trivial. Many fiber-level parameters, such as cross-sectional fiber distribution and fiber migration, are based on the statistical properties of a yarn’s micro-geometry. These properties are difficult to acquire
due to their small scale, and challenging to fit to because of their naturally existing irregularities.

We address these challenges by acquiring a yarn’s micro-geometry using micro CT imaging, and introducing an end-to-end pipeline to automatically and robustly fit procedural representations to the measured data. In addition, we introduce a new model for capturing flyaway fibers which contributes significantly to photorealism.

A.3 Model Fitting

We present our end-to-end pipeline for fitting procedural yarns to physical measurements of micro-geometries. The parameters needed to be fit are summarized in Table A.1. The challenge is converting volumetric CT data with no synthetic information to a procedural yarn model with correct ply-level and fiber-level properties. Many parameters must be identified properly to match the appear-
Our parameter fitting pipeline. This pipeline is approximately in the opposite order of the procedural yarn generation algorithm (Algorithm 1) since it is fitting parameters.

Our pipeline, from a high level, is analogous to executing Algorithm 1 in reverse order (Figure A.4). In particular, given measured yarn micro-geometry (§A.3.1):

- First, we estimate ply twisting as well as the component plies’ cross-sectional shapes (§A.3.2). Using this information, we ‘untie’ and deform all component plies so that they are centered on the Z-axis and have circular cross-sections.
- Then, we analyze each untied ply and classify the constituent fibers into two categories: flyaway and regular (§A.3.3).
- Next, based on the obtained flyaway fibers, we determine the parameter values required by our improved model (§A.4.2).
- Lastly, we fit the cross-sectional fiber distribution and migration parameters using the previously obtained regular fibers (§A.3.4).

In the rest of this section, we first describe our CT imaging process stage leveraging state-of-the-art techniques [257, 102] to obtain clean input geometry.
Table A.1: **List of parameters** used by the procedural yarn model. Parameters shown in purple are per-ply attributes. Those shown in blue are specified for every fiber by independently sampling the corresponding statistical distributions. In particular, per-fiber flyaway parameters (i.e., $R_{\text{loop}}^{\text{max}}$, $R_{\text{hair}}^{\text{min}}$, $R_{\text{hair}}^{\text{span}}$, and $\theta_{\text{hair}}^{\text{min}}$, $\theta_{\text{hair}}^{\text{span}}$) are sampled from normal distributions (see §A.4 and Table A.3) while others (i.e., $R_i$, $\theta_i$, and $\theta_i^{(0)}$) are drawn uniformly (§A.2.2).

to our main fitting pipeline (§A.3.1). Then, we explain our pipeline following the flow of Algorithm 1 (i.e., from right to left in Figure A.4).

### A.3.1 Input

We acquire micro geometries of physical yarns using micro CT imaging. As shown in Figure A.5-a, multiple yarns are packed together for faster acquisition since they can be scanned simultaneously. Given micro CT measurements with volumetric fiber densities (Figure A.5-b), we process them using the approach developed by Khungurn et al. [102] to extract fiber curves (Figure A.5-c).
addition, we perform yarn tracking [257] to extract the center curve for each component ply (Figure A.5-d). The recovered fiber curves and ply centers (Figure A.5-cd) act as input to our main parameter fitting pipeline.

A.3.2 Ply Twisting and Cross Section Estimation

The first step of our parameter fitting pipeline is to recover the input yarn’s plying (i.e., ply twisting and cross-sectional shape) so that we can separate individual plies and perform later per-ply analysis. In this paper, we assume all plies in a yarn are identical and evenly distributed around the yarn center.

**Ply twisting.** For each given ply center \(i\) (represented as a polyline), we fit a helical curve \(S_i\) in the form of (A.4) which requires estimating the ply radius \(R_i^{\text{ply}}\), the pitch \(\alpha_i^{\text{ply}}\), and initial angle \(\theta_i^{\text{ply}}\). Given our assumption of identical and evenly distributed plies, this boils down to finding:
• One set of $R^\text{ply}$ and $\alpha^\text{ply}$ shared by all piles;

• $\theta_1^\text{ply}$, the initial angle of the first ply, which can be used to determine those of all other plies by setting $\theta_i^\text{ply} = \theta_1^\text{ply} + 2\pi \frac{i-1}{K}$.

To determine these parameters, we optimize the L2 distance between the given and fitted ply centers by minimizing:

$$E^\text{ply}(R^\text{ply}, \alpha^\text{ply}, \theta_1^\text{ply}) = \sum_{i=1}^{K} \int_{z_0}^{z_1} \| S_i(z \mid R^\text{ply}, \alpha^\text{ply}, \theta_1^\text{ply}) - S^\text{tracked}_i(z) \|_2^2 \, dz,$$

where $K$ is the number of plies, $S_i$ is given by (A.4), and $S^\text{tracked}_i$ is the $i$-th input ply center (Figure A.5-d) whose two endpoints give $z_0$ and $z_1$.

To minimize (A.5), we use an open source implementation [77] of the Nelder-Mead Simplex Method [160]. One could also leverage more advanced optimization methods such as the floating tangent algorithm [35], possibly leading to better accuracy and performance. However, since our acquisition setup keeps yarn samples straight (Figure A.5-a), the input curves (i.e., $S^\text{tracked}_i$) are already close to circular helices. We found that our simple approach was fast, well-behaved (i.e., there were no convergence issues), and produced high-quality results for rendering purposes.

**Cross-section estimation.** As in [208], we model the cross-sections of the plies as ellipses that are allowed to intersect, to mimic ply compression. Given a cross-sectional plane, we place an ellipse at the center of each ply with its short axis pointing towards the center of a polygon formed by all the ply centers (see Figure A.6 for a 3-ply example). These ellipses are used to determine the lengths
of the ellipse axes (i.e., $e_X$ and $e_Y$). We assume identical plies, and therefore obtain one set of $e_X$ and $e_Y$ values based on information from all plies. In particular, we rotate and translate each ply (with all its constituent fiber centers) in the plane, making it axis-aligned and centered at the origin (Figure A.6, right). By stacking all transformed plies from all cross-sectional planes,\(^1\) we obtain an accumulated 2D point cloud consisting of fiber centers (Figure A.7). Then, we set $e_X$ and $e_Y$ to twice the standard deviation of the X- and Y-coordinates of all these points, respectively. The resulting ellipse covers approximately 95% of the fiber centers. We rely on our flyaway fiber model to capture the remaining, mostly irregular, fibers.

Besides ply twisting and cross-section parameters, we estimate the number of fibers per ply as $m = \lfloor L_{\text{total}}/(L_{\text{ply}} K) \rfloor$, where $L_{\text{total}}$ indicates the total length of fibers in the measured geometry (Figure A.5-c), $L_{\text{ply}}$ denotes the length of a fitted ply center (all $S_i$ curves have identical lengths), and $K$ is the number of plies.

### A.3.3 Fiber Classification

Using the plying information obtained in §A.3.2, we untie the plies so that they are all centered around the Z-axis. All fibers in these piles then form a ‘fiber soup’ which will be analyzed by the following steps of our pipeline. In this step, we classify these fibers into regular versus flyaway.

To classify each fiber into one of the two categories, we consider its minimal and maximal distances denoted as $d_{\text{min}}$ and $d_{\text{max}}$ to the ply center (i.e., Z-axis).

\(^1\)In practice, we take around 1000 cross-sectional planes for each yarn, similar to the resolution of micro CT measurements.
Figure A.6: **Fitting ply cross-sections** (3-ply example). (left) We place an ellipse at each ply center. The short axis of this ellipse points toward the center of a polygon (shown in gray with dashed boundaries) formed by all ply centers. (right) We then rotate and translate all plies (in plane) with the corresponding fiber points. These transformed 2D fiber locations will then be used to fit the ply cross-sections (Figure A.7).

Given a fiber with \( n \) vertices \((x_1, y_1, z_1), \ldots, (x_n, y_n, z_n)\), we have

\[
\begin{align*}
    d_{\text{min}} &:= \min_{1 \leq i \leq n} \| (x_i, y_i) \|_2, &
    d_{\text{max}} &:= \max_{1 \leq i \leq n} \| (x_i, y_i) \|_2.
\end{align*}
\]

CT measurements of real yarns usually contain alien components (e.g., dust) that do not belong to the yarn. They can be distinguished by large \( d_{\text{min}} \) values since alien materials tend to stay far from the ply center. In practice, we consider all fibers with \( d_{\text{min}} \) beyond some threshold as alien and simply ignore them. We then categorize the remaining fibers into regular and flyaway based on their \( d_{\text{max}} \) values. Intuitively, fibers with small \( d_{\text{max}} \) values stay close to the ply center and are likely to be regular. On the other hand, those with large \( d_{\text{max}} \) values are at some point far from the center. Thus, they are considered flyaway.

To robustly obtain the thresholds for respectively identifying alien and flyaway fibers, we consider the means (denoted as \( \mu_{\text{min}}, \mu_{\text{max}} \)) and standard deviations (denoted as \( \sigma_{\text{min}}, \sigma_{\text{max}} \)) of all \( d_{\text{min}} \) and \( d_{\text{max}} \) values respectively. Precisely, we
Figure A.7: **Fitting ply cross-sections (cont’d).** A density visualization of transformed fiber centers (Figure A.6) from a sequence of cross-sectional planes. We use this density information to determine the shape of ply cross-sections.

 treat all fibers satisfying \( d_{\text{min}} > \mu_{\text{min}} + c_{\text{min}} \sigma_{\text{min}} \) as alien, and the remaining ones with

\[
d_{\text{max}} > \mu_{\text{max}} + c_{\text{max}} \sigma_{\text{max}}
\]

(A.6)

as flyaway (where \( c_{\text{min}} \) and \( c_{\text{max}} \) are user-specified constants). In practice, we use \( c_{\text{min}} = c_{\text{max}} = 2 \) and perform fiber classification separately for each (untied) ply. Figure A.8 shows an example classification.

### A.3.4 Fitting Fiber Distribution and Migration

Given the set of regular fibers obtained in §A.3.3, we now present the last step of our pipeline which fits the fiber distribution (A.1) and migration (A.3) parameters. Recall that all these fibers belong to untied plies, meaning that they are all centered around the Z-axis.

In theory, fiber migration parameters \( R_{\text{min}}, R_{\text{max}}, \) and \( s \), can be recovered from
Figure A.8: **Fiber classification**: we classify all input fibers into two categories: regular and flyaway (with alien ones removed). (a) Scatter plot of \((d_{\text{min}}, d_{\text{max}})\) for each fiber. (b, c, d) 2D fiber visualizations where the horizontal and vertical axes show \(z_i\) and \(\| (x_i, y_i) \|_2\) for each fiber vertex, respectively. The dashed line indicates the threshold beyond which a fiber is considered flyaway.

a single regular fiber. However, due to irregularities in real data, using only one fiber generally yields highly unreliable results. Furthermore, many fibers in the input geometry are short due to limitations of the CT imaging processing step [102], making migration parameter estimation even more challenging.

We tackle this problem by minimizing a reconstruction error defined as:

\[
E_{\text{mig}}(R_{\text{min}}, R_{\text{max}}, s) := \sum_{i} \min_{\{ R_i, \theta_i^{(0)} \}} E_i \left( R_i, \theta_i^{(0)} \left| R_{\text{min}}, R_{\text{max}}, s \right. \right),
\]

where the summation is over all regular fibers, and \(E_i\) indicates the squared L2 difference between fiber \(i\) (represented as a polyline) and the helix generated...
with $R_i, \theta_i^{(0)}, R_{\min}, R_{\max},$ and $s$ via (A.1) and (A.3). Namely,

$$E_i(R_i, \theta_i^{(0)} \mid R_{\min}, R_{\max}, s) := \int_z \|F_i(z) - \tilde{F}_i(z)\|^2 \, dz,$$

(A.8)

where $F_i$ and $\tilde{F}_i$ respectively denote the input and generated fibers, both of which are parameterized by $z$. The limits of this 1D integral are given by the Z-coordinates of $F_i$’s two endpoints. Then, we set $(R^*_{\min}, R^*_{\max}, s^*) = \arg\min E_{\text{mig}}(R_{\min}, R_{\max}, s)$.

Minimizing this reconstruction error (A.7), however, is non-trivial since the error metric itself includes minimization over $R_i$ and $\theta_i^{(0)}$. Thus, we enumerate a densely sampled set of $R_{\min}, R_{\max},$ and $s$ values. For each combination, we solve the inner optimization problem (i.e., RHS of (A.7)). Similar to minimizing (A.5), we found it easy to find optimizers for this inner problem and used the same implementation.

After determining the fiber migration parameters, only the cross-sectional fiber distribution remains unknown. Let $R^*_i$ and $\theta_i^{(0)*}$ be the minimizers of $E_i(R_i, \theta_i^{(0)} \mid R^*_{\min}, R^*_{\max}, s^*)$ for each fiber $i$. We then apply Maximum-Likelihood Estimation (MLE) over all $R^*_i$ values to obtain the fiber distribution parameters $\epsilon$ and $\beta$. Notice that the density function (A.1) is essentially conditional since it is used within a rejection sampling framework (Algorithm 2) where $R$ is not uniformly distributed. The unconditional density for the MLE should be:

$$p_{\text{MLE}}(R) = 2R \, p(R) = 2R \left[ (1 - 2\epsilon) \left( \frac{e^{-e^{R^*}}}{e - 1} \right)^\beta + \epsilon \right].$$

(A.9)

Let $q(\epsilon, \beta)$ be the normalization term for (A.9)

$$q(\epsilon, \beta) := \int_0^1 p_{\text{MLE}}(R) \, dR,$$

(A.10)

we have the normalized density function $p_{\text{norm}}(R \mid \epsilon, \beta) := p_{\text{MLE}}(R)/q(\epsilon, \beta)$, which can be used in MLE. In practice, we use Matlab’s mle() function with $q$ evalu-
ated numerically which can sometimes be prone to (numerically similar) local optima. However, this has caused no visual difference in our experiments.

At this point, we have obtained a set of ply-based and fiber-based parameters which can be used to procedurally generate regular fiber curves. With only regular fibers, however, yarns look unrealistic because they lack irregularities. In the next section, we present our improved flyaway fiber model in §A.4.1, and in §A.4.2, we describe how to fit the model parameters given real flyaway fibers obtained in §A.3.3.

A.4 Our Flyaway Fiber Model

Natural cloth yarns normally contain flyaway fibers which deviate from most other fibers, causing great irregularity in a yarn’s micro-geometry. Although there are much fewer flyaways fibers comparing to regular ones, they contribute significantly to a fabric’s fuzziness. Thus, to accurately reproduce the appearance and structure of real-world fabrics, these fibers need to be properly modeled.

A.4.1 Model Description

Flyaway fibers have been studied by textile researchers. In particular, Voborova et al. [235] categorize flyaway fibers into five types. Inspired by their work, we introduce an improved flyaway fiber model over the one developed by Schröder et al. [208] based on Perlin noise.
Our model classifies flyaway fibers into two categories: *loop* and *hair* (Figure A.9). Our hair-type fibers correspond to the first two flyaway classes in [235], while our loop-type to their third class.

**Loop-type.** Loop-type flyaway fibers have both endpoints inside the main ply body. Each of these fibers was originally regular but part of it has been accidentally pulled out during the manufacturing process (Figure A.9-a). The density of these fibers and the distance each loop deviates from the ply center provide an important visual cue on how tight the fibers are twisted.

Schröder et al. [208] do not explicitly model these type of fibers. Instead, they use very strong fiber migrations to achieve similar effects. Unfortunately, this approach cannot easily capture occasionally occurring loop-type fibers, which

![Figure A.9: The two types of flyaway fibers in our model.](image)

![Figure A.10: Comparison of loop-type flyaway fiber models:](image)

(a) Migration-based  
(b) Ours

**Comparison of loop-type flyaway fiber models:** (a) the migration-based approach [208] lacks the generality to represent occasionally occurring loops; (b) our model has sufficient representative power to capture such fiber geometry.
Figure A.11: Generation of loop-type flyaway fibers. (a) We randomly pick a vertex maximizing the distance to the ply center (indicated with an orange circle). (b) We then select all vertices belonging to the same migration cycle (shown in green) and scale their radii so that the selected vertex has radius $R_{\text{loop max}}$ afterwards.

are the common case in reality. The resulting geometry normally contains regularly appearing loops (Figure A.10-a).

We generate loop-type flyaway fibers by modifying the regular ones built previously (Line 2 of Algorithm 1). When turning a regular fiber into a loop-type flyaway, we randomly pick one of its vertices with (A.3) maximized and raise its radius to $R_{\text{loop max}}$ (by scaling its X- and Y- coordinates). We also scale up the radii of neighboring vertices belonging to the same migration period (i.e., one period of $R_i(\theta)$ in (A.3), see Figure A.11). To create a set of loop-type flyaway fibers (based on existing regular ones), we draw $R_{\text{loop max}}$ from a normal distribution for each of them.

For each ply, we use $\rho_{\text{loop}}$ to capture the density of loop-type fibers. Given an untied ply centered around $Z_0Z_1$ with $Z_0 = (0,0,z_0)$ and $Z_1 = (0,0,z_1)$, we repeat the aforementioned process to generate $[\rho_{\text{loop}}(z_1 - z_0)]$ loop-type flyaway fibers.
Figure A.12: **Comparison of flyaway fiber models (hair-type):** (a) measured yarn geometry (ground truth); (b) procedural yarn with no flyaway fibers; (c) procedural yarn with Perlin noise-based flyaways [208]; (d) procedural with flyaways generated with our model. Fiber migration is increased in (c) for simulating loop-type effects.

Figure A.13: **Simplifications** made by our hair-type flyaway fiber model: (top) our hair-type flyaways all have arc shapes; (bottom) real hair-type fibers have more complex trajectories.
Figure A.14: **Processing Flyaway Fiber.** (a) One fiber can contain both hair and loop components. (b) We split each flyaway fiber into multiple hair and loop segments.

**Hair-type.** As shown in Figure A.9-b, each hair-type flyaway fiber has one endpoint outside the body of its hosting ply. This type of fiber contributes most significantly to a yarn’s hairy appearance. We create a hair-type fiber (for Line 3 of Algorithm 1) by adding its visible (flyaway) part explicitly. That is, we generate an ‘arc’ determined by its starting and ending radii \( R_{\text{min}}^{\text{hair}}, R_{\text{max}}^{\text{hair}} \) (i.e., distance to the ply center), azimuthal angles \( \theta_{\text{min}}^{\text{hair}}, \theta_{\text{max}}^{\text{hair}} \), as well as locations \( z_{\text{min}}^{\text{hair}}, z_{\text{max}}^{\text{hair}} \). Let \( R_{\text{span}}^{\text{hair}} := R_{\text{max}}^{\text{hair}} - R_{\text{min}}^{\text{hair}}, \theta_{\text{span}}^{\text{hair}} := \theta_{\text{max}}^{\text{hair}} - \theta_{\text{min}}^{\text{hair}}, \) and \( z_{\text{span}}^{\text{hair}} := z_{\text{max}}^{\text{hair}} - z_{\text{min}}^{\text{hair}} \), a hair-type flyaway fiber parameterized by \( t \in [0, 1] \) can be represented as

\[
\begin{align*}
    x^{\text{hair}}(t) &= R^{\text{hair}}(t) \cos \left( \theta^{\text{hair}}(t) \right), \\
    y^{\text{hair}}(t) &= R^{\text{hair}}(t) \sin \left( \theta^{\text{hair}}(t) \right), \\
    z^{\text{hair}}(t) &= z_{\min}^{\text{hair}} + z_{\span}^{\text{hair}} t,
\end{align*}
\]

with \( R^{\text{hair}}(t) = R_{\min}^{\text{hair}} + R_{\span}^{\text{hair}} t \) and \( \theta^{\text{hair}}(t) = \theta_{\min}^{\text{hair}} + \theta_{\span}^{\text{hair}} t \).

To generate a set of hair-type fibers around a common center with endpoints \( z_0 \) and \( z_1 \), we draw \( z_{\min}^{\text{hair}} \) from \( U[z_0, z_1] \) and \( \theta_{\min}^{\text{hair}} \) from \( U[0, 2\pi] \) for each of them. The other parameters \( R_{\min}^{\text{hair}}, R_{\span}^{\text{hair}}, \theta_{\span}^{\text{hair}}, \) and \( z_{\span}^{\text{hair}} \) are sampled from separate normal distributions. Similar to loop-type, we use \( \rho^{\text{hair}} \) to describe the density of hair-
type fibers.

In [208], hair-type fibers are generated by randomly adding short curves with tangent directions disturbed by 3D Perlin noise (Figure A.12-c). Although able to generate hair-like fibers, the oversimplified nature of this approach makes it tricky to match reality and, more importantly, difficult to fit to given measured yarn geometries. Our model, on the other hand, carries clear statistical meaning and is easy to fit to (§A.4.2).

**Fiber perturbations.** Until this point, all regular and flyaway fibers were perfectly smooth curves. This level of smoothness rarely exists in reality: real fibers generally contain many small but irregular turns. To mimic this effect, we add small perturbations to all fibers by randomly scaling the radius (i.e., X- and Y-coordinates) of each vertex.

**Discussion.** Our flyaway fiber models make two notable simplifications. In particular, our loop-type flyaway fibers always “span” exactly one migration cycle (as demonstrated in Figure A.11), which is generally not the case in reality. Moreover, our hair-type flyaways all have “arc” shapes due to (A.11), while real hair-type fibers normally have more complex trajectories (Figure A.13). Despite these simplifications we match overall yarn appearance well (see Figure A.15). They are introduced to ensure that our model can be reliably fitted from measurements containing limited number of flyaway fibers (e.g., Figure A.5-b). In the future, with greater amount of measurements, flyaway fiber models based on more complex statistical distributions could be designed.
Table A.2: Fiber scattering parameters using the model introduced by Khungurn et al. [102] for generating all renderings in Figure A.15. We manually adjusted the parameter values since finding optimal ones automatically is orthogonal to our current work. For better appearance matching, we recommended using advanced inverse rendering techniques such as [102].

A.4.2 Model Fitting

Recall that in A.3.3, fibers from untied plies are classified as regular and fly-away. Given our flyaway fiber model (A.4.1), we now describe how to fit model parameters, i.e., \( \rho_{\text{hair}}, R_{\text{min}}^{\text{hair}}, R_{\text{span}}^{\text{hair}}, z_{\text{span}}^{\text{hair}}, \) and \( \theta_{\text{span}}^{\text{hair}} \) for hair-type ones, as well as \( \rho_{\text{loop}} \) and \( R_{\text{max}}^{\text{loop}} \) for loop-type ones. Notice that \( z_{\text{min}}^{\text{hair}} \) and \( \rho_{\text{min}}^{\text{hair}} \) are not fitted as we sampled them uniformly in the generation process.

After obtaining flyaway fibers for each untied ply, we start the fitting process by identifying the loop and hair components of each fiber. As illustrated in Figure A.14-a, one flyaway fiber can contain both components. Thus, for each of them, we start by checking if any of its endpoints has a radius (i.e., \( \| (x, y) \|_2 \)) above the flyaway threshold (A.6). If so, we cut off the corresponding end of the fiber at vertices where the radius is both at a local minimum and below the threshold and add it to the collection of hair-type flyaway fibers. The remaining
part (if any) is then added to the set of loop-type fibers. This fiber-splitting operation is illustrated in Figure A.14-b.

After all fibers have been processed, we obtain the flyaway densities $\rho_{\text{loop}}$ and $\rho_{\text{hair}}$ using the number of corresponding fibers divided by the length of the untied ply (i.e., $z_1 - z_0$). Then, for every loop-type fiber, we compute the radii of all its vertices and set $R_{\text{loop}}^{\max}$ as the maximum. Similarly, $R_{\text{hair}}^{\min}$, $R_{\text{hair}}^{\text{span}}$, $\theta_{\text{hair}}^{\text{span}}$, $z_{\text{hair}}^{\text{span}}$ can be evaluated for each hair-type fiber by examining its vertices. We then compute the sample mean and standard deviation for each of these quantities, which can be used to generate hair-type fibers via (A.11).

With flyaway fiber parameters fitted, we finally have recovered the values of all parameters in Table A.1. Using Algorithm 1, realistic yarn geometries can now be generated without the need for cumbersome CT measurement.

A.5 Results

We measured the micro-geometries of nine real-world yarns through three micro CT scans. These yarns are made from cotton, rayon, silk, and polyester, a good sampling of commonly used yarn materials. Our acquisition setup is shown in Figure A.5-a. All yarns were scanned in an XRadia MicroXCT scanner under a resolution of $2.5 \, \mu\text{m}$, and each scan took approximately three hours. All rendered images are created using a modified version of the Mitsuba physically based renderer [82].
**Performance.** Our fitting algorithm is fast: given the processed CT data (Figure A.5-cd), it takes fewer than ten CPU core minutes to fit each yarn. Compared to the CT data processing step (A.3.1) which normally takes several core hours, our fitting cost is negligible. Our method is generally well-behaved: despite the presence of non-convex optimization (e.g., A.3.4), we did not observe any visual difference caused by local minima.

**Validation.** Figure A.15 shows all our measured yarn geometries (a) and our fitting results (b). Procedural yarns generated with our method closely capture the statistics of the measured geometries, conveying a convincing impression of the corresponding yarn types. To validate this, we compare full renderings of our models (d) with photographs of those yarns (c). To describe the optical properties of our procedurally generated yarns, we use the fiber scattering model introduced in [102].

See Table A.2 for corresponding parameter values. Notice that our models can sometimes appear slightly less hairy since our measured geometries (a), which come from small (around 0.5 cm in length) yarn segments, might not capture all irregularities the photos have. This is not a limitation of our approach though, since one can scan multiple copies of the same type of yarn for variety and feed all the information to our parameter fitting algorithm.

**Full textiles.** To describe full textiles, existing micro-appearance models usually take tens of megabytes (fiber-based) or several gigabytes (volumetric) of data. Our procedural model representation, on the other hand, is extremely compact: before instantiating fibers using Algorithm 1, only 22 numbers (see Table A.3) need to be stored for each type of yarn. By combining these distribu-
tional parameters with a sparse set of yarn center curves, highly detailed textile models can be generated. This is demonstrated in Figure A.16 where column (a) shows yarn curves that we use to populate fiber-level structures.\(^2\) Without these small-scale details, the renderings offer limited quality. With added micro-geometry, on the other hand, the textiles appear much more realistic (b). The model constructed by our approach is fully procedural, offering easy editability. This is demonstrated in column (c) of Figure A.16. For the silk scene, in particular, we adjusted fiber twisting so that the fibers are better aligned. Such a fiber-level modification not only results in significantly stronger anisotropic highlights, but also causes the fabric to become less transparent (which one can tell from the darker shadow on the ground). This is because when the fibers are better aligned, they tend to block light more effectively by having fewer ‘gaps’.

The second example in Figure A.16 shows how changing the flyaway distribution affects overall appearance of a fabric globally. By increasing the density of flyaway fibers, the entire cloth becomes significantly fuzzier, resulting in less structured highlights but more small random ones or ‘glints’. The last result in Figure A.16 is a homogeneous basket weave. We use the procedural geometry of “Polyester 1” with altered optical parameters. When making the shape of ply cross-sections more elliptical (c), the underlying fiber structure changes dramatically, which leads to reduced overall glossiness. This further demonstrates the versatility of our approach and the significance of precise fitting of model parameters.

\(^2\)We generated the yarn curves by uniformly sampling in the UV space of a given base mesh, and shifting the vertices along local normal directions following the weave pattern. In the future, one can leverage yarn-based simulation methods [91, 33] to obtain more physically accurate curves.
Figure A.15: **Parameter fitting results.** (a) CT-measured yarn geometries. (b) Procedural yarn geometries fitted using our approach. (c) Photographs of real-world yarns. (d) Full renderings of our procedural yarns under similar lighting/viewing conditions. Our fitted parameter values are shown in Table A.3.
<table>
<thead>
<tr>
<th>Yarn ID</th>
<th>Fiber-level parameters</th>
<th>Ply-level parameters</th>
<th>Flyaway fiber distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Distrb &amp; twisting</td>
<td>Mig.</td>
<td>eX</td>
</tr>
<tr>
<td></td>
<td>(\beta) (\epsilon) (\alpha) (R_{\text{min}})</td>
<td>(s)</td>
<td>(e_x)</td>
</tr>
<tr>
<td>Cotton 1</td>
<td>0.148 0.000 -0.369 0.80 1.1</td>
<td>0.026 0.020 0.038 0.453</td>
<td>75</td>
</tr>
<tr>
<td>Cotton 2</td>
<td>0.430 0.033 -0.436 0.95 1.9</td>
<td>0.025 0.018 0.033 0.336</td>
<td>37</td>
</tr>
<tr>
<td>Rayon 1</td>
<td>0.682 0.139 0.710 0.85 1.9</td>
<td>0.053 0.036 0.051 -0.639</td>
<td>150</td>
</tr>
<tr>
<td>Rayon 2</td>
<td>0.293 0.052 -0.381 0.85 1.6</td>
<td>0.028 0.021 0.029 0.372</td>
<td>80</td>
</tr>
<tr>
<td>Rayon 3</td>
<td>0.640 0.000 0.609 0.95 1.9</td>
<td>0.044 0.034 0.042 -0.638</td>
<td>143</td>
</tr>
<tr>
<td>Rayon 4</td>
<td>0.671 0.361 0.560 0.70 1.7</td>
<td>0.048 0.034 0.045 -0.498</td>
<td>122</td>
</tr>
<tr>
<td>Silk 1</td>
<td>0.462 0.000 -0.458 0.95 1.8</td>
<td>0.025 0.022 0.038 0.526</td>
<td>33</td>
</tr>
<tr>
<td>Silk 2</td>
<td>0.554 0.405 -0.556 0.70 1.2</td>
<td>0.031 0.025 0.045 0.702</td>
<td>47</td>
</tr>
<tr>
<td>Polyester</td>
<td>0.258 0.020 -0.379 0.75 1.5</td>
<td>0.029 0.021 0.029 0.370</td>
<td>60</td>
</tr>
</tbody>
</table>

Table A.3: **Our fitted parameter values** for all results shown in Figure A.15. Positive and negative \(\alpha\) values indicate counter-clockwise and clockwise (ply/fiber) twisting, respectively. We assume, without loss of generality, that \(R_{\text{max}} = 1\). For flyaway distribution parameters, paired numbers in parentheses indicate the mean and standard deviation of the corresponding normal distributions. When generating yarns using Algorithm 1, all parameters not mentioned here (e.g., \(R_i, \theta_i, \theta_i^{(0)}\)) are drawn uniformly from their domains.
Figure A.16: **Full textiles** with fiber-level geometries added procedurally using our method. The original parameters used in the three rows are: “Silk 1” + “Silk 2” (first), “Rayon 3” + “Rayon 4” (second) and “Polyester 1” (third). The edited silk example is obtained by multiplying ply and fiber twisting (i.e., $\alpha_{\text{ply}}$ and $\alpha$) by $10\times$. The modified rayon model results from tripling flyaway densities $\rho_{\text{loop}}$ and $\rho_{\text{hair}}$ as well as doubling $R_{\text{max}}^{\text{loop}}$ and $R_{\text{max}}^{\text{hair}}$. The altered polyester example is created by halving $e_Y$ (the short axis of the ellipse) while doubling $e_X$ (the long axis).
A.6 Realization-Minimizing Rendering of Procedural Textiles

We extend the above framework by introducing a novel realization-minimizing technique that enables physically-based rendering of procedural textiles, without the need of full model realizations. The key ingredients of our technique are new data structures and search algorithms that lookup regular and flyaway fibers on the fly, efficiently and consistently. Our technique works with state-of-the-art procedural models in their exact form with no approximation imposed. In practice, our method can render very large models that are practically unrenderable using existing methods, while using considerably less memory (60–200× less) and achieving good performance (see Figure A.17). For more technical details on this part, please refer to the EGSR 2017 paper [135].

A.7 Conclusion

Limitations and future work. Our fitting pipeline requires solving several non-convex optimization problems (A.3.4). In the future, advanced solution

Table A.4: Model size and rendering times for all our results. The render times are measured by rendering a 1280 pixels wide image, at 128 samples per pixel, on an Intel server with four Xeon X7560 eight-core CPUs.
techniques could be explored for better numerical accuracy and performance. Currently, rendering textiles with yarns represented by our model requires re-alizing all fibers (using Algorithm 1). We intend to develop efficient rendering algorithms that directly work on our procedural geometry. In addition, our yarn model makes some simplifying assumptions:

- Ply cross-sections are assumed to be elliptical and spatially invariant, causing small geometric mismatches for Rayon 3, Silk 1, and Silk 2 in Figure A.15-ab;
- Flyaway fibers have relatively simplified shapes (discussed in A.4.1).

Thus, further generalizations of the yarn model and developing corresponding fitting techniques will be valuable. The level of geometric variance our method can capture is limited by the maximal sample size a micro CT scanner can take. Thus, large yarn-level variations that are difficult to measure are missing from our model.

Finally, our on-the-fly technique becomes expensive when handling hair-type fibers in extremely large scenes: the pure scheme is impractically slow, while even the hybrid scheme could require too much storage. In the future, improved hair-type fibers models and/or lookup algorithms are worth exploring. In addition, level-of-detail is not yet supported by our method. Given the complexity of procedural textiles, future research along this direction can be highly beneficial.
Figure A.17: **Rendered results (woven and knitted):** Our method enables physically-based rendering of these large textiles with fiber-level details. Please see Table A.4 for data size and performance statistics.
APPENDIX B

IMAGE STYLE TRANSFER FOR PHOTOS AND PAINTINGS

In this appendix, we present our image style transfer project for photos (CVPR 2017) and paintings (EGSR 2018).

B.1 Introduction

Photographic style transfer is a long-standing problem that seeks to transfer the style of a reference style photo onto another input picture. For instance, by appropriately choosing the reference style photo, one can make the input picture look like it has been taken under a different illumination, time of day, or weather, or that it has been artistically retouched with a different intent. So far, existing techniques are either limited in the diversity of scenes or transfers that they can handle or in the faithfulness of the stylistic match they achieve. In this paper, we introduce a deep-learning approach to photographic style transfer that is at the same time broad and faithful, i.e., it handles a large variety of image content while accurately transferring the reference style. Our approach builds upon the recent work on Neural Style transfer by Gatys et al. [52]. However, as shown in Figure B.1, even when the input and reference style images are photographs, the output still looks like a painting, e.g., straight edges become wiggly and regular textures wavy. One of our contributions is to remove these painting-like effects by preventing spatial distortion and constraining the transfer operation to happen only in color space. We achieve this goal with a transformation model that is locally affine in colorspace, which we express as a custom fully differentiable energy term inspired by the Matting Laplacian [113]. We show that this approach successfully suppresses distortion while having a
Figure B.1: Given a reference style image (a) and an input image (b), we seek to create an output image of the same scene as the input, but with the style of the reference image. The Neural Style algorithm [52] (c) successfully transfers colors, but also introduces distortions that make the output look like a painting, which is undesirable in the context of photo style transfer. In comparison, our result (d) transfers the color of the reference style image equally well while preserving the photorealism of the output. On the right (e), we show 3 insets of (b), (c), and (d) (in that order). Zoom in to compare results.

Our other key contribution is a solution to the challenge posed by the difference in content between the input and reference images, which could result in undesirable transfers between unrelated content. For example, consider an image with less sky visible in the input image; a transfer that ignores the difference in context between style and input may cause the style of the sky to “spill over” the rest of the picture. We show how to address this issue using semantic segmentation [31] of the input and reference images. We demonstrate the effectiveness of our approach with satisfying photorealistic style transfers for a broad variety of scenarios including transfer of the time of day, weather, season, and artistic edits.
B.2 Method

Our algorithm takes two images: an input image which is usually an ordinary photograph and a stylized and retouched reference image, the reference style image. We seek to transfer the style of the reference to the input while keeping the result photorealistic. Our approach augments the Neural Style algorithm [52] by introducing two core ideas.

• We propose a photorealism regularization term in the objective function during the optimization, constraining the reconstructed image to be represented by locally affine color transformations of the input to prevent distortions.

• We introduce an optional guidance to the style transfer process based on semantic segmentation of the inputs (similar to [27]) to avoid the content-mismatch problem, which greatly improves the photorealism of the results.

Background. For completeness, we summarize the Neural Style algorithm by Gatys et al. [52] that transfers the reference style image $S$ onto the input image $I$ to produce an output image $O$ by minimizing the objective function:

$$L_{\text{total}} = \sum_{\ell=1}^{L} \alpha_\ell L_{c}^{\ell} + \Gamma \sum_{\ell=1}^{L} \beta_\ell L_{s}^{\ell}$$

(B.1a)

with:

$$L_{c}^{\ell} = \frac{1}{2Nd_{\ell}} \sum_{ij} (F_{\ell}[O] - F_{\ell}[I])_{ij}^2$$

(B.1b)

$$L_{s}^{\ell} = \frac{1}{2N_{\ell}^2} \sum_{ij} (G_{\ell}[O] - G_{\ell}[S])_{ij}^2$$

(B.1c)

where $L$ is the total number of convolutional layers and $\ell$ indicates the $\ell$-th convolutional layer of the deep convolutional neural network. In each layer, there
are $N_\ell$ filters each with a vectorized feature map of size $D_\ell$. $F_\ell[\cdot] \in \mathbb{R}^{N_\ell \times D_\ell}$ is the feature matrix with $(i, j)$ indicating its index and the Gram matrix $G_\ell[\cdot] = F_\ell[\cdot]F_\ell[\cdot]^T \in \mathbb{R}^{N_\ell \times N_\ell}$ is defined as the inner product between the vectorized feature maps. $\alpha_\ell$ and $\beta_\ell$ are the weights to configure layer preferences and $\Gamma$ is a weight that balances the tradeoff between the content (Eq. B.1b) and the style (Eq. B.1c).

**Photorealism regularization.** We now describe how we regularize this optimization scheme to preserve the structure of the input image and produce photorealistic outputs. Our strategy is to express this constraint not on the output image directly but on the transformation that is applied to the input image. Characterizing the space of photorealistic images is an unsolved problem. Our insight is that we do not need to solve it if we exploit the fact that the input is already photorealistic. Our strategy is to ensure that we do not lose this property during the transfer by adding a term to Equation B.1a that penalizes image distortions. Our solution is to seek an image transform that is locally affine in color space, that is, a function such that for each output patch, there is an affine function that maps the input RGB values onto their output counterparts. Each patch can have a different affine function, which allows for spatial variations. To gain some intuition, one can consider an edge patch. The set of affine combinations of the RGB channels spans a broad set of variations but the edge itself cannot move because it is located at the same place in all channels.

Formally, we build upon the Matting Laplacian of Levin et al. [113] who have shown how to express a grayscale matte as a locally affine combination of the input RGB channels. They describe a least-squares penalty function that can be minimized with a standard linear system represented by a matrix $M_\ell$ that only
depends on the input image \( I \) (We refer to the original article for the detailed derivation. We name \( V_c[O] \) the vectorized version \((N \times 1)\) of the output image \( O \) in channel \( c \) and define the following regularization term that penalizes outputs that are not well explained by a locally affine transform:

\[
\mathcal{L}_m = \sum_{c=1}^{3} V_c[O]^T M_I V_c[O] \tag{B.2}
\]

Using this term in a gradient-based solver requires us to compute its derivative w.r.t. the output image. Since \( M_I \) is a symmetric matrix, we have:

\[
\frac{\partial \mathcal{L}_m}{\partial V_c[O]} = 2M_I V_c[O].
\]

**Augmented style loss with semantic segmentation.** A limitation of the style term (Eq. B.1c) is that the Gram matrix is computed over the entire image. Since a Gram matrix determines its constituent vectors up to an isometry [240], it implicitly encodes the exact distribution of neural responses, which limits its ability to adapt to variations of semantic context and can cause “spillovers”. We address this problem with an approach akin to Neural Doodle [12] and a semantic segmentation method [31] to generate image segmentation masks for the input and reference images for a set of common labels (sky, buildings, water, etc.). We add the masks to the input image as additional channels and augment the neural style algorithm by concatenating the segmentation channels and updating the style loss as follows:

\[
L'_{ss+} = \sum_{c=1}^{C} \frac{1}{2N^2_{\ell,c}} \sum_{i,j} (G_{\ell,c}[O] - G_{\ell,c}[S])_{i,j}^2 \tag{B.3a}
\]

\[
F_{\ell,c}[O] = F_{\ell}[O] M_{\ell,c}[I] \quad F_{\ell,c}[S] = F_{\ell}[S] M_{\ell,c}[S] \tag{B.3b}
\]

where \( C \) is the number of channels in the semantic segmentation mask, \( M_{\ell,c}[\cdot] \) denotes the channel \( c \) of the segmentation mask in layer \( \ell \), and \( G_{\ell,c}[\cdot] \) is the Gram
matrix corresponding to $F_{\ell,c}[:]$.

We downsample the masks to match the feature map spatial size at each layer of the convolutional neural network.

To avoid “orphan semantic labels” that are only present in the input image, we constrain the input semantic labels to be chosen among the labels of the reference style image. While this may cause erroneous labels from a semantic standpoint, the selected labels are in general equivalent in our context, e.g., “lake” and “sea”. We have also observed that the segmentation does not need to be pixel accurate since eventually the output is constrained by our regularization.

**Our approach.** We formulate the photorealistic style transfer objective by combining all 3 components together:

$$L_{\text{total}} = \sum_{\ell=1}^{L} \alpha_{\ell} L_{\ell}^{c} + \Gamma \sum_{\ell=1}^{L} \beta_{\ell} L_{\ell}^{s} + \lambda L_{m}$$  \hspace{1cm} (B.4)

where $L$ is the total number of convolutional layers and $\ell$ indicates the $\ell$-th convolutional layer of the deep neural network. $\Gamma$ is a weight that controls the style loss. $\alpha_{\ell}$ and $\beta_{\ell}$ are the weights to configure layer preferences. $\lambda$ is a weight that controls the photorealism regularization. $L_{\ell}^{c}$ is the content loss (Eq. B.1b). $L_{\ell}^{s}$ is the augmented style loss (Eq. B.3a). $L_{m}$ is the photorealism regularization (Eq. B.2).

**B.3 Deep Photo Style Transfer**

We have performed a series of experiments to validate our approach. We first discuss visual comparisons with previous work before reporting the results of
two user studies.

We compare our method with Gatys et al. [52] (*Neural Style* for short) and Li et al. [116] (*CNNMRF* for short) across a series of indoor and outdoor scenes in Figure B.2. Both techniques produce results with painting-like distortions, which are undesirable in the context of photographic style transfer. The Neural Style algorithm also suffers from spillovers in several cases, e.g., with the sky taking on the style of the ground. And as previously discussed, CNNMRF often generates partial style transfers that ignore significant portions of the style image. In comparison, our photorealism regularization and semantic segmentation prevent these artifacts from happening and our results look visually more satisfying. In Figure B.3, we compare our method with global style transfer methods that do not distort images, Reinhard et al. [191] and Pitié et al. [181]. Both techniques apply a global color mapping to match the color statistics between the input image and the style image, which limits the faithfulness of their results when the transfer requires spatially-varying color transformation. Our transfer is local and capable of handling context-sensitive color changes.

### B.4 Deep Painting Harmonization

We extend the above framework to painting compositing. The key contribution of our technique is a novel algorithm to copy an object in a photograph and paste it into a painting seamlessly, i.e., the composite still looks like a genuine painting. We have introduced a two-pass algorithm that first transfers the overall style of the painting to the input and then refines the result to accurately match the painting’s color and texture. This latter pass relies on mapping neu-
ral response statistics that ensures consistency across the network layers and in image space. To cope with different painting styles, we have trained a separate network to adjust the transfer parameters as a function of the style of the background painting. Our experiments show that our approach succeeds on a diversity of input and style images, many of which are challenging for other methods. We have also conducted two user studies that show that users often identify our results as unedited paintings and prefer them to the outputs of other techniques. For more technical details on this part, please refer to the EGSR 2018 paper [134].

B.5 Conclusions

We introduce a deep-learning approach that faithfully transfers style from a reference image for a wide variety of image content. We use the Matting Laplacian to constrain the transformation from the input to the output to be locally affine in colorspace. Semantic segmentation further drives more meaningful style transfer yielding satisfying photorealistic results in a broad variety of scenarios, including transfer of the time of day, weather, season, and artistic edits.
Figure B.2: Comparison of our method against Neural Style and CNNMRF. Both Neural Style and CNNMRF produce strong distortions in their synthesized images. Neural Style also entirely ignores the semantic context for style transfer. CNNMRF tends to ignore most of the texture in the reference style image since it uses nearest neighbor search. Our approach is free of distortions and matches texture semantically.
Figure B.3: Comparison of our method against Reinhard et al. [191] and Pitié [181]. Our method provides more flexibility in transferring spatially-variant color changes, yielding better results than previous techniques.
Figure B.4: Example painting harmonization results with insets on proposed composite for unadjusted cut-and-paste, four state-of-the-art methods and our results. We show that our method captures both spatial and inter-scale color and texture and produces harmonized results on paintings with various styles. Zoom in for details.


[137] Robert Maier, Kihwan Kim, Daniel Cremers, Jan Kautz, and Matthias Nießner. Intrinsic3d: High-quality 3D reconstruction by joint appear-


particles: radiative transfer and coherent backscattering. Cambridge University, 2006.


[194] Osborne Reynolds. *Papers on mechanical and physical subjects: the sub- 
-mechanics of the universe*, volume 3. The University Press, 1903.

[195] Jérémie Riviere, Pieter Peers, and Abhijeet Ghosh. Mobile surface reflectom-
Library, 2016.

[196] Jérémie Riviere, Ilya Reshetouski, Luka Filipi, and Abhijeet Ghosh. Polar-
ization imaging reflectometry in the wild. *ACM Trans. Graph.*, 36(6):1–14, 
2017.

[197] Gareth O Roberts and Jeffrey S Rosenthal. Coupling and ergodicity of 
adaptive markov chain monte carlo algorithms. *Journal of applied probabi-


[199] Gareth O Roberts and Osnat Stramer. Langevin diffusions and 
metropolis-hastings algorithms. *Methodology and computing in applied prob-
ability*, 2002.

[200] Gareth O Roberts and Richard L Tweedie. Exponential convergence of 


[203] Y Sahillioğlu and Yücel Yemez. Coarse-to-fine surface reconstruction from 
silhouettes and range data using mesh deformation. *Computer Vision and Image 

[204] Carolin Schmitt, Simon Donne, Gernot Riegler, Vladlen Koltun, and And-
dreas Geiger. On joint estimation of pose, geometry and svBRDF from a 

[205] Carolin Schmitt, Simon Donne, Gernot Riegler, Vladlen Koltun, and And-
dreas Geiger. On joint estimation of pose, geometry and svbrdf from a 


[216] Vincent Sitzmann, Justus Thies, Felix Heide, Matthias Nießner, Gordon


[249] Kuk-Jin Yoon, Emmanuel Prados, and Peter Sturm. Joint estimation of
shape and reflectance using multiple images with known illumination conditions. IJCV, 2010.


