UC Santa Barbara

UC Santa Barbara Electronic Theses and Dissertations

Title

Utilizing Machine Learning for Filtering General Monte Carlo Noise

Permalink

https://escholarship.org/uc/item/6cw687x8

Author

Khademi Kalantari, Nima Khademi

Publication Date

2015

Supplemental Material

https://escholarship.org/uc/item/6cw687x8#supplemental

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

UNIVERSITY OF CALIFORNIA
Santa Barbara

Utilizing Machine Learning for Filtering General Monte Carlo Noise

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Electrical and Computer Engineering

by

Nima Khademi Kalantari

Committee in Charge:

Professor Pradeep Sen, Chair

Professor B. S. Manjunath

Professor Alberto Giovanni Busetto

Professor Matthew Turk

December 2015

The Dissertation of
Nima Khademi Kalantari is approved:

Professor B. S. Manjunath

Professor Alberto Giovanni Busetto

Professor Matthew Turk

Professor Pradeep Sen, Committee Chairperson

December 2015

Utilizing Machine Learning for Filtering

General Monte Carlo Noise

Copyright © 2015

by

Nima Khademi Kalantari

To Mom and Dad for their love.

Acknowledgements

I would like to thank my advisor, Pradeep Sen, for his insightful advice, encouragement, and financial support. From Pradeep, I have learned a great deal about research, career, and life. Furthermore, he gave me the freedom to pursue my own ideas and provided constant guidance throughout my Ph.D.

I also want to thank Eli Shechtman and Dan Goldman for providing excellent research mentorship during my two internships at Adobe.

Thank you to my colleagues at UCSB MIRAGE Lab and UNM AGL for great discussions and useful research advice, including Steve Bako, Yuxiang Wang, Abhishek Badki, Chieh-Chi Kao, Atieh Taheri, Ekta Prashnani from MIRAGE Lab and Soheil Darabi, Maziar Yaesoubi, Lei Xiao, Vahid Noormofidi, and Mauricio Gomez from AGL. Specifically, I would like to thank Steve Bako for his great help with several projects including all my dissertation projects, and Maziar Yaesoubi, Soheil Darabi, and Lei Xiao for their great help with several projects.

I wish to thank my parents and family for their constant support and encouragement.

Finally, I thank the National Science Foundation (IIS-1342931 and IIS-1321168) for funding this work. Moreover, I would like to thank Adobe Inc. for their partial financial support during my Ph.D. Also, I thank Nvidia for the hardware donation

of a GeForce GTX TITAN and various artists for providing scenes and models used in this work.

Curriculum Vitæ

Nima Khademi Kalantari

Education

2015	Ph.D. in Electrical and Computer Engineering, University of California at Santa Barbara, USA.
2009	M.S. in Electrical and Computer Engineering, Amirkabir University of Technology, Iran.
2007	B.S. in Electrical and Computer Engineering, Amirkabir University of Technology, Iran.

Experience

2012–2015	Research Assistant, MIRAGE Lab, University of California at Santa Barbara, USA.
2012, 2013	Research Intern, Adobe Creative Technologies Lab, Seattle, USA.
2010–2012	Research Assistant, Advanced Graphics Lab, University of New Mexico, USA.
2007–2009	Research Assistant, Speech Processing Research Lab, Amirkabir University of Technology, Iran.

Selected Awards

2015	Dissertation Fellowship, ECE Department, University of California at Santa Barbara
2011	Departmental Fellowship, ECE Department, University of New Mexico
2010	Best Master's Thesis Award, Amirkabir University of Technology
2010	Best Master's Thesis Award, IEEE Iran Section
2007	Best Undergraduate Project Award, Amirkabir University of Technology

Selected Publications

h-index: 10 (source Google Scholar, Sept. 2015)

- Nima Khademi Kalantari, Steve Bako, and Pradeep Sen. A learning-based approach for filtering Monte Carlo noise. *ACM Trans. Graph. (SIGGRAPH 2015)*, 34(4), August 2015.
- Abhishek Badki, Nima Khademi Kalantari, and Pradeep Sen. Robust radiometric calibration for dynamic scenes in the wild. *In Computational Photography (ICCP), 2015 IEEE International Conference on*, April 2015.

- Nima Khademi Kalantari, Eli Shechtman, Soheil Darabi, Dan B. Goldman, and Pradeep Sen. Improving patch-based synthesis by learning patch masks. *In Computational Photography (ICCP), 2014 IEEE International Conference on*, pages 1–8, May 2014.

- Nima Khademi Kalantari, Eli Shechtman, Connelly Barnes, Soheil Darabi, Dan B. Goldman, and Pradeep Sen. Patch-based high dynamic range video. *ACM Trans. Graph. (SIGGRAPH Asia 2013)*, 32(6):202:1–202:8, November 2013.

- Nima Khademi Kalantari and Pradeep Sen. Removing the noise in Monte Carlo rendering with general image denoising algorithms. *Computer Graphics Forum (Proceedings of Eurographics 2013)*, 32(2):93–102, 2013.

- Pradeep Sen, Nima Khademi Kalantari, Maziar Yaesoubi, Soheil Darabi, Dan B. Goldman, and Eli Shechtman. Robust patch-based HDR reconstruction of dynamic scenes. *ACM Trans. Graph. (SIGGRAPH Asia 2012)*, 31(6):203:1–203:11, November 2012.

- Nima Khademi Kalantari and Pradeep Sen. Fast generation of approximate blue noise point sets. *Computer Graphics Forum (Proceedings of EGSR 2012)*, 31(4):1529–1535, June 2012.

- Nima Khademi Kalantari and Pradeep Sen. Efficient computation of blue noise point sets through importance sampling. *Computer Graphics Forum (Proceedings of EGSR 2011)*, 30(4):1215–1221, 2011.

- Nima Khademi Kalantari and Seyed Mohammad Ahadi. A logarithmic quantization index modulation for perceptually better data hiding. *Image Processing, IEEE Transactions on*, 19(6):1504–1517, June 2010.

- Mohammad Ali Akhaee, Nima Khademi Kalantari, and Farokh Marvasti. Robust audio and speech watermarking using gaussian and laplacian modeling. *Signal Process.*, 90(8):2487–2497, August 2010.

- Nima Khademi Kalantari, Seyed Mohammad Ahadi, and Mansur Vafadust. A robust image watermarking in the Ridgelet domain using universally optimum decoder. *Circuits and Systems for Video Technology, IEEE Transactions on*, 20(3):396–406, March 2010.

- Nima Khademi Kalantari and Seyed Mohammad Ahadi. Rational dither modulation using logarithmic quantization with optimum parameter. *In Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*, pages 1738–1741, March 2010.

- Nima Khademi Kalantari, Mohammad Ali Akhaee, Seyed Mohammad Ahadi, and Hamidreza Amindavar. Robust multiplicative patchwork method for au-

dio watermarking. *Audio, Speech, and Language Processing, IEEE Transactions on*, 17(6):1133–1141, Aug 2009.

- Mohammad Ali Akhaee, Nima Khademi Kalantari, and Farokh Marvasti. Robust multiplicative audio and speech watermarking using statistical modeling. *In Communications (ICC), 2009 IEEE International Conference on*, pages 1–5, June 2009.

- Nima Khademi Kalantari and Seyed Mohammad Ahadi. Logarithmic quantization index modulation: A perceptually better way to embed data within a cover signal. *In Acoustics, Speech and Signal Processing (ICASSP), 2009 IEEE International Conference on*, pages 1433–1436, April 2009.

- Nima Khademi Kalantari, Seyed Mohammad Ahadi, and Hamidreza Amindavar. A universally optimum decoder for multiplicative audio watermarking. *In Multimedia and Expo (ICME), 2008 IEEE International Conference on*, pages 225–228, June 2008.

- Nima Khademi Kalantari and Seyed Mohammad Ahadi. Vector quantization index modulation watermarking using concentric hyperspherical codebooks. *In Acoustics, Speech and Signal Processing (ICASSP)*, 2008 IEEE International Conference on, pages 1741–1744, March 2008.

Abstract

Utilizing Machine Learning for Filtering General Monte Carlo Noise

Nima Khademi Kalantari

Producing photorealistic images from a scene model requires computing a complex multidimensional integral of the scene function at every pixel of the image. Monte Carlo (MC) rendering systems approximate this integral by tracing light rays (samples) in the multidimensional space to evaluate the scene function. Although an approximation to this integral can be quickly evaluated with just a few samples, the inaccuracy of this estimate relative to the true value appears as unacceptable noise in the resulting image.

One way to mitigate this problem is to quickly render a noisy image with a few samples and then filter it as a post-process to generate an acceptable, noise-free result. This approach has been the subject of extensive research in recent years and many algorithms have been developed. However, the majority of these approaches use simple, heuristic rules to design the algorithm and, as a result, have various limitations.

We begin by studying how standard image denoising techniques can be applied to the problem of Monte Carlo rendering. To do this, we propose a way to use

any standard image denoising method (e.g., BM3D) to remove noise from MC rendered images. We do this by estimating the amount of noise at each pixel of the image and apply a multilevel algorithm that denoises the image in a spatially-varying manner. We then show that although this approach works better than the previous color-based schemes, i.e., the methods that only use color information, it cannot handle complex scenes with severe noise. This is due to the fact that this algorithm does not utilize additional scene features such as world positions, shading normals, and texture values, which are available in MC rendering.

To address the filtering problem systematically, we then present a new way of analyzing the MC filtering approaches. We observe that the major challenge in all filtering techniques is filter parameter estimation. Our key contribution is to address this challenging problem with two machine learning approaches. Specifically, we first propose to estimate the optimal filter parameters at each pixel directly from the output of the MC renderer using a neural network. We train the network on a set of scenes by minimizing the error between the filtered and ground truth images. Second, we propose to find the optimal filter parameter sets in an error-minimization filtering approach to produce filtered results as close as possible to the ground truth. We optimize these candidate filter parameter sets on a set of training scenes by minimizing the error between the filtered and ground truth images.

We show that the proposed approaches outperform state-of-the-art methods in removing general MC noise. In this thesis, we present the first attempt to use machine learning for removing noise from MC rendered images. We believe this opens a new avenue for future work and we hope other researchers can build upon the ideas presented here to further advance the MC filtering field.

Contents

List of Figures **xvii**

List of Tables **xix**

1 Introduction **1**
1.1 Thesis statement 10
1.2 Contributions 10
1.3 Outline 11

2 Related work in MC filtering **13**
2.1 Color-based filters 14
2.2 Filters that use additional information 16
2.3 Conclusion 18

3 Removing MC noise with standard image denoising **19**
3.1 Background in image denoising 20
3.2 Proposed algorithm 22
3.2.1 Noise estimation metric 23
3.2.2 Multilevel denoising framework 24
3.2.3 Adaptive sampling using noise estimation 27
3.3 Results 28
3.4 Discussion 33
3.5 Conclusion 35

4 Reformulating the MC filtering problem **37**
4.1 Filter parameter estimation 41
4.1.1 Direct estimation 42
4.1.2 Error-minimization 43

5 Learning-based Monte Carlo filtering **45**
5.1 Neural networks in graphics/denoising 47
5.2 A new learning framework for MC filtering 48
5.2.1 Our proposed neural network 51
5.2.2 Primary features 58
5.2.3 Secondary features 60
5.3 Implementation details 62
5.3.1 Neural network structure 62
5.3.2 Training 63
5.3.3 Handling high dynamic range spikes 65
5.4 Results 66
5.5 Discussion 74
5.6 Conclusion 80

6 Training for optimal filter parameter sets **82**
6.1 Error-minimization with optimal filter sets 83
6.1.1 Finding optimal candidate filter parameters 85
6.1.2 Filter 87
6.1.3 No-reference error metric 91
6.1.4 Final image reconstruction 91
6.2 Extensions 93
6.3 Results 94
6.4 Discussion 98
6.5 Conclusion 102

7 Discussion and future work **103**

8 Conclusion **111**

Bibliography **113**

List of Figures

1.1 MC rendering convergence 5
1.2 Direct application of BM3D on an MC rendered image 8
1.3 Additional scene features in MC rendering 9

3.1 Noise estimation accuracy 24
3.2 Noise level selection 26
3.3 Overview of our adaptive sampling algorithm 28
3.4 Comparison of denoising algorithms 30
3.5 Comparison against state-of-the-art approaches 32
3.6 Effect of changing the number of levels 34
3.7 Limitation of color-based approaches 35

4.1 Illustration of the requirement to vary filter parameters at each pixel 40
4.2 Overview of direct estimation approaches 41
4.3 Overview of error-minimization approaches 43

5.1 Multilayer perceptron (MLP) 50
5.2 Algorithm overview 51
5.3 Effect of spike removal 66
5.4 Robustness of the learning system 68
5.5 Training set 68
5.6 Comparison against state-of-the-art methods 69
5.7 Convergence 73
5.8 Comparison against state-of-the-art approaches using NLM filter . 75
5.9 Effect of network size 77
5.10 Effect of removing the glossy scenes from the training set 78
5.11 Effect of computing the secondary features only on the color . . . 79
5.12 Comparison at high sampling rate 80

6.1 Equal-time comparison against state-of-the-art 95
6.2 Convergence 98
6.3 Comparison at high sampling rate 99
6.4 Comparison of the candidate filters 101

7.1 General filtering framework 105

List of Tables

6.1 Filter parameters for the two candidates 100

Chapter 1

Introduction

Rendering is the process of producing photorealistic images from a scene description which contains information about the geometry of the objects, position of the light sources, camera, etc. Rendering has a variety of applications including creating movie or TV visual effects and animations, video games, architecture designs, and simulators. This process requires simulating the light transport in the scene by computing a complex multidimensional integral of the scene function at every pixel of the image. For example, generating distributed effects like depth of field and motion blur requires integrating over domains such as lens position and time. In this case, we can compute a specific pixel value by taking an integral of a 5D continuous "scene function" $f(x, y, u, v, t)$, where (x, y) defines the position on the image domain, (u, v) is the position on the lens, and t describes a specific time within the interval that the camera shutter is open. Intuitively, this scene function measures the amount of incoming light (radiance) through the image

plane for a specific multidimensional ray which in this case is defined by x, y, u, v, and t. The integral of this scene function at each pixel can be formalized as:

$$c_p = \int_{p_x-\frac{1}{2}}^{p_x+\frac{1}{2}} \int_{p_y-\frac{1}{2}}^{p_y+\frac{1}{2}} \int_{-1}^{1} \int_{-1}^{1} \int_{t_0}^{t_1} f(x, y, u, v, t)\, \mathrm{d}t\, \mathrm{d}v\, \mathrm{d}u\, \mathrm{d}y\, \mathrm{d}x, \tag{1.1}$$

where c_p is the final color at pixel p, computed by integrating the scene function over pixel p in the x and y directions, over the camera lens from $[-1, 1]$, and over the time that the shutter is open $[t_0, t_1]$. Additional distributed effects such as area lighting and glossy reflections increase the dimensionality of the scene function and, consequently, the complexity of the integral. Unfortunately, for a complex scene with a variety of distributed effects, finding an analytical solution to this integral is almost impossible.

Whitted [107] proposed one of the first renderers that traced a single ray at each pixel to evaluate the scene radiance and produced simple effects such as reflections and hard shadows. Shortly after, Monte Carlo (MC) approaches [13, 14, 44] were introduced to handle a variety of interesting effects such as depth of field, motion blur, and glossy reflections. These MC methods approximate the rendering integral by tracing randomly generated light rays (samples) in the multidimensional space to evaluate the scene function. The basic idea is that the continuous integral can be approximated by point sampling the scene function

and averaging all the samples at every pixel. To better understand these MC approaches, we analyze the numerical evaluation of the following simple integral:

$$c = \int_a^b f(x)\mathrm{d}x. \tag{1.2}$$

As discussed, MC approaches estimate this integral by evaluating the function $f(x)$ at N samples $x_1, \cdots, x_N$ and averaging the results as follows:

$$\bar{c} = \frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)}{p(x_i)}, \tag{1.3}$$

where $p(x_i)$ is the probability of each sample and $\bar{c}$ is the MC estimation of the true value of the integral, c. The expected value of this estimator can be computed as follows:

$$\begin{aligned} E[\bar{c}] &= E[\frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)}{p(x_i)}] \\ &= \frac{1}{N}\sum_{i=1}^{N}E[\frac{f(x_i)}{p(x_i)}] \\ &= E[\frac{f(x)}{p(x)}] = \int_a^b \frac{f(x)}{p(x)}p(x)\mathrm{d}x \\ &= \int_a^b f(x)\mathrm{d}x = c. \end{aligned} \tag{1.4}$$

Here, we were able to remove the summation since all the individual samples, x_i, have the same statistical behavior. This equation shows that the MC estimator is unbiased since the expected value of the estimate is always equal to the true value of the integral. However, this estimator has variance which decreases linearly

($O(1/N)$) with the number of samples. Assuming all the samples to have a uniform probability density function, i.e., $p(x_i) = p(x_j) = 1/(b-a)$, and to be independent, we can compute this variance as follows:

$$\begin{aligned} \text{var}[\bar{c}] &= \text{var}[\frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)}{p(x_i)}] \\ &= \text{var}[\frac{1}{N}\sum_{i=1}^{N}(b-a)f(x_i)] \\ &= \frac{(b-a)^2}{N^2}\sum_{i=1}^{N}\text{var}[f(x_i)] \\ &= \frac{(b-a)^2}{N}\text{var}[f(x)]. \end{aligned} \tag{1.5}$$

Note that this analysis can be easily extended to multidimensional functions. Therefore, if we evaluate the scene function at enough multidimensional samples, the variance of the MC rendering system will converge to zero, resulting in a physically correct image. However, with just a few samples, the inaccuracy of this estimate relative to the true value appears as unacceptable noise in the resulting image, as shown in Fig. 1.1. Although we can reduce the noise by using more samples at each pixel, many expensive multidimensional samples are required to get a reliable estimate of the integral since the variance decreases linearly. The high cost of computing additional rays results in lengthy render times that negatively affect the applicability of MC renderers in real applications.



Figure 1.1: We show the San Miguel Terrace scene, rendered with different sampling rates. Although we can quickly render an image with 4 samples per pixel (spp) to approximate the integral in Eq. 1.1, the inaccuracy of this estimate with respect to the true value of the integral appears as noise. As the sampling rate increases, our estimation becomes more accurate, and as a result, the final image becomes less noisy. However, since the variance of the MC estimator decreases linearly with the number of samples, many samples are required to produce a visually-pleasing noise-free result. Here, for example, the converged result is obtained after rendering 64K samples per pixel over the course of four days. Such lengthy render times limit the applicability of MC rendering in practice.

Researchers have proposed a variety of variance reduction techniques over the past years. One way to reduce this variance is to use better sampling patterns such as stratified [13, 20], low-discrepancy [91], and Poisson-disk samples [13, 20, 46, 49, 106] which have been shown to be superior to simple random sampling. For example, in stratified sampling, the integration domain is divided into equally-sized non-overlapping regions and a single sample is placed randomly inside each region. In general, these sampling patterns usually distribute the samples better and, therefore, reduce the variance of MC estimator more effectively.

Importance sampling is another variance reduction technique in which the samples are placed in the more important regions [12, 36, 63, 64, 99]. These

approaches try to estimate the probability of the scene function and place more samples in the regions with higher probabilities. This is a challenging task as we usually do not have knowledge about the scene function and these probabilities need to be estimated at run-time.

Faster ray-tracing algorithms have also been investigated in the past. These include faster intersection algorithms [38, 48, 65], techniques using more efficient data structures [30, 32, 33, 39, 74, 101], memory coherent approaches [31, 69, 102, 103], and fast parallel/GPU implementations [1, 9, 34, 71, 76, 104]. Although these approaches do not reduce the variance of the estimator, they render more samples in a given time and, thus, produce results with less noise.

Moreover, several approaches have been proposed to reduce the variance of the estimator through better light path sampling [42, 44, 50, 100]. For example, Kajiya [44] proposed path tracing where only a single path is traced from the camera to the light source in order to evaluate the scene function at each sample. To further reduce the variance, Lafortune and Willems [50] extended this idea by presenting bi-directional path tracing where each path was traced from both the light and the camera.

Another effective noise reduction technique, and the focus of this thesis, is to quickly render a noisy image with a few samples and then filter it to generate a noise-free result. Note that the approaches in other categories modify the renderer

to trace the samples more efficiently and are orthogonal to simple post-process filtering techniques. Therefore, these approaches can usually be utilized in the MC renderer to further improve the quality of the filtered results.

The filtering idea was introduced shortly after the introduction of MC rendering. However, after a few initial attempts to filter general MC noise [53, 84], the rendering community largely ignored filtering techniques due to the commonly-held belief that the filters cannot distinguish between unwanted noise and valid scene details. It is worth noting that many filtering techniques have been proposed in the past, but most of these approaches [27, 28, 60, 62] tackle a specific distributed effect such as depth of field and global illumination and are not general. Inspired by image denoising techniques, several methods have been proposed [19, 70, 80, 81, 108] to remove general MC noise. These methods usually modify a specific image denoising method for the MC rendering problem. However, they have had limited success for various reasons, as we will discuss in this thesis.

We begin by studying the possibility of applying standard image denoising methods to the problem of MC denoising. We do this by proposing a framework that is able to use any powerful image denoising technique to remove general MC noise (Chapter 3). This is challenging since most of the image denoising algorithms assume that the noise is spatially-invariant and does not change glob-

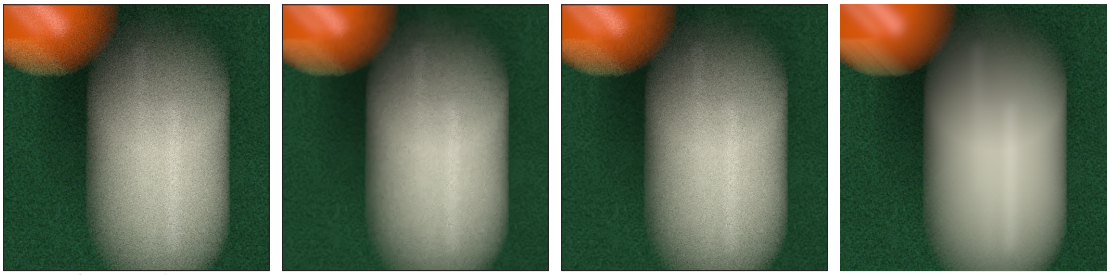


Figure 1.2: On the left, we show an inset of the PoolBall scene with motion blur rendered at 4 samples per pixel (spp). As seen, the green surface is noise-free, while the white and orange balls appear noisy due to motion blur. We use BM3D [15] to denoise the input noisy image using two different noise standard deviations (10 and 17). When the noise standard deviation is set to 17, the details on the green surface are removed, but the balls are fairly smooth. The result with a noise standard deviation of 10 keeps the details on the green surface better, but contains severe noise on the balls. Therefore, standard spatially-invariant image denoising techniques cannot be directly applied to MC rendered images with spatially-varying noise.

ally across the image. However, this is not true in MC filtering since the noise due to the random parameters in the MC rendering system is often local and spatially-varying. Therefore, applying the image denoising algorithms directly on MC rendered images produces unsatisfactory results, as shown in Fig. 1.2. We overcome the this difficulty by using a metric which estimates the amount of noise at each pixel in the image.

However, although our approach produces better results than other aforementioned techniques [19, 80, 81], it still is not able to handle MC images with severe noise, a common situation when rendering complex scenes with a variety of distributed effects. This is due to the fact that these approaches only use pixel color

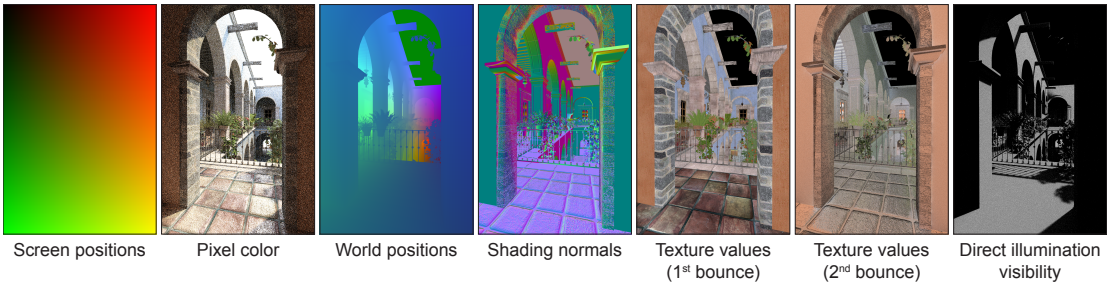


Figure 1.3: The MC renderer can directly output a set of features such as world positions, shading normals, and texture values, etc. in addition to color and screen positions when shading samples. Here, we visualize them by averaging all the samples in each pixel and displaying them as color images. For example, we show the screen positions by replacing the R and G channels of an image with the screen positions in the x and y directions, respectively. As seen, these features are highly correlated with scene detail and provide useful information to improve the quality of the filtered result. For example, normals help recover the bump map on the floor from the noisy pixel color image.

to remove noise from the rendered images. However, MC rendering systems can directly output additional scene features such as world positions, shading normals, texture values, etc. These features, shown in Fig. 1.3, can be used in the filtering process to greatly improve the quality of the results as they are highly correlated with scene detail.

More recently, several approaches have proposed to utilize additional scene features using feature-based filters such as cross-bilateral and cross non-local means filters [56, 66, 67, 82, 90], which renewed interest in filtering techniques in the rendering community. In this thesis, we present a new way of looking at the MC filtering problem and observe that the main challenge in all the filtering techniques is how to set the filter weight for these features (called *filter parameters*

hereafter). We then present a new way of classifying the previous general MC filtering approaches into two classes based on how they estimate the filter parameters. Some techniques directly estimate the filter parameters from the output of MC rendering, while others select the filter parameters with minimum error from a pre-defined sets of filter parameters. Our key contribution is to introduce machine learning to address the challenging filter parameter estimation problem in these two classes of approaches (Chapters 5 and 6).

1.1 Thesis statement

In this thesis, we observe that the main challenge with all general MC filtering approaches is finding the optimum filter parameters at every pixel that will produce a result as close as possible to the ground truth. By focusing on this problem, we can not only gain a better understanding of previous methods, but also develop new algorithms for MC denoising. Our main contribution is to introduce machine learning to address this challenging problem in a systematic way. Our proposed approaches show improvement over the state-of-the-art filtering methods.

1.2 Contributions

This thesis makes the following major contributions:

- We present a novel framework by which we are able to use any high-performance image denoising algorithm to remove MC noise. We further show that although this approach is superior to previous color-based MC filtering techniques, it cannot handle cases with severe noise since it only uses pixel color information.

- We present a new way of looking at the MC filtering problem and identify the filter parameter estimation problem to be the main challenge in all the filtering techniques.

- We introduce machine learning to address the challenging filter parameter estimation problem in two major filtering frameworks. First, we train a neural network to estimate the filter parameters in a direct estimation framework. Second, we optimize the candidate filter parameter sets in an error-minimization framework using a set of training scenes. We note that this thesis demonstrates the first use of machine learning for removing noise from MC rendered images.

1.3 Outline

The remainder of the thesis is organized as follows. We start by reviewing the related MC filtering work in Chapter 2. In Chapter 3, we explain our proposed

framework for applying any spatially-invariant image denoising technique for MC filtering. Then, in Chapter 4, we present a new way of looking at the filtering problem, identify the major challenge in all the filtering approaches to be the filter parameter estimation, and classify them into two major classes. Chapters 5 and 6 explain our machine learning approach to address the parameter estimation problem in these two major classes of algorithms. In Chapter 7, we show the connection between our proposed approaches in Chapters 5 and 6, discuss their advantages and disadvantages, and summarize the agenda for future investigation. Finally, we conclude the thesis in Chapter 8.

Chapter 2

Related work in MC filtering

The use of Monte Carlo algorithms for rendering was introduced by Cook et al. [13, 14] in their seminal work that extended standard Whitted ray-tracing [107] to produce a variety of interesting effects, such as depth of field and motion blur. Since then, researchers have been exploring a variety of algorithms to reduce MC noise and produce high-quality images in less time. As discussed in Chapter 1, there are various variance reduction techniques such as better sampling patterns [13, 20], importance sampling [63, 64], faster ray-tracing [33, 71], and more efficient light path sampling [44, 50].

Other strategies such as image reconstruction using compressed sensing [88, 89] and reprojection methods [54, 55] have also been proposed. Since the focus of this thesis is filtering general MC noise, we will mainly focus on filtering approaches since many of these other methods can be considered orthogonal. We refer readers

seeking more background on MC rendering to texts on the subject [25, 73] or the recent state-of-the-art report by Zwicker et al. [111].

Filtering approaches render a noisy image with a few samples and then denoise it through a filtering post-process. In addition, some of these methods use adaptive sampling to place more samples in the noisy regions and reduce noise, further improving the results. We divide the previous work on MC filtering into two general categories: 1) methods that use only sample color during filtering, and 2) methods that use additional scene features such as world positions, shading normals, texture values, etc.

2.1 Color-based filters

These methods, similar to image denoising techniques, only use pixel color information from the rendering system to remove MC noise. Early work by Lee and Redner [53] used nonlinear filters (median and alpha-trimmed mean filters) to remove spikes while preserving edges. Rushmeier and Ward [84] proposed to spread the energy of input samples through variable width filter kernels. To reduce the noise in path-traced images, Jensen and Christensen [43] separated illumination into its direct and indirect components, filtered the indirect portion, and then added the components back together. Bala et al. [2] exploited an edge image

to facilitate the filtering process, while Xu and Pattanaik [108] used a bilateral filter [97] to remove MC noise.

Following the seminal work in studying the frequency aspects of light transport [10, 24, 77], several methods used similar analysis to optimize the adaptive placement of samples and the filtering for specific MC effects. Soler et al. [93] proposed to leverage sparsity in the Fourier domain for efficient sample placement and reconstruction of scenes with depth of field. Egan et al. used frequency analysis to shear a filter for specific distributed effects such as motion blur [28] and occlusion/shadowing [26, 27], while Mehta et al. used related analysis to derive simple formulas that set the variance of a simple screen-space Gaussian filter to target noise from specific effects [60, 61, 62]. Finally, Belcour et al. [6] proposed using 5D frequency analysis to efficiently handle depth of field and motion blur. However, these approaches focus on specific MC effects and are not general.

Inspired by the advances in image denoising using wavelet transforms, Overbeck et al. [70] proposed an adaptive wavelet rendering algorithm. They used a modified standard soft-thresholding denoising method [22] to filter MC rendered images. Rousselle et al. [80] proposed to select an appropriate scale for a Gaussian filter at every pixel to minimize the reconstruction error. Rousselle et al. [81] proposed a dual buffer strategy to estimate the variance at each pixel and filtered the MC images using a non-local means filter [7]. Finally, Delbracio et al. [19] pro-

posed a method based on non-local means filtering which computes the distance between two patches using their color histograms. The main problem with these previous approaches is that they modify a specific image denoising technique to remove noise from MC images. In contrast, we propose an algorithm (Chapter 3) that performs filtering using any advanced image denoising technique.

2.2 Filters that use additional information

The approaches in this category leverage additional scene features (e.g., world positions, shading normals, texture values, etc.) which are directly computed by the MC renderer. Thus, they tend to generate higher-quality results compared to the color-based approaches mentioned above. For example, McCool [59] removed MC noise by using depths and normals to create a coherence map for an anisotropic diffusion filter.

To efficiently render scenes with global illumination, Segovia et al. [87] and Laine et al. [51] used a geometry buffer. Meanwhile, to reduce global illumination noise, Dammertz et al. [18] incorporated wavelet information into the bilateral filter and Bauszat et al. [5] used guided image filtering. Shirley et al. [92] used a depth buffer to handle depth of field and motion blur effects, while Chen et al. [11] combined a depth map with sample variance to filter the noise from depth of field.

More recently, Bauszat et al. [3] proposed a fast manifold filtering approach for handling global illumination and depth of field effects. Note that all of these methods focus on a fixed set of distributed effects and are not general.

To handle general MC noise using additional scene features, Sen and Darabi [89, 90] observed the need to vary the filter's feature weights (filter parameters) across the image. Specifically, they proposed to compute these filter parameters using mutual information to approximate the functional dependencies between scene features and the random parameters. Li et al. [56] used Stein's unbiased risk estimator (SURE) [94] to estimate the appropriate spatial filter parameter in a cross-bilateral filter, while hard coding the weights of the remaining cross terms. Rousselle et al. [82] significantly improved upon this by using the SURE metric to select between three candidate cross non-local means filters that each weight color and features differently. Moon et al. [66] computed a weighted local regression on a reduced feature space and evaluated the error for a discrete set of filter parameters to select the best one. Bauszat et al. [4] posed the filter selection problem as an optimization and solved it with graph cuts. Finally, Moon et al. [67] proposed to use linear prediction at sparse points in the image for the final image reconstruction.

The main problem with such approaches is that they define the filtering system's parameters simply through heuristic rules. Thus, they are not able to ro-

bustly estimate the appropriate parameters in challenging cases. In contrast, we use machine learning to set the optimal parameters of two MC denoising frameworks in a systematic way (Chapters 5 and 6).

2.3 Conclusion

Although there are a significant number of MC filtering approaches, the majority of previous techniques define the filter parameters through logical, but heuristic rules. Therefore, these methods cannot handle complex scenes and usually produce results with under/over blurred regions. In this thesis, we systematically address this challenging problem by using machine learning to set the optimal filter parameters at each pixel.

Chapter 3

Removing MC noise with standard image denoising

Our main goal in this chapter is to study how to apply standard image denoising techniques to the MC noise reduction problem. This is challenging since, as discussed in Chapter 1, image denoising methods typically assume the noise to be spatially-invariant. As shown in Fig. 1.2, direct application of these approaches to Monte Carlo (MC) images produces unsatisfactory results. Therefore, most previous MC filtering approaches modify a specific image denoising scheme for the MC filtering problem.

Our key insight is that we can do this by first applying a metric that accurately estimates the standard deviation of the noise at every local region in the image. Once this is known, we can then use a multilevel algorithm to efficiently apply any spatially-invariant denoising method to the entire image with a small set of different noise parameters, and combine the results to produce the final noise-

free image. The noise estimation metric can also be used to guide an adaptive sampling step.

Using this general framework, we show that even the most advanced image denoising approaches cannot produce high-quality results in challenging cases. This is due to the fact that these image denoising methods inherently rely on only color information, while in MC rendering we have access to additional features such as world positions and shading normals.

In the next section, we start with a brief background on image denoising and then explain our algorithm.

3.1 Background in image denoising

Image denoising approaches typically formulate the problem by assuming the noise to be additive white Gaussian as follows,

$$\bar{c}_p = c_p + n_p, \tag{3.1}$$

where subscript p is the pixel coordinate and n_p is zero-mean Gaussian noise, with a constant standard deviation, σ, for all pixels in the image. The goal of denoising is therefore to estimate c_p given the noisy input image $\bar{c}_p$ and noise standard deviation σ at every pixel of the image. This formulation inherently assumes stationary noise, i.e., noise standard deviation σ is constant over the

entire image. Although this is a reasonable assumption in image processing since the noise typically comes from the sensor, it is not true in MC rendering where the noise comes from the MC process itself.

The more successful image denoising approaches use wavelet transform to perform denoising. The basic idea behind wavelet image denoising is to transform the noisy input image to the wavelet domain, apply a denoising step, and then transform it back to the spatial domain,

$$\hat{c} = \mathcal{W}^{-1}\left[\mathcal{D}\left(\mathcal{W}\left(\bar{c}\right), \sigma\right)\right],$$

where $\hat{c}$ is the estimate of the noise-free image, $\mathcal{W}\left(\bar{c}\right)$ transforms the input noisy image to the wavelet domain, $\mathcal{D}$ is the denoising step that takes the noisy coefficients and the noise standard deviation σ as input, and $\mathcal{W}^{-1}$ transforms the noise-free wavelet coefficients back to the spatial domain.

In our framework, we use two wavelet-based denoising algorithms which are briefly described below. Readers interested in a complete explanation of these techniques are referred to the original papers [75, 15].

Bayes least squares - Gaussian scale mixture (BLS-GSM) [75] – In this method, the noisy image is decomposed into pyramid subbands at different scales using an overcomplete basis. The noise-free coefficients are then estimated from

the noisy ones using a Bayes least-square estimator. Then, an inverse wavelet transform of the noise-free coefficients is calculated to obtain the final image.

Block-matching and 3D filtering (BM3D) [16] – In this method, the noisy input image is first divided into overlapping blocks of fixed size. For each block, the most similar blocks in the image are found and stacked into a 3D array. The wavelet transform of this 3D array is taken and the noise is attenuated by hard thresholding [21]. Finally, all the denoised blocks are returned to their original positions and averaged together to produce the denoised image. This process is repeated to further suppress the noise.

As mentioned earlier, these powerful methods assume a fixed noise level, σ, and cannot be used directly for denoising MC rendered images. In the next section, we explain our framework for applying these techniques to MC rendering.

3.2 Proposed algorithm

In order to leverage powerful image denoising methods such as BLS-GSM and BM3D to denoise MC rendering, the noise standard deviation, σ, at every pixel in the image must first be accurately estimated. We must then find a way to use these spatially-invariant algorithms in a spatially-varying manner. In the sections that follow, we describe the core of our proposed method.

3.2.1 Noise estimation metric

The goal of this section is to find a noise map that estimates the noise standard deviation for every pixel $\hat{\sigma}_p$. To do this, we leverage the median absolute deviation (MAD) metric introduced by Donoho and Johnstone [21], which assumes that noise is stationary inside a small window of pixels (8×8 in our implementation) around the pixel in question:

$$\hat{\sigma}_p = \frac{\text{median}(|\mathcal{D}_0^1 - \text{median}(\mathcal{D}_0^1)|)}{0.6745}, \tag{3.2}$$

where $\mathcal{D}_0^1$ represents the diagonal detail coefficients of the finest level of the wavelet transform, 0.6745 is a scaling constant, and $\hat{\sigma}_p$ is the estimated standard deviation of the noise at pixel p. This metric works reasonably well when the noise is locally stationary (see Fig. 3.1) because in practice these wavelet coefficients represent pure noise [21]. The upward bias due to the presence of a signal at this finest level $\mathcal{D}_0^1$ is addressed by taking the median, instead of taking the standard deviation of the coefficients directly.

We use this metric to calculate a noise map by estimating the standard deviation of the noise at each pixel. The overall map might have "noisy" $\hat{\sigma}_p$ estimates when using a small number of samples. Thus we perform a 3-pixel dilation process on the noise map to smooth it out and fill the holes. This dilation simply replaces each value with the maximum value in its 3×3 neighborhood around it. Before

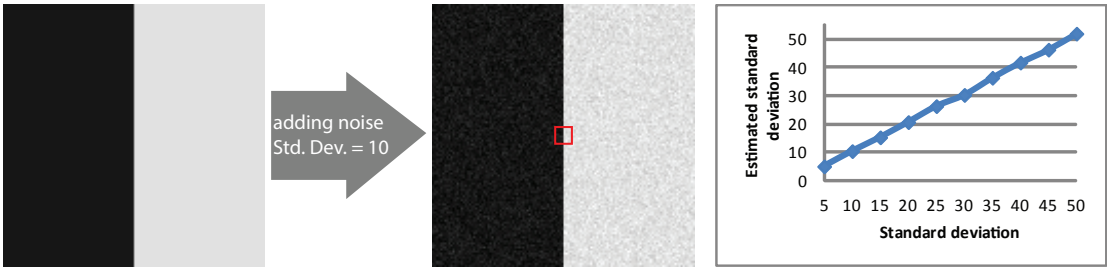


Figure 3.1: To show the effectiveness of the MAD metric for locally stationary noise, we created the test image on the left with half the pixel colors set to 25 and the others set to 225. We then added Gaussian noise with different σ values (from 5 to 50) to the entire image, an example of which is shown in the middle with $\sigma = 10$. The metric in Eq. 3.2 is computed for the pixels in the box shown in red and used to estimate $\hat{\sigma}_p$ where p is the center pixel in the box. The plot shown on the right shows that $\hat{\sigma}_p$ accurately tracks the actual σ after averaging over 100 runs. On the other hand, other metrics commonly used in the graphics community to estimate noise such as variance and contrast [36, 63] would fail in this case because the step edge here affects the results.

we use this noise map with our multilevel algorithm, we scale it by g, a global parameter that defines the smoothness of the results (larger values result in more smoothness). We use a constant g value for all the results shown here.

3.2.2 Multilevel denoising framework

The noise map obtained in the previous section tells us how much each pixel should be denoised by giving us an estimate of the standard deviation of the noise level at every pixel, $\hat{\sigma}_p$. We now discuss how to use spatially-invariant denoising algorithms to remove this varying noise. The naïve way to do this is to repeatedly denoise the entire image using all the different $\hat{\sigma}_p$ values and then select the final pixels from the appropriate denoised results. This is expensive because there are

many different $\hat{\sigma}_p$ values, which would require invoking the denoising algorithm hundreds of thousands of times. Furthermore, trying to accelerate this process by selecting only a small region around each pixel to denoise does not work, since many denoising algorithms like BM3D [16] use information from other parts of the noisy image to denoise a particular block. Therefore, we would still need to process a large region in the image for each $\hat{\sigma}_p$ level to denoise each pixel.

Instead, we propose to accelerate this process by selecting a small subset of standard deviations that best represent the varying noise levels in the entire image. Our basic idea is to apply the denoising filter a small number of times using these representative noise levels, and then select the final pixels from results with similar noise levels.

We propose to find the representative noise levels by dedicating more levels to the standard deviations with more pixels in the noise map. This is similar to importance sampling where samples are placed based on a specific probability density function (pdf) with the goal of sampling more in the high probability regions. To do this, we first compute the cumulative distribution function (cdf) of the noise map $F(\Theta)$ and then invert it using uniform sampling to get a discrete set of levels, as shown in Fig. 3.2. For a desired number of levels L, we compute the standard deviation for each discrete level i as

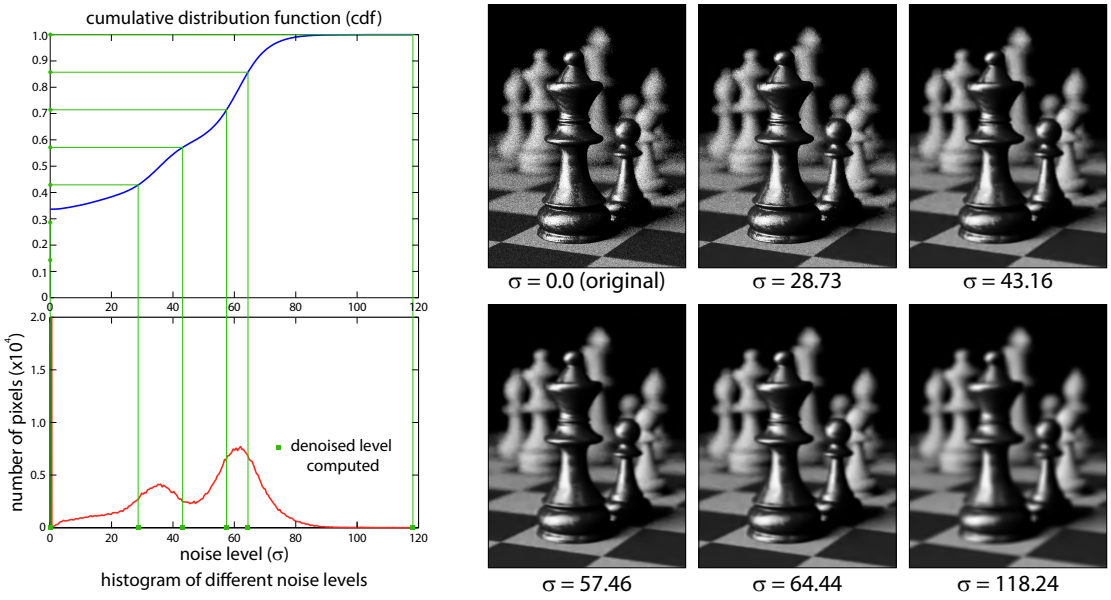


Figure 3.2: On the left, we show how our algorithm decides which noise levels to denoise ($L = 8$). As in standard importance sampling, we uniformly sample the y-axis of the CDF with L samples and determine where the levels lie in terms of σ. This gives us a set of discrete levels that has been selected based on the number of pixels that require certain amount of denoising. On the right, we show the denoised images at each of those levels. Note that three levels (green dots on the vertical axis of the cdf) maps to the same standard deviation of 0 which makes the actual number of levels 6. The individual denoised layers are noisy or blurry which shows that applying a denoising method with a single global σ does not work.

$$\sigma(i) = F^{-1}(\frac{i-1}{L-1}),\ i = 1, \ldots, L. \tag{3.3}$$

Once we have the standard deviation of the noise for each of the L levels, we run the denoising algorithm with each parameter to produce L denoised images: $D_1, \ldots, D_L$. Our last step is to combine these denoised images to obtain our final image. To do this, we simply interpolate the pixel value for each final pixel using

the corresponding values from the two closest noise levels. For example, for the case in Fig. 3.2, if the pixel has $\hat{\sigma}_p = 50$, the denoised images with $\sigma = 43.16$ and $\sigma = 57.46$ would be used to compute its final value. The simple alpha-blending process to compute each pixel in the final image $\hat{c}_p$ can be written as,

$$\hat{c}_p = \alpha \cdot D_k(p) + (1 - \alpha) \cdot D_{k-1}(p),$$

where k is the minimum i that satisfies $\sigma(i) > \hat{\sigma}_p$ and α is defined as,

$$\alpha = \frac{\log[\hat{\sigma}_p] - \log[\sigma(k-1)]}{\log[\sigma(k)] - \log[\sigma(k-1)]},$$

where the logarithm is used to give higher weight to smoother denoised images.

3.2.3 Adaptive sampling using noise estimation

Given the accurate noise metric introduced in Sec. 3.2.1, an obvious extension is to use it to adaptively place samples for reducing the noise. Here, the goal is to place more samples in the noisier regions. In this case, since the variance of the MC estimator decreases linearly with the number of samples, we compute the importance map by estimating the noise variance $\hat{\sigma}_p^2$ at every pixel. Similar to the noise map calculation, we perform a 5-pixel dilation process on the obtained importance map to smooth it out.

Our adaptive sampling algorithm then uses this final importance map to decide where to allocate its budget of samples. To do this, we use an iterative approach

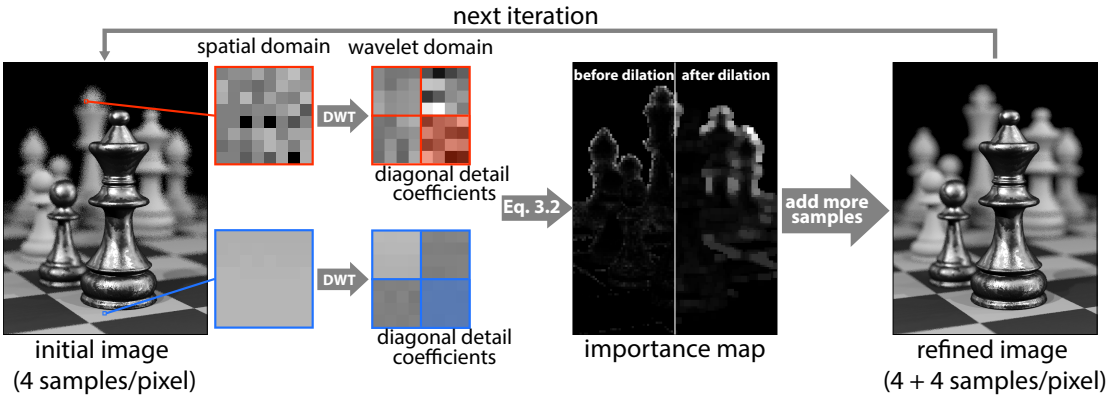


Figure 3.3: Here, we show an overview of our adaptive sampling algorithm. We use a total of three iterations and split the total budget (12 samples per pixel in this case) evenly across these iterations. To initialize our importance map, we first uniformly sample the entire image. The importance map is then calculated using Eq. 3.2 and a 5-pixel dilation is performed to obtain the final importance map. This importance map is used to sample the image adaptively at the next iteration. We perform one more iteration using this scheme to obtain the final rendered image.

that splits the budget evenly across iterations and recomputes the importance map at every iteration. In the first step, we sample the whole scene uniformly to initialize our importance map. After this initialization step, we spend the remaining sample budget across two more iterations. An illustration of our adaptive sampling process is shown in Fig. 3.3.

3.3 Results

We implemented the adaptive sampling method in C++ and integrated it into PBRT2 [73]. We used MATLAB to write the rest of the proposed framework and

ran the adaptive sampling part directly from this MATLAB program. Furthermore, we wrote the MAD computation in CUDA for GPU acceleration. Note that in our current implementation we made a few changes to the original paper [47] for speed and quality. To test a couple of image denoising algorithms within our framework, we ran our experiments on the BLS-GSM [75] and BM3D [16] denoising algorithms mentioned earlier. We used the code provided by the authors of the BLS-GSM algorithm. For BM3D, we used the implementation by Yun-Ta et al. [98] which utilizes a fast approximate nearest neighbor search strategy to speed up the patch search process with a GPU. Since the core of BM3D is based on non-local means (NLM) [7] filtering, we also show the results of NLM using Yun-Ta et al.'s implementation for completeness. We ran all the experiments with the standard default parameters provided by the authors.

To handle color images, we compute the noise map and the importance map for each channel separately and then average them before computing the final noise/importance map value. We also tried applying weights based on the perceptual importance of each color channel (more weight to the green and less weight to the blue channel), but the difference in the results was not noticeable. Furthermore, we tonemap the sample values to the range 0 to 1 before computing these terms because most denoising algorithms work on standard images rather than raw high dynamic range images. As for the parameters of our algorithm, we use

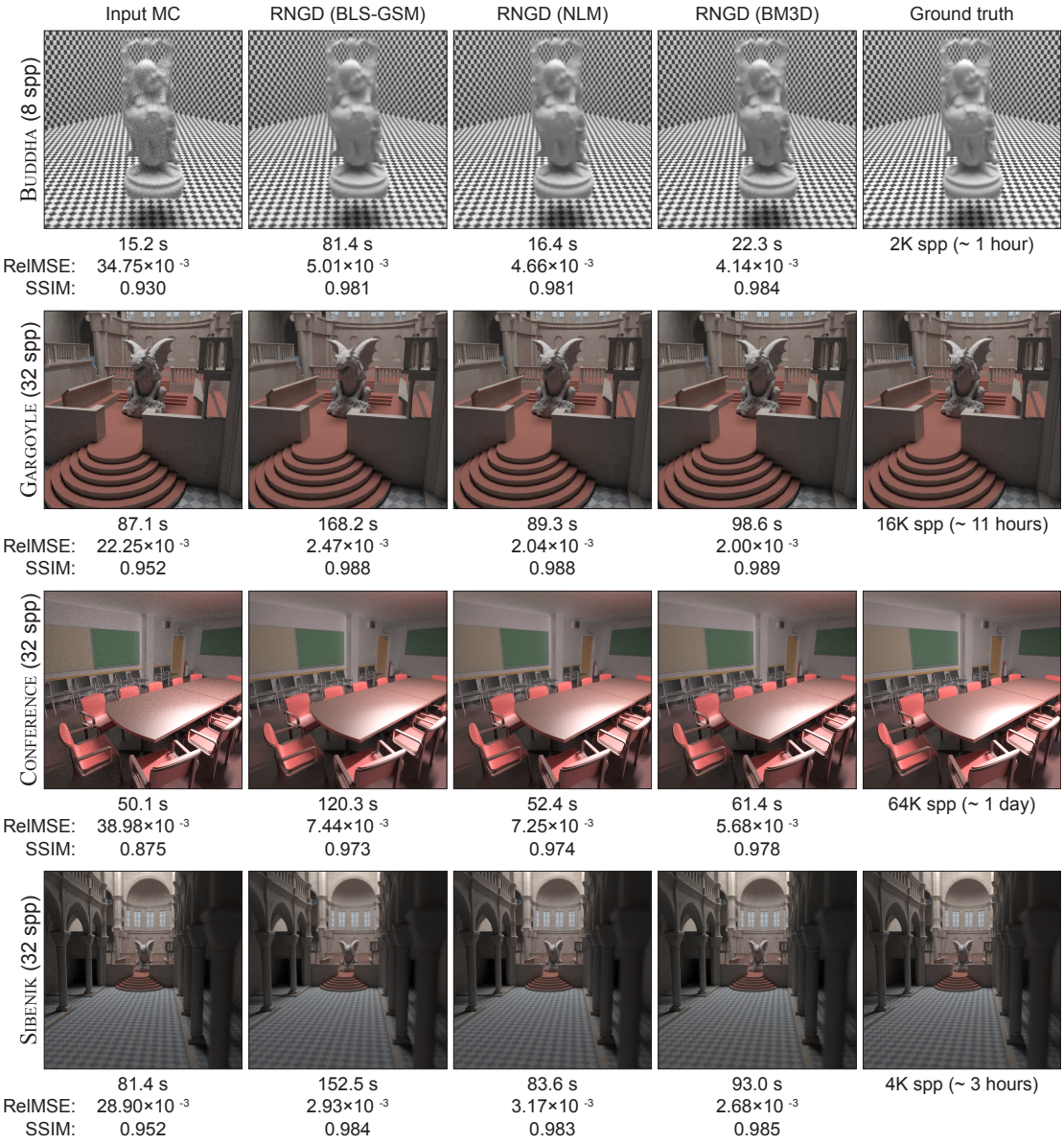


Figure 3.4: We show the result of our framework (RNGD) with different denoising algorithms. We list the relative MSE and SSIM index values (for SSIM higher is better) below each image for quantitative comparison. Although all the denoising methods produce high-quality results, BM3D provides the best tradeoff between quality and speed.

three denoising levels ($L = 3$) and a smoothness parameter (Sec. 3.2.1) $g = 1$ for BLS-GSM and $g = 2$ for NLM and BM3D. This parameter, g, is kept constant for all scenes and there are no other parameters in our code.

All the results shown here were computed on an Intel quad-core 3.4 GHz machine with 24 GB of memory and a GeForce GTX 760 GPU. We compare our approach, called removing noise with general denoising (RNGD), against three state-of-the-art color-based approaches for filtering general MC noise: adaptive sampling and reconstruction (ASR) [80], non-local means filtering (NLM) [81], and ray histogram fusion (RHF) [19]. For all these algorithms, we used the implementations provided by the respective authors. For quantitative comparison, we use relative MSE [80]: $(\hat{c} - c)^2/(c^2 + \varepsilon)$, where $\hat{c}$ and c are the filtered and ground truth images, respectively, and ε is a small number (0.01) to avoid division by zero. We also provide the structural similarity (SSIM) index [105] to measure the perceptual quality of the images. SSIM is a value between 0 and 1, where 1 indicates perfect quality with respect to the ground truth image.

In Fig. 3.4, we begin by showing the result of our framework using three denoising algorithms (BLS-GSM, NLM, and BM3D) on scenes with a variety of distributed effects such as global illumination, depth of field, and area lighting. As can be seen, all of the denoising methods in our framework produce smooth, high-

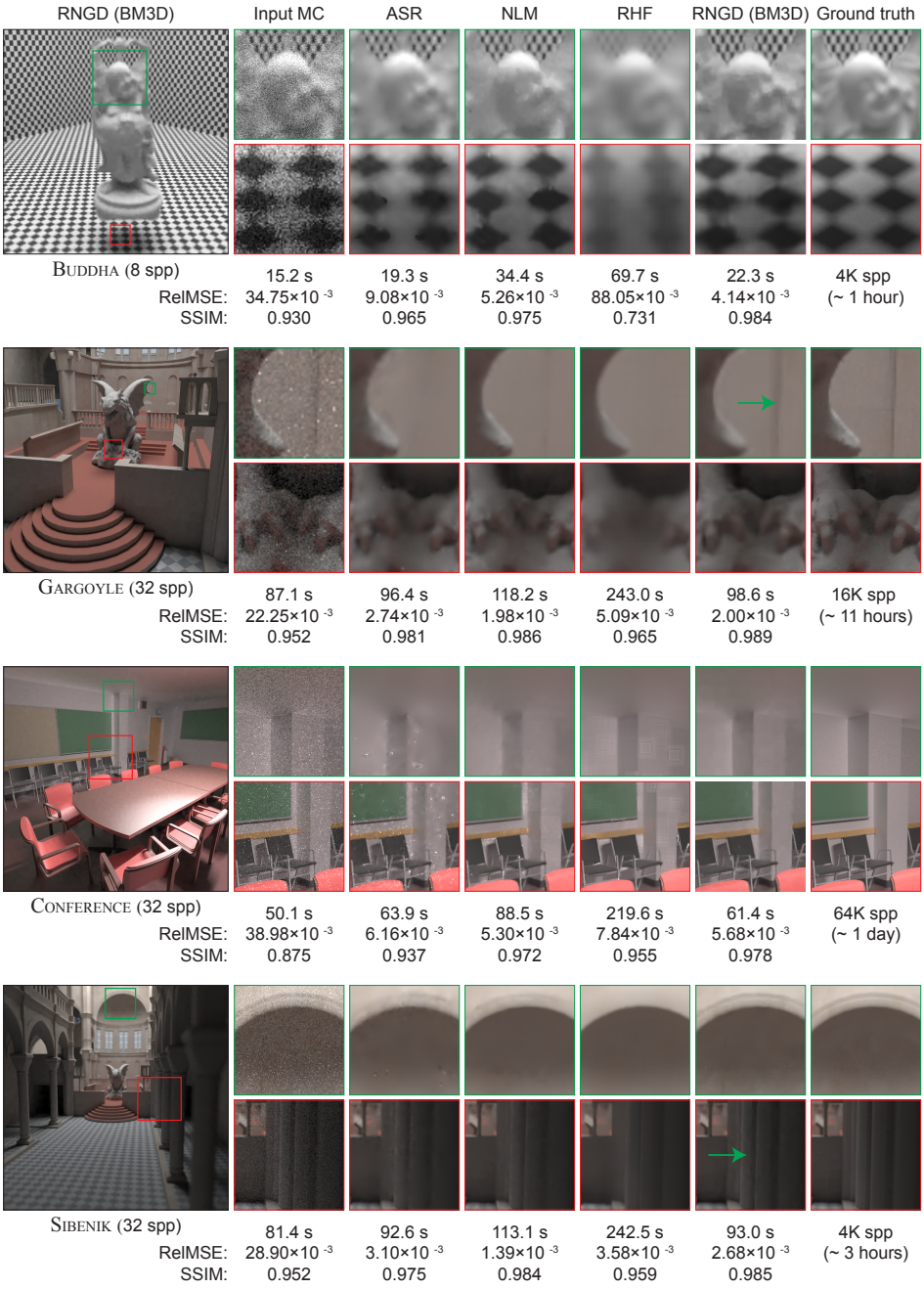


Figure 3.5: We show equal-sample comparisons of our approach (RNGD) with BM3D against several state-of-the-art color-based methods. Our method is faster than most other approaches and preserves the structures and texture details better because it uses a more sophisticated denoising algorithm.

quality results. However, we found BM3D to provide the best tradeoff between quality and speed and, therefore, we use it in the remaining comparisons.

Next, we show equal-sample comparisons of our method with BM3D against several state-of-the-art algorithms in Fig. 3.5. Note that, like our technique, these approaches use only pixel color information. Moreover, although our method is faster than other techniques in most cases, we show equal-sample comparisons to demonstrate the effectiveness of using a more advanced denoising algorithm (BM3D) in our framework. As seen, our approach can keep the structures and textures better than the other color-based approaches.

Finally, in terms of timing, our algorithm is reasonably fast, taking on the order of seconds to denoise an image. For example, our approach with BM3D removes noise from a 1024×1024 image in roughly 12 seconds, where less than 1 second is spent on computing the noise map and representative noise levels and the rest (11 seconds) is dedicated to denoising the $L = 3$ levels.

3.4 Discussion

The flexibility and simplicity of our framework allows us to use state-of-the-art video denoising methods to remove the noise from Monte Carlo renderings of animated sequences. In the supplemental video, we use our framework with the

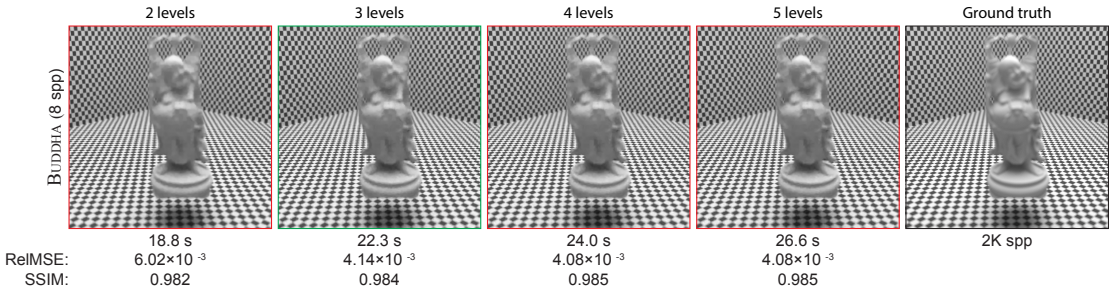


Figure 3.6: Here we vary the number of levels L for denoising, as described in Sec. 3.2.2. There is a significant difference in quality between 2 levels and 3, but afterwards there is not much improvement. Therefore, we chose $L = 3$ levels (indicated by green border) to produce all the results since it provides the best trade-off between quality and speed.

BM3D video denoising method [15] to produce a noise-free result on the Chess scene (800×600) rendered at 16 samples per pixel. Our method takes less than 10 seconds per frame (excluding the rendering time) and produces a temporally-coherent high-quality result.

We also experimented with varying the number of noise levels, L, computed during our multilevel denoising process (see Sec. 3.2.2). Fig. 3.6 shows the result of varying this parameter, which provides a tradeoff between quality and speed. In practice, we found that $L = 3$ levels were sufficient for high-quality results.

Similar to other color-based approaches, the main limitation of our algorithm is that it does not utilize additional scene features such as world positions and shading normals, which are available from the MC rendering system. As a result, it produces suboptimal results in challenging cases, such as the one shown in Fig. 3.7. Here, we compare our approach against other color-based techniques and our

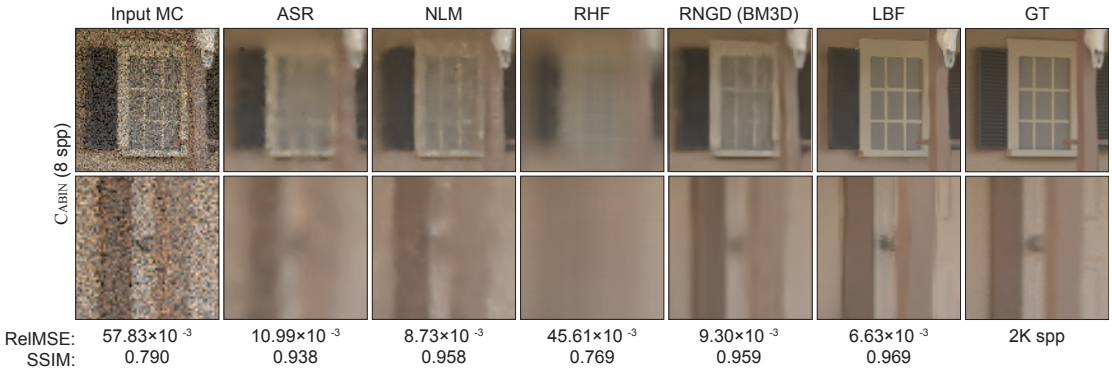


Figure 3.7: We compare the result of our approach (RNGD) with several color-based schemes and our proposed learning-based filtering (LBF) method, which will be explained in Chapter 5. Although we produce better results than the other three color-based methods (ASR, NLM, and RHF), our image has lower quality than LBF since we only use the pixel colors in the filtering process.

learning-based filtering (LBF) method (Chapter 5) that uses additional features. Although our result is comparable or better than the other color-based schemes, it has lower quality than LBF's result since our approach does not utilize the additional scene features. In the next chapter, we present a new way of looking at the filtering problem to be able to develop algorithms that systematically address this challenging problem.

3.5 Conclusion

Although powerful denoising algorithms have been developed over the years by the image processing community, they cannot be directly used in rendering because of their inherent assumption of spatially-invariant noise. In this chapter,

we presented a way to use general spatially-invariant denoising algorithms by first estimating the per-pixel noise level followed by a multilevel algorithm that applies the denoising method in a spatially-varying manner. We showed that this approach produces better results than the other state-of-the-art color-based techniques. However, even using the most advanced image denoising techniques, this approach cannot handle complex scenes with severe noise as it does not utilize additional scene features such as world positions and shading normals.

Chapter 4

Reformulating the MC filtering problem

In order to address the filtering problem in a systematic way, we first need to identify the main challenges. Therefore, in this chapter, we present a new and general way of looking at the MC filtering problem and study the previous filtering approaches from this new perspective. Based on this analysis, we can identify the main drawbacks of previous approaches and have a better understanding of how to address these shortcomings.

In general, the filtered image $\hat{\mathbf{c}} = \{\hat{c}_r, \hat{c}_g, \hat{c}_b\}$ at pixel p is computed as a weighted average of all of the pixels in a square neighborhood $\mathcal{N}(p)$ centered around pixel p:

$$\hat{\mathbf{c}}_p = \frac{\sum_{q \in \mathcal{N}(p)} d_{p,q} \bar{\mathbf{c}}_q}{\sum_{q \in \mathcal{N}(p)} d_{p,q}}, \tag{4.1}$$

where $d_{p,q}$ is the weight between pixel p and its neighbor q as defined by the filter and $\bar{\mathbf{c}}_q$ is the noisy pixel color computed by averaging all the sample colors in pixel q. A filter usually takes a collection of mean *primary features*[1], $\bar{\mathbf{s}}_{\mathcal{N}(p)}$, in the neighborhood of the p^{th} pixel to define the weights, $d_{p,q}$, and filter the image.

The ultimate goal of *all* post-process filtering algorithms for MC noise is to find a filter $\hat{h}$ at each pixel that produces a filtered result as close as possible to the ground truth. Formally, this can be written as,

$$\hat{c}_p = \hat{h}(\bar{\mathbf{s}}_{\mathcal{N}(p)}), \quad \text{where} \quad \hat{h} = \arg\min_h E(h(\bar{\mathbf{s}}_{\mathcal{N}(p)}), c_p), \tag{4.2}$$

where E is the error metric, which computes the distance between the potential filtered pixel color $h(\bar{\mathbf{s}}_{\mathcal{N}(p)})$ and the ground truth c_p. In this case, the filter is characterized by a set of weights, $d_{p,q}$ where $q \in \mathcal{N}(p)$, and the goal of optimization is to find these weights at each pixel. For example, for a neighborhood of size 55×55, we need to compute $55 \times 55 = 3025$ optimal weights at each filter.

Although this is the most general form of presenting the problem, performing this minimization over the space of all possible weights is difficult. Therefore, filtering approaches consider a general form for the filter, which can be characterized by a set of M filter parameters, $\boldsymbol{\theta}$. For example, for a standard Gaussian

[1]In this thesis, the term "primary features" refers to scene features that are computed directly by the rendering system when shading samples. This includes sample positions, colors, and K scene features such as world positions, shading normals, and texture values. The word "mean" refers to the fact that the features of every sample in a pixel are averaged together.

filter, $d_{p,q}$ would be the Gaussian-weighted distance between pixels p and q in the spatial domain and it has only one parameter ($M = 1$) which defines the variance of the Gaussian filter. Another example is the more sophisticated cross-bilateral filter [29, 72, 97] which can leverage additional scene features (e.g., world positions, shading normals, etc.) to improve the quality of the filtered results. Here, $d_{p,q}$ is given by[2]:

$$d_{p,q} = \exp\left[-\frac{\|\bar{\mathbf{p}}_p - \bar{\mathbf{p}}_q\|^2}{2\alpha_p^2}\right] \times \exp\left[-\frac{\|\bar{\mathbf{c}}_p - \bar{\mathbf{c}}_q\|^2}{2\beta_p^2}\right] \times \prod_{k=1}^{K} \exp\left[-\frac{\|\bar{\mathbf{f}}_{p,k} - \bar{\mathbf{f}}_{q,k}\|^2}{2\gamma_{k,p}^2}\right], \quad (4.3)$$

where $\bar{\mathbf{p}}_p$, $\bar{\mathbf{c}}_p$, and $\bar{\mathbf{f}}_{p,k}$ refer to pixel p's screen space position, color, and scene feature k, respectively, and α_p^2, β_p^2, and $\gamma_{k,p}^2$ are the variances at pixel p for the spatial, color, and k^{th} feature terms. Therefore, in this case, the filter parameters are the collection of all these parameters, i.e., $\boldsymbol{\theta} = \{\alpha^2, \beta^2, \gamma_1^2, \cdots, \gamma_K^2\}$. Using a general form for the filter, Eq. 4.2 can be written as:

$$\hat{c}_p = h(\bar{\mathbf{s}}_{\mathcal{N}(p)}, \hat{\boldsymbol{\theta}}_p), \quad \text{where} \quad \hat{\boldsymbol{\theta}}_p = \arg\min_{\boldsymbol{\theta}_p} E(h(\bar{\mathbf{s}}_{\mathcal{N}(p)}, \boldsymbol{\theta}_p), c_p). \quad (4.4)$$

Thus, in this case, the dimensionality of the problem is reduced from optimizing 3025 weights to M filter parameters (e.g., 7 parameters) at each pixel. However, finding the optimal filter parameters at each pixel is still difficult as the

[2]Note that this is a simpler version of the cross-bilateral filter which we use to present the problem. In Chapter 5, we explain a more complicated version of this filter used in our implementation.

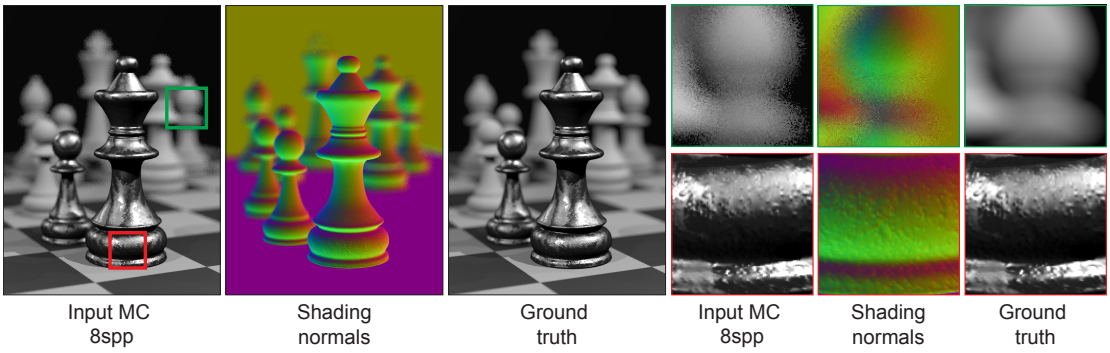


Figure 4.1: We show the CHESS scene rendered at 8 samples per pixel as well as the visualization of the shading normals and the ground truth image on the left. To remove noise from the out-of-focus region (green), we need to ignore the noisy shading normals and, thus, require a wide filter. On the other hand, to preserve the high frequency details on the chess piece on the in-focus region (red), we need to rely on the shading normals, requiring a tight filter. This example shows that we need to vary the filter parameters across the image to produce high-quality results.

ground truth image is not available at run-time. Note that these parameters need to be set correctly at each pixel independently to produce high-quality results. For example, as shown in Fig. 4.1, the high frequency details in the out-of-focus region are MC noise and to remove it, we need to ignore the noisy shading normals and, thus, require a wide filter. On the other hand, to preserve the scene detail on the in-focus region, we need to rely on the shading normal, requiring a tight filter. Therefore, there is not a single filter parameter that works for all these different regions.

Therefore, the main challenge in all MC filtering techniques is estimating the optimal filter parameters without having access to the ground truth image. In the

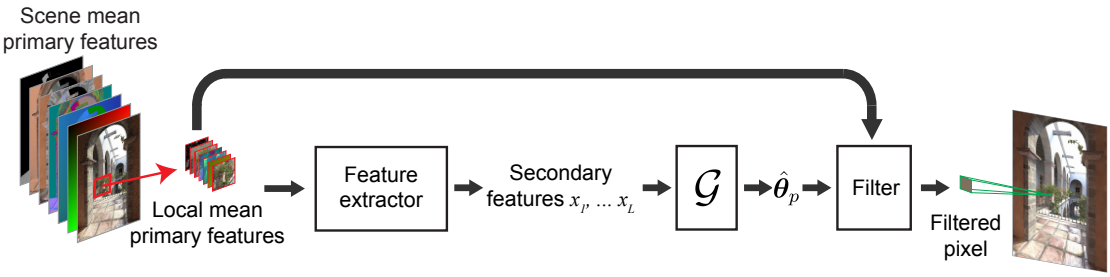


Figure 4.2: The direct estimation approaches attempt to directly estimate the filter parameters from the local primary features. They first extract a set of secondary features $\{x_1, \cdots, x_L\}_p$ from these mean primary features in a local neighborhood around each pixel. These features are then combined through a function, $\mathcal{G}$, to estimate the filter parameters, $\hat{\boldsymbol{\theta}}_p$. The estimated filter parameters are then used by the filter to produce the filtered pixel.

next section, we give a brief overview of the previous general filtering approaches and discuss their main drawback.

4.1 Filter parameter estimation

This way of presenting the MC filtering problem gives us a new way of classifying the previous filtering approaches based on how they estimate the filter parameters. We observe that all the previous general MC filtering approaches can be categorized into two general classes of direct estimation and error-minimization. We explain each in detail in the following sections.

4.1.1 Direct estimation

As shown in Fig. 4.2, these methods directly estimate the filter parameters at each pixel from the primary features. To do so, they first process the noisy primary features in a pixel's neighborhood to generate a set of more meaningful data, which we call *secondary features* $\mathbf{x}_p = \{x_1, x_2, \cdots, x_L\}_p$. Previous approaches in this category have proposed a variety of secondary features including mutual information [90], dual buffer variance [81], and median absolute deviation (MAD) [47] (discussed in Chapter 3). These secondary features are then usually combined through a function, $\mathcal{G}$, to approximate the filter parameters at each pixel as follows:

$$\hat{\boldsymbol{\theta}}_p = \mathcal{G}(\mathbf{x}_p). \tag{4.5}$$

The function $\mathcal{G}$, describes the relationship between the secondary features and the filter parameters and each method has a specific way of defining it. For example, Sen and Darabi [90] normalized the computed mutual information to obtain filter parameters while Rousselle et al. [81] and Kalantari and Sen [47] defined this function to be identity.

The main drawback of these approaches is that the estimated filter parameters are not guaranteed to minimize the error between the filtered and ground truth images. Moreover, these methods require us to define this complicated function

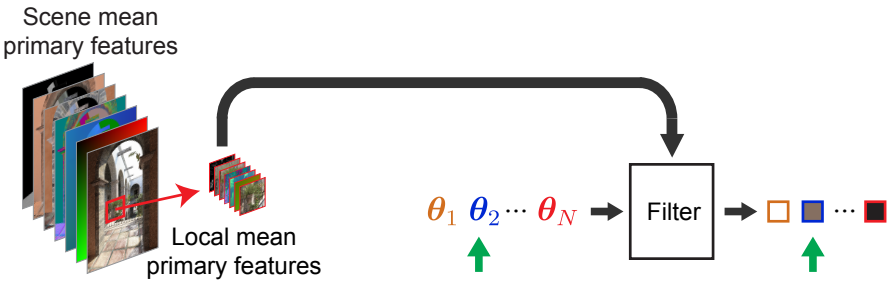


Figure 4.3: The error minimization approaches efficiently minimize the error between the filtered and ground truth images by filtering the noisy input image with a set of N candidate filter parameters at each pixel, $\boldsymbol{\theta}_1, \cdots, \boldsymbol{\theta}_N$. They then estimate the error of each filtered pixel using a no-reference error estimation metric and choose the one with smallest error.

$\mathcal{G}$ to set the filter parameters at each pixel. However, previous approaches define this function heuristically. In order to address these issues, we propose to learn this function $\mathcal{G}$ during training on a set of scenes (Chapter 5). Once we obtain the optimal $\mathcal{G}$, we can use it to estimate the optimal filter parameters and, consequently, remove MC noise from any new test scenes.

4.1.2 Error-minimization

The error minimization techniques attempt to directly minimize the error between the filtered and ground truth images (see Eq. 4.4). However, since the ground truth image is not available at run-time, they use a no-reference error estimation metric to approximate the actual error. Therefore, instead of optimizing Eq. 4.4, these methods perform the following optimization:

$$\hat{c}_p = h(\bar{\mathbf{s}}_{\mathcal{N}(p)}, \hat{\boldsymbol{\theta}}_p), \quad \text{where} \quad \hat{\boldsymbol{\theta}}_p = \arg\min_{\boldsymbol{\theta}_p} \hat{E}(h(\bar{\mathbf{s}}_{\mathcal{N}(p)}, \boldsymbol{\theta}_p)), \tag{4.6}$$

where $\hat{E}$ is the no-reference error estimation metric. Previous approaches have proposed to use different error estimation metrics such as bias and variance [80, 66] and Stein's unbiased risk estimator [56, 82]. To efficiently perform the above minimization, these methods filter the image several times using a set of N candidate filter parameters, $\boldsymbol{\theta}_1, \cdots, \boldsymbol{\theta}_N$, as shown in Fig. 4.3. They then select the filtered pixel with the lowest estimated error[3] at each pixel to reconstruct the final filtered image.

Although these approaches attempt to minimize the error, they use a set of heuristically selected candidate filter parameters and, thus, do not utilize the full potential of the framework. We mitigate this problem in Chapter 6 by optimizing the candidate filter parameters on a set of training scenes. Once the set of N candidate filter parameters are obtained, we can use them to remove noise from any new noisy test scenes.

[3]In practice, since the error estimates are usually noisy, choosing the filter with the lowest error is not always optimal. Therefore, these methods usually use a more sophisticated approach that reconstructs the final image using the N filtered images and their corresponding error. We describe such approaches in more detail in Chapter 6.

Chapter 5

Learning-based Monte Carlo filtering

As discussed in Chapter 4, the main challenge in MC filtering approaches is estimating the optimal filter parameters. The direct estimation approaches attempt to estimate the filter parameters directly from the primary features (output of the MC renderer). They do so by first processing the primary features to extract a set of secondary features. They then combine these features through a function to estimate the filter parameters. However, the main drawbacks of previous techniques is that they define this function heuristically and the estimated filter parameters are not guaranteed to minimize an error with respect to the ground truth.

Our goal in this chapter is to address these problems with the previous direct estimation approaches. Our key observation is that there is a complex relationship between the input noisy data and the optimal filter parameters and, thus, this function is difficult to define explicitly. Moreover, these filter parameters can be

effectively estimated using different factors (e.g., feature variances and noise in local regions), but each individual factor by itself, as used in previous approaches, cannot accurately predict them. Based on these observations and to avoid the problems of previous approaches, we propose a *supervised learning* algorithm that learns the complex relationship between the aforementioned factors (e.g., feature variances and noise in local regions) and the optimal filter parameters. To do this, we train a nonlinear regression model on a set of noisy MC rendered images and their corresponding ground truth images using a multilayer perceptron (MLP) tightly coupled with a matching filter during both training and testing.

During the training stage, we render both the noisy images at low sampling rates as well as their corresponding ground truth images for a set of scenes with a variety of distributed effects. The noisy images are then processed to extract a set of secondary features in square regions around every pixel. Finally, the proposed network is trained on these features to drive the filter to produce images that resemble the ground truth according to a specific error metric. Once the network has been trained, we use it in the test stage to filter new scenes with general distributed effects. Our method takes only a few seconds, yet produces better results than existing methods for a wide range of distributed effects including depth of field, motion blur, area lighting, glossy reflections, and global illumination.

Since we use a neural network in our system, we give a brief summary of previous work using neural networks in computer graphics and image denoising in the next section.

5.1 Neural networks in graphics/denoising

Grzeszczuk et al. [35] used neural networks to create physically realistic animation. Nowrouzezahrai et al. [68] used them to predict per vertex visibility. Dachsbacher [17] classified different visibility configurations using neural networks. Ren et al. [78] used a neural network to model the radiance regression function to render indirect illumination of a fixed scene in real time. On the other hand, neural networks have also been used in image denoising where they have been directly trained on a set of noisy and clean patches [8]. Finally, it is worth noting that Jakob et al.'s method [40], while not utilizing neural networks, performs learning through expectation maximization to find the appropriate parameters of a Gaussian mixture model to denoise photon maps, a different but related problem. To the best of our knowledge, our method is the first to use neural networks (or even machine learning, for that matter) for MC noise reduction.

5.2 A new learning framework for MC filtering

The goal of our approach is to take a noisy input image rendered with only a few samples and generate a noise-free image that is similar to the ground truth rendered with many samples. As discussed in Chapter 4, the filter outputs a filtered pixel by computing a weighted average of all the noisy pixels in a square neighborhood around the pixel of interest (see Eq. 4.1). The weight between the pixel of interest, p, and its neighbor q is given by $d_{p,q}$ and is defined by the filter. For the cross-bilateral filter used in our implementation, $d_{p,q}$ is given by:

$$d_{p,q} = \exp\left[-\frac{\|\bar{\mathbf{p}}_p - \bar{\mathbf{p}}_q\|^2}{2\alpha_p^2}\right] \times \exp\left[-\frac{D(\bar{\mathbf{c}}_p, \bar{\mathbf{c}}_q)}{2\beta_p^2}\right] \times \prod_{k=1}^{K} \exp\left[-\frac{D_k(\bar{\mathbf{f}}_{p,k}, \bar{\mathbf{f}}_{q,k})}{2\gamma_{k,p}^2}\right]. \quad (5.1)$$

Here, D and D_k are specific distance functions for colors and scene features (see Sec. 5.2.2). Note that here we use the cross-bilateral filter to explain our algorithm, but our framework can be extended to use other differentiable filters in a similar way, as we discuss later.

Since the filter parameters α_p, β_p, and $\gamma_{k,p}$ control the widths for this filter, the challenge is how to find the values for each to produce the highest-quality results possible. As discussed in Chapter 4, we can write the filtering process more generally as:

$$\hat{\mathbf{c}}_p = h(\bar{\mathbf{s}}_{\mathcal{N}(p)}, \boldsymbol{\theta}_p), \quad \text{where} \quad \bar{\mathbf{s}}_{\mathcal{N}(p)} = \bigcup_{q \in \mathcal{N}(p)} \bar{\mathbf{s}}_q, \quad (5.2)$$

where $\bar{\mathbf{s}}_{\mathcal{N}(p)}$ is the collection of mean primary features in the neighborhood of the p^{th} pixel, h is the filter function which implements Eq. 4.1, and $\boldsymbol{\theta}_p$ is an array of M filter parameters at pixel p. For example, for the cross-bilateral filter shown in Eq. 5.1, $\boldsymbol{\theta}_p$ has $M = K + 2$ parameters ($\boldsymbol{\theta}_p = \{\alpha, \beta, \gamma_1, \cdots, \gamma_K\}_p$) corresponding to the standard deviations for each component of the filter.

The direct estimation approaches process the noisy mean primary features in a pixel's neighborhood to generate a set of more meaningful data (secondary features), $\mathbf{x}_p = \{x_1, x_2, \cdots, x_L\}_p$. These secondary features include feature variances, noise approximation in local regions, and so on. They then approximate the filter parameters through a function $\mathcal{G}$ of the secondary features, as shown in Eq. 4.5. We observe that function $\mathcal{G}$, which attempts to approximate the relationship between the secondary features and the optimal filter parameters, is complicated and difficult to model explicitly. For this reason, we propose to find this function with a learning system that performs the following energy minimization on training data:

$$\mathcal{G}^* = \arg\min_{\mathcal{G}} E(h(\bar{\mathbf{s}}_{\mathcal{N}(p)}, \mathcal{G}(\mathbf{x}_p)), \mathbf{c}_p), \tag{5.3}$$

where optimizing this energy function requires finding a $\mathcal{G}$ that produces a filtered result close to the ground truth image according to a specific error function E. Once we minimize this energy function to find an approximate $\hat{\mathcal{G}}$, we can compute

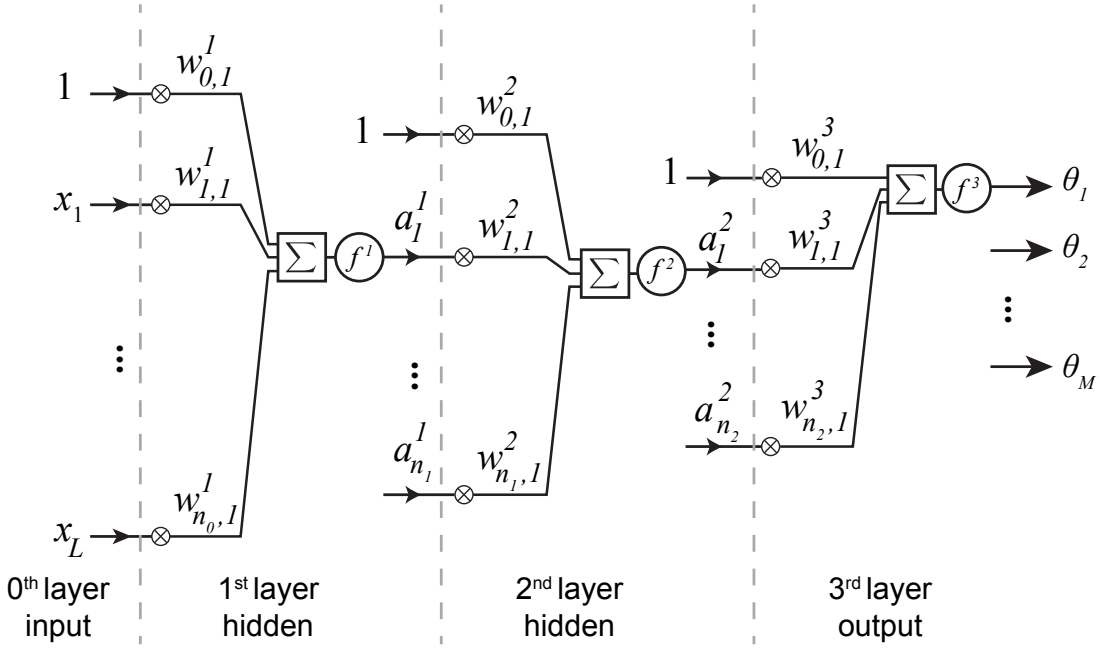


Figure 5.1: A multilayer perceptron (MLP) consists of multiple layers, each with several nodes. The nodes at each layer are fully connected to the nodes in the next layer through a set of weights, where weight $w^l_{t,s}$ connects node t in layer $l-1$ to node s at layer l, and $w^l_{0,s}$ is the bias for the s^{th} node at layer l. The output of each node is computed using Eq. 5.4. For clarity, we only show connections for one node at each layer. In our implementation, we used a neural network with one hidden layer of size 10 to model $\mathcal{G}$ in Eq. 4.5.

the filter parameters $\hat{\boldsymbol{\theta}}_p = \hat{\mathcal{G}}(\mathbf{x}_p)$ that will be used in the filter to generate a filtered image that is close to the ground truth for a new test scene.

Therefore, we approximate $\mathcal{G}$ in a systematic way with a learning system that directly minimizes the error in Eq. 5.3. Specifically, we represent $\mathcal{G}$ using a nonlinear regression model based on a neural network (see Fig. 5.1) and directly combine it with a matching filter during training and testing (see Fig. 5.2). Note that since the ground truth images are available during training, we can directly compute

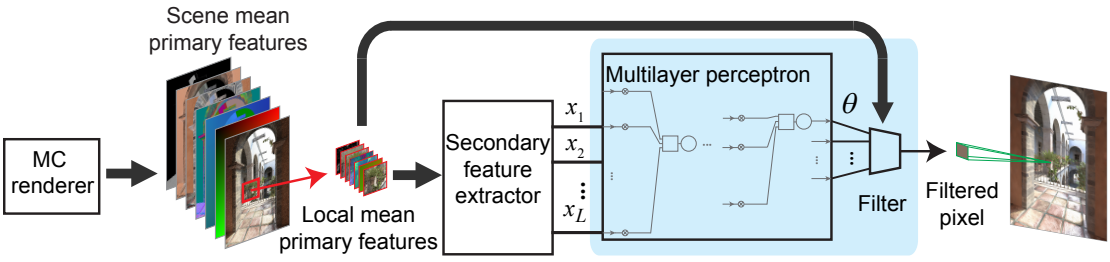


Figure 5.2: Our approach combines a standard MLP (Fig. 5.1) with a matching filter. The local mean primary features (illustrated by a stack of images) contain color, position, and additional features such as world positions, shading normals, etc. A set of secondary features $\{x_1, \cdots, x_L\}_p$ (see Sec. 5.2.3) are extracted from the mean primary features in a local neighborhood of each pixel. The MLP takes the secondary features and outputs the parameters of the filter. The filter then takes the block of mean primary features and the parameters to outputs a filtered pixel. During training, we minimize the error between the filtered pixel and the ground truth. Once trained, the network can generate appropriate filter parameters for an arbitrary test image.

the error between the filtered and ground truth image without the need for error estimation. At test time, we then use the trained $\hat{\mathcal{G}}$ on the secondary features from *new* scenes to compute filter parameters that will allow the filter to produce results close to the ground truth.

We now describe how to train a neural network in combination with the filter by minimizing the energy function from Eq. 5.3.

5.2.1 Our proposed neural network

As with any machine learning system, there are three crucial elements we must decide: 1) the choice of model for representing $\mathcal{G}$, 2) an appropriate error metric,

E, to measure the distance between the filtered and ground truth images, and 3) an optimization strategy to minimize the energy function in Eq. 5.3.

For our representation, we use a neural network, formally a multilayer perceptron (MLP), as a regression model since it is a simple, yet powerful system for discovering complex nonlinear relationships between its inputs and outputs. Moreover, MLPs are inherently parallel and can be efficiently implemented on a GPU. However, our proposed approach differs from standard MLPs in that we have incorporated the filter into the training process, which requires us to "back-propagate" through it to update the weights of the network during training. This implies that the filter must be differentiable with respect to its filter parameters. Fortunately, common filters such as the Gaussian, cross-bilateral, and cross non-local means filters are all differentiable and can be applied in our system.

As shown in Fig. 5.1, MLPs consist of multiple layers known as the input, hidden, and output layers. Each layer has several nodes which are fully connected to all nodes in the next layer through weights. The output of a certain node is a function of the weighted sum of the outputs of the nodes from the previous layer plus an additional bias term used as an offset. Specifically, the output of the s^{th} node at the l^{th} layer is given by:

$$a_s^l = f^l \left(\sum_{t=1}^{n_{l-1}} w_{t,s}^l {a_t}^{l-1} + w_{0,s}^l \right), \tag{5.4}$$

where n_{l-1} is the number of nodes in layer $l-1$, $w^l_{t,s}$ is the weight associated with the connection between node t in layer $l-1$ and node s in layer l, $w^l_{0,s}$ is the bias for this node, and f^l is the activation function for layer l. If we use linear activation functions (e.g., $f^l(x) = x$) in all layers, the neural network becomes a simple linear regression model. Therefore, we typically use nonlinear activation functions, such as the sigmoid function $f^l(x) = 1/(1+e^{-x})$. See Sec. 5.3.1 for a complete description of the actual MLP architecture we used in this work.

Next, we describe our metric to measure the error between the filtered and ground truth pixel values. Rather than using standard MSE, we use the relative mean squared error (RelMSE) metric proposed by Rousselle et al. [80] which is better suited for our application, since the human visual system is more sensitive to color variations in darker regions of the image. However, we modify relative MSE slightly to get:

$$E_p = \frac{n}{2} \sum_{j \in \{r,g,b\}} \frac{\left(\hat{c}_{p,j} - c_{p,j}\right)^2}{c^2_{p,j} + \varepsilon}, \tag{5.5}$$

where n is the number of samples per pixel, $\hat{c}_{p,j}$ and $c_{p,j}$ are the j^{th} color channel of the filtered and ground truth pixels, respectively, and ε is a small number (0.01 in our implementation) to avoid division by zero. Here, the division by $c^2_{p,j}$ accounts for the aforementioned perceptual effect by giving higher weight to the regions where the ground truth image is darker.

Our modification to Rousselle et al.'s original relative MSE metric is the multiplication by $n/2$. Since the squared error decreases linearly with the number of samples in MC rendering, images with fewer samples have greater error than images with more samples. This would make our learning system biased in favor of training images with lower sampling rates. By multiplying the squared error by n, we cancel this inverse relationship and force all of the images to have an equal contribution to the error regardless of their sampling rate. Furthermore, the error is divided by 2 to produce a simpler derivative (see Eq. 5.7). Note that our proposed metric is different than the commonly-used error metrics in the neural network community, such as mean squared error (MSE) and cross-entropy error.

Finally, we describe our optimization strategy to train the network. To do this, we first need a large set of input data and the corresponding ground truth images, which can be generated prior to training. For each noisy image, we extract a set of secondary features at each pixel (see Section 5.2.3). With this data in hand, we can then train the network through an iterative, three-step process called *backpropagation* [83]. The goal of this process is to find the optimal weights $w_{t,s}^{l}$ for all nodes in the network that minimize the error between the computed and desired outputs (i.e., ground truth values) for all pixels in the training images, $E = \sum_{p \in \text{all pixels}} E_p$.

Before starting the backpropagation process, the weights are randomly initialized to small values around zero (between -0.5 to 0.5 in our implementation). Then in the first step, known as the *feed-forward* pass, the output of the network is computed using all inputs. This can be implemented efficiently using a series of matrix multiplications and activation functions applied to the input data to evaluate Eq. 5.4. In the second step, the error between the computed and desired outputs is used to determine the effect of each weight on the output error. This requires taking the derivative of the error with respect to each weight $\partial E/\partial w^l_{t,s}$. Thus, the activation functions (and in our case, the filter as well) need to be differentiable. These two steps are performed for all of the data in the training set and the error gradient of each weight is accumulated. Finally, in the third step, all the weights are updated according to their error gradient. This completes a single iteration of training, known as an *epoch*, and typically many epochs are needed to properly train the system and obtain a converged set of weights.

We now examine the differentiability of our system as required by backpropagation. Using the chain rule, we express the derivative of the energy function from Eq. 5.5 with respect to the weights $w^l_{t,s}$ as:

$$\frac{\partial E_p}{\partial w^l_{t,s}} = \sum_{m=1}^{M} \left[\sum_{j \in \{r,g,b\}} \left[\frac{\partial E_p}{\partial \hat{c}_{p,j}} \frac{\partial \hat{c}_{p,j}}{\partial \theta_{m,p}} \right] \frac{\partial \theta_{m,p}}{\partial w^l_{t,s}} \right], \tag{5.6}$$

where M is the number of filter parameters. The first term is the derivative of the error with respect to the filtered pixels $\hat{c}_{p,j}$, which can be easily calculated as follows:

$$\frac{\partial E_p}{\partial \hat{c}_{p,j}} = n \frac{\hat{c}_{p,j} - c_{p,j}}{c_{p,j}^2 + \varepsilon}. \tag{5.7}$$

In addition, the $\theta_{m,p}$'s are the outputs of a standard MLP network (see Fig. 5.1), so the last term in Eq. 5.6 can be analytically calculated as usual [83]. Finally, the middle term requires that our filter be differentiable so we can compute the derivative of the filtered color with respect to the filter parameters $\partial h_j(\bar{\mathbf{s}}_{\mathcal{N}(p)}, \boldsymbol{\theta}_p)/\partial \theta_{m,p}$. Observing that the filter weights $d_{p,q}$ are a function of the filter parameters $\boldsymbol{\theta}_i$ and using the chain rule and Eq. 4.1, we have:

$$\frac{\partial \hat{c}_{p,j}}{\partial \theta_{m,p}} = \frac{\partial h_j(\bar{\mathbf{s}}_{\mathcal{N}(p)}, \boldsymbol{\theta}_p)}{\partial \theta_{m,p}} = \sum_{q \in \mathcal{N}(p)} \frac{\partial h_j(\bar{\mathbf{s}}_{\mathcal{N}(p)}, \boldsymbol{\theta}_p)}{\partial d_{p,q}} \frac{\partial d_{p,q}}{\partial \theta_{m,p}}. \tag{5.8}$$

Using the quotient rule, we can write the first term as:

$$\frac{\partial h_j(\bar{\mathbf{s}}_{\mathcal{N}(p)}, \boldsymbol{\theta}_p)}{\partial d_{p,q}} = \frac{\bar{\mathbf{c}}_{q,j} \sum_{n \in \mathcal{N}(p)} d_{p,n} - \sum_{n \in \mathcal{N}(p)} d_{p,n} \bar{\mathbf{c}}_{n,j}}{\left(\sum_{n \in \mathcal{N}(p)} d_{p,n}\right)^2}. \tag{5.9}$$

The second term in Eq. 5.8 depends on the filter choice. For the cross-bilateral filter defined in Eq. 5.1, we need to take the derivative with respect to the position (α_p), color (β_p), and additional features ($\gamma_{k,p}$) filter parameters:

$$\frac{\partial d_{p,q}}{\partial \alpha_p} = d_{p,q}\frac{\|\bar{\mathbf{p}}_p - \bar{\mathbf{p}}_q\|^2}{\alpha_p^3},$$
$$\frac{\partial d_{p,q}}{\partial \beta_p} = d_{p,q}\frac{D(\bar{\mathbf{c}}_p, \bar{\mathbf{c}}_q)}{\beta_p^3},$$
$$\frac{\partial d_{p,q}}{\partial \gamma_{k,p}} = d_{p,q}\frac{D_k(\bar{\mathbf{f}}_{p,k}, \bar{\mathbf{f}}_{q,k})}{\gamma_{k,p}^3}. \tag{5.10}$$

Other differentiable filters can be applied similarly in our system. For example, we can use the cross non-local means filter [56, 82]:

$$d_{p,q} = \exp(-\frac{D(\bar{\mathbf{Q}}_p, \bar{\mathbf{Q}}_q)}{2\rho_p^2}) \times \prod_{k=1}^{K} \exp(-\frac{D_k(\bar{\mathbf{f}}_{p,k}, \bar{\mathbf{f}}_{q,k})}{2\gamma_{k,p}^2}), \tag{5.11}$$

where $\bar{\mathbf{Q}}_p$ is a small patch (5×5 in our implementation) of mean sample colors around pixel p and ρ_p is the standard deviation at pixel p for the non-local means term. Furthermore, D is defined as:

$$D(\bar{\mathbf{Q}}_p, \bar{\mathbf{Q}}_q) = \frac{1}{|\bar{\mathbf{Q}}|}\sum_{k\in\mathcal{K}} \frac{\|\bar{\mathbf{c}}_{p+k} - \bar{\mathbf{c}}_{q+k}\|^2}{\psi_{p+k}^2 + \psi_{q+k}^2 + \zeta} \tag{5.12}$$

where $|\bar{\mathbf{Q}}|$ is the number of pixels in the patch (25 for a 5×5 patch) and $\mathcal{K}$ is a set of offsets from the center pixel within a patch. Moreover, ψ_{p+k} and ψ_{q+k} are the standard deviations of color samples at pixel $p + k$ and $q + k$, respectively, and ζ is a small number (10^{-10}) to avoid division by zero. The derivative of this

filter with respect to the filter parameters can be calculated similarly to the cross-bilateral filter. The only difference is that, instead of the position and color terms in Eq. 5.10, we have:

$$\frac{\partial d_{p,q}}{\partial \rho_p} = d_{p,q} \frac{D(\bar{\mathbf{Q}}_p, \bar{\mathbf{Q}}_q)}{\rho_p^3} \tag{5.13}$$

Note that the cubic terms in the denominator of Eqs. 5.10 and 5.13 increase computational complexity. Therefore, in our implementation, we define auxiliary filter parameters $\lambda_{m,p} = 1/(2\theta_{m,p}^2)$ to use instead of the original filter parameters. The filter derivatives with respect to these auxiliary parameters do not have the cubic term in the denominator and are, therefore, more computationally efficient. To reflect this change in our system, the network outputs these auxiliary parameters and the filters defined in Eqs. 5.1 and 5.11 are modified appropriately.

To summarize the training process, we compute the derivative in Eq. 5.6 for each weight in the network, and update the weights after every epoch. This iterative process continues until convergence.

5.2.2 Primary features

Primary features are those directly output by the rendering system. We use 7 primary features ($M = 7$) in our cross-bilateral filter (Eq. 5.1), consisting of

screen position, color, and five additional features ($K = 5$): **(1)** world position, **(2)** shading normal, **(3,4)** texture values for the first and second intersections, and **(5)** direct illumination visibility. This last feature was also used by Rousselle et al. [82] and we found it to be useful in our system.

During rendering, for each sample we output screen position in x and y, color in RGB format, world position in Cartesian coordinates (x, y, and z), shading normal (i, j, and k), texture values for the first and second intersections in RGB format, and a single binary value for the direct illumination visibility, for a total of 18 floating point values. We then average these values for all samples in a pixel to produce the mean primary features for every pixel in the image. Note that at this point, the average direct illumination visibility represents the fraction of shadow rays that see the light and is not a binary value. Moreover, as was done in Rousselle et al. [82], we prefilter the additional features using a non-local means filter in an 11×11 window with patch size 7×7.

Similar to recent approaches [56, 82], we normalize the distance of the color and additional features by their variances. We use the following function for the color term:

$$D(\bar{\mathbf{c}}_p, \bar{\mathbf{c}}_q) = \frac{\|\bar{\mathbf{c}}_p - \bar{\mathbf{c}}_q\|^2}{\psi_p^2 + \psi_q^2 + \zeta}, \tag{5.14}$$

where ψ_p and ψ_q are the standard deviation of color samples at pixel p and q, respectively, and ζ is a small number (10^{-10}) to avoid division by zero. For the additional features, we use:

$$D_k(\bar{\mathbf{f}}_{p,k}, \bar{\mathbf{f}}_{q,k}) = \frac{\|\bar{\mathbf{f}}_{p,k} - \bar{\mathbf{f}}_{q,k}\|^2}{\max(\psi_{k,p}^2, \delta)}, \tag{5.15}$$

where $\psi_{k,p}$ is the standard deviation of the k^{th} feature at pixel p and δ is a small number (10^{-4}) to avoid division by zero. These parameters are all similar to those of Rousselle et al. [82]. On a final note, we smooth the noisy standard deviations for the additional features, $\psi_{k,p}$, by filtering them using the same weights computed by the non-local means filter when filtering the primary features.

5.2.3 Secondary features

At every pixel, we compute a set of secondary features from the noisy samples in a local region to serve as inputs to our neural network. Most features are inspired by (or are direct implementations of) those used in previous algorithms for MC denoising.

Feature statistics: We compute the mean and standard deviation for the $K = 5$ additional features of all samples in the pixel. To capture more global statistics, we also calculate the mean and standard deviation of the pixel-averaged features in a 7×7 block around each pixel. We compute the statistics for each

component (e.g., i, j, k for shading normal) separately and average them together to get a single value per feature. Thus, we have 20 total values for each pixel and the block around it (5 for the means and 5 for standard deviations of each).

Gradients: The gradients of features have been previously used to decrease the weight of a feature in regions with sharp edges [82]. We calculate the gradient magnitude (scalar) of the K additional features using a Sobel operator (5 values).

Mean deviation: This term is the average of the absolute difference between each individual pixel in a block and the block mean. A normalized version of this metric was proposed by Hachisuka et al. [36] as an error metric for adaptive sampling. This feature can help identify regions with large error so the network can adjust the filter parameters accordingly. For each of the K additional features, we compute the mean deviation of all the pixel-averaged features in a 3×3 block around each pixel. As with feature statistics, this is computed on each component separately and then averaged to obtain a single value for each additional feature (5 values total).

MAD: We use the median absolute deviation (MAD) proposed by Kalantari and Sen [47] (discussed in Chapter 3) to estimate the amount of noise in each pixel, which is directly related to the size of the filter. We compute MAD for each K additional features (5 values total).

Sampling rate: Finally, we use the inverse of the sampling rate $1/n$ as a secondary feature. The variance of MC noise decreases linearly with the number of samples and, therefore, the filter parameters should reflect this. Since we only train a single network, it should be able to handle different sampling rates and adjust the filter size accordingly.

In summary, we compute a total of $N = 36$ secondary features for each pixel. The network takes these features as input and outputs the parameters to be used by the filter to generate the final filtered pixel. This is done for all the pixels to produce the final result.

5.3 Implementation details

5.3.1 Neural network structure

The complexity of a network is determined by the number of layers and the node count of each. Although a more complex network containing several layers with many nodes can model more complicated relationships between its inputs and outputs, training such networks is difficult and there is an increased likelihood of overfitting to the training data. In these cases, the network could model the noise rather than the underlying relationships and, therefore, would perform poorly on actual test data. We empirically found that one hidden layer of size 10 provides

the best performance in general (see Sec. 5.5). The input layer consists of 36 nodes corresponding to our secondary features, while our output layer has one node for each filter parameter (6 total). Note that we use a fixed color weight, β_i, and, thus, our output layer only requires 6 nodes (see Sec. 5.3.2).

For the hidden layer, we chose the sigmoid activation function $f(x) = 1/[1 + \exp(-x)]$ and for the output layer, we used the softplus function $f(x) = \log(1+e^x)$, a smooth differentiable approximation to the rectifier function. The output of the softplus function is always positive $(0, \infty)$, making it the ideal candidate to output the filter parameters in the final layer of the MLP network.

5.3.2 Training

As explained in Section 5.2, training is done using the iterative backpropagation algorithm [83]. At every iteration, all of the weights in the network are updated based on the error gradient with respect to the weights. We used the resilient backpropagation (RPROP) method [79] to update the weights, rather than the commonly used gradient descent. In RPROP, the weights are updated based on the signs of the gradients, rather than their magnitudes. At every iteration of the backpropagation process, the gradients of a particular weight are rewarded if the current and previous iterations are consistent (have the same sign) and penalized otherwise. We found this method results in faster convergence for our system

compared to gradient descent. To further speed up convergence, we normalize the network features by subtracting the mean and dividing by the standard deviation of the training set, as is frequently done in neural networks [52]. Finally, we observed that, in practice, optimizing the filter parameter for the color term of the cross-bilateral filter results in overfitting. We experimentally found that setting this parameter to a fixed value, $\beta_i = 7$, is sufficient to keep the necessary scene details while removing noise.

In our training set, we used 20 scenes with different distributed effects such as depth of field, motion blur, and global illumination (see Fig. 5.5). For each scene, we rendered the ground truth image with a large number of samples (e.g., 32K). To handle scenes with different sampling rates, we rendered each training scene with 4, 8, 16, 32, and 64 samples per pixel and trained all of the images on a single network. Moreover, to avoid overfitting to one particular noise pattern, we rendered each scene five times at each sampling rate (a total of 25 images for each scene). To train the network on all of these images efficiently, we performed mini-batch training [37], where each iteration of the backpropagation process is performed on a subset of the training set. Specifically, we compute the derivative from Eq. 5.6 for each pixel across all of the images in the subset. We then average all of the gradients and use the result to update the weights. In our implementation, our subset consisted of one image at each sampling rate from all of the

training scenes. Furthermore, we alternated between the five noisy examples at each iteration.

5.3.3 Handling high dynamic range spikes

MC rendered images have high dynamic range and could contain spikes, pixels with radiance values orders of magnitude greater than their neighbors. Usually these high energy pixels cannot be spread efficiently, even with a big filter, and, thus, need to be handled differently. After applying the filter, we identify these spikes in the filtered result by first calculating the mean and standard deviation of the RGB colors for all the neighboring pixels in a block of size 3×3. If any color channel of the center pixel has values more than 2 standard deviations away from the block average, then it is labeled as a spike and replaced with the median block color. Fig. 5.3 shows how spike removal affects KITCHEN, a test scene. As can be seen, our system does not heavily rely on spike removal and can spread the energy of most of the high magnitude pixels itself. Although this spike removal process is not differentiable, we found that it does not aversely affect the training. A better way of handling spikes within our framework is left for future investigation.



Figure 5.3: The image on the left shows the result of our approach before spike removal on an inset of the KITCHEN scene. In our method, we remove high magnitude spikes in the filtered image as a post-process to produce the result shown in the middle. The ground truth image is shown on the right for comparison.

5.4 Results

We implemented our algorithm, called learning-based filtering (LBF), in C++ and integrated it into PBRT2 [73]. The neural network and filter were written in CUDA for GPU acceleration. All timings were obtained on an Intel quad-core 3.7 GHz machine with 24GB of memory and a GeForce GTX TITAN GPU. We used uniform low-discrepancy samples to generate the noisy images for both the training and test sets. Note that we trained a single network on scenes with 4, 8, 16, 32, and 64 samples per pixel to produce all the results in this work. Moreover, we used a cross-bilateral filter (of size 55×55) unless otherwise noted.

We begin by evaluating the robustness of our learning system to the training set. To do this, we randomly selected 8 test scenes from a pool of 42 scenes spanning a variety of distributed effects. We then trained four different networks on random training sets composed of 16 randomly selected scenes from the remaining

34. To compare the convergence of these four networks, we evaluated the progress of each network during training by computing the average error (see Eq. 5.5) on the 8 test scenes at every epoch. As seen in Fig. 5.4, all four networks converge to approximately the same error, showing that our learning system is fairly robust to the choice of training set.

For the main comparisons, we trained our network on 20 scenes with a variety of Monte Carlo effects (see Fig. 5.5). As the robustness test in Fig. 5.4 shows, the network typically converges after approximately 25 epochs. However, to ensure that the network was fully converged, we performed 50 epochs which took approximately 9 hours, and then used these converged weights to generate the results for comparisons. Note that all results shown here (with the exception of Fig. 5.5) are produced on test scenes that were not part of our training set.

We compare our method against state-of-the-art approaches including non-local means filtering (NLM) [81], random parameter filtering (RPF) [90], SURE-based filtering (SBF) [56], robust denoising using feature and color (RDFC) [82], and weighted local regression (WLR) [66]. Note that with the exception of RPF, all the other algorithms are significantly benefiting from adaptive sampling in these comparisons, while our approach is not adaptive. We also compare against adaptive sampling and reconstruction (ASR) [80] and ray histogram fusion (RHF) [19] in the supplementary materials.

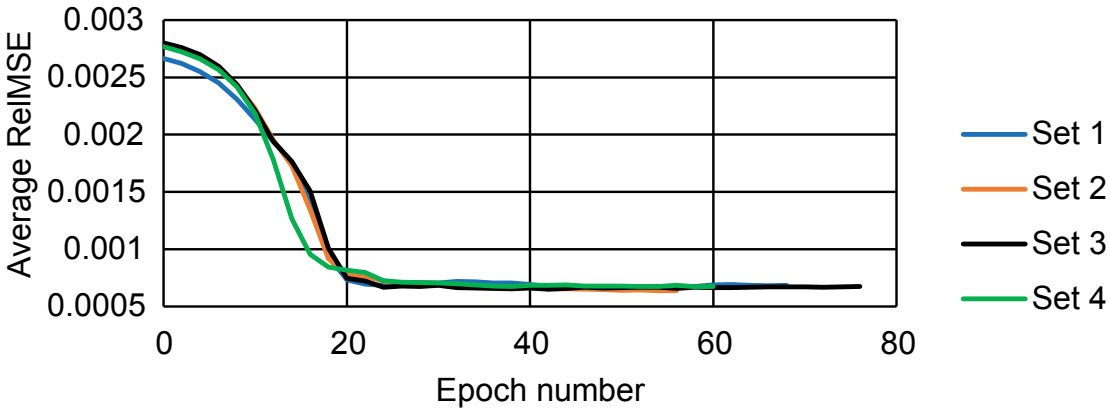


Figure 5.4: We evaluate the robustness of our learning system to the choice of training set by training four different networks on 16 randomly selected scenes drawn from a pool of 34. We plot the average relative MSE of each network for a set of 8 test scenes at each epoch of the training. As can be seen, all four networks converge to roughly the same test error which shows the robustness of our approach to the training set. The images used in each set as well as the test images are available in the supplementary materials.

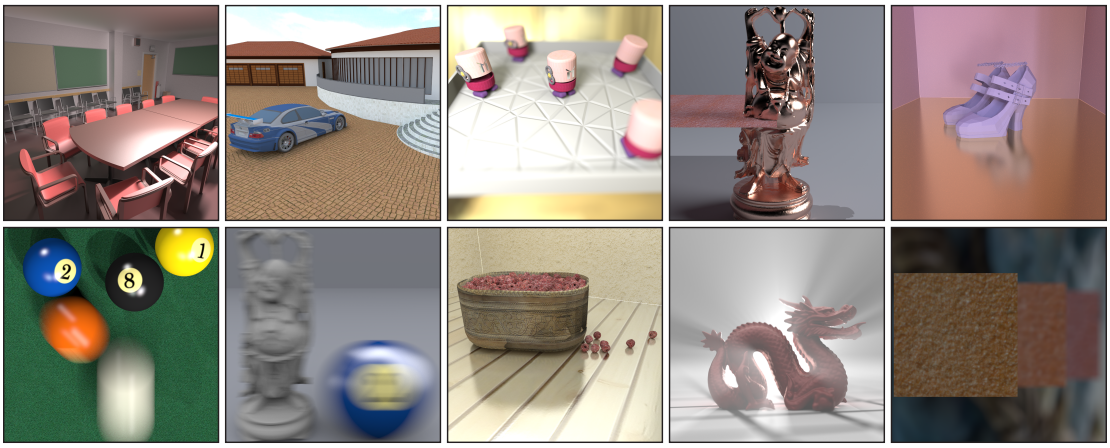


Figure 5.5: A subset of our training scenes (cropped for best fit). These images have different distributed effects including depth of field, motion blur, glossy reflections, and global illumination. All 20 training images are available in the supplementary materials.

For all algorithms, we used the implementations provided by the authors with their default parameters. For RDFC, we used a window size of 20 for higher qual-

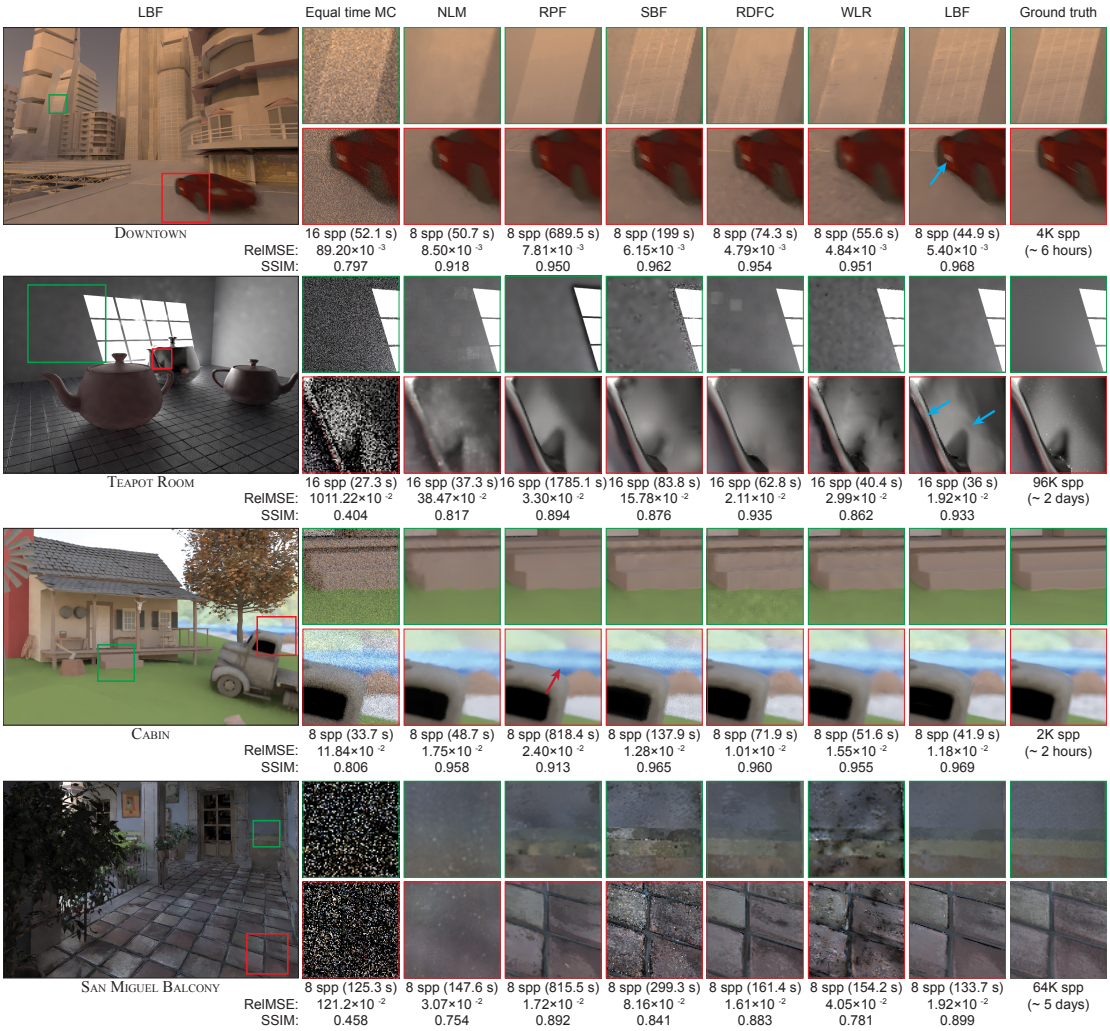


Figure 5.6: We compare our algorithm (LBF) against several state-of-the-art techniques as well as equal-time Monte Carlo. Specifically, we compare our approach against non-local means filtering (NLM) [81], random parameter filtering (RPF) [90], SURE-based filtering (SBF) [56], robust denoising using feature and color (RDFC) [82], and weighted local regression (WLR) [66]. Our algorithm is faster than all these filtering approaches, but we perform equal-sample comparisons to guarantee fairness. Note that since the low-discrepancy sampling pattern in PBRT2 requires a power-of-two sampling rate, we could not match timings exactly for equal-time MC. The relative MSE and SSIM index values are listed below each image (for SSIM higher is better). Tonemapping of the insets have been adjusted equally for all algorithms for best visibility. Full images are available in the supplementary materials.

ity, as suggested by the authors [82]. For quantitative comparisons, we calculated relative MSE as proposed by Rousselle et al. [80]: RelMSE $= (\hat{c} - c)^2/(c^2 + \varepsilon)$ averaged over all pixels, where $\hat{c}$ and c are the filtered and ground truth pixel colors, respectively, and ε is a small number (0.01) to avoid division by zero. We also measure the perceptual quality of the results using the structural similarity (SSIM) index [105] which is a value from 0 to 1, where 1 indicates perfect quality with respect to the ground truth image.

Fig. 5.6 compares our approach to state-of-the-art methods on four scenes with different distributed effects. Although our algorithm is faster than all other methods shown here, we perform equal-sample comparisons to ensure fairness (please refer to the supplementary materials for additional equal-time comparisons for some scenes). For completeness, we also show the approximate results of equal-time Monte Carlo, since the low-discrepancy sampling in PBRT2 requires a power-of-two sampling rate.

First, we examine the path-traced DOWNTOWN scene with global illumination and motion blur. NLM uses only color information when filtering, so it cannot preserve the geometry and texture details in the scene. Moreover, although the other approaches use additional features, they often do not have appropriate filter weights, resulting in either overblurred textures or residual motion blur noise. Our approach is not only the fastest but it also preserves the textures while removing

the motion blur noise, resulting in an image with fewer artifacts than the other techniques. Note, for example, that only our approach is able to preserve the thin lines between the stripes on the car (indicated by the blue arrow).

The Teapot Room scene is a challenging, path-traced scene containing one glossy and two diffuse teapots with mostly indirect illumination. None of the other methods can effectively remove the strong indirect illumination noise on the back wall without introducing artifacts or overblurring the glossy reflections on the body and spout of the teapot. Note the ground truth image still has visible spikes at 96K spp, while we produce a relatively noise-free result.

The Cabin scene is a path-traced scene that includes global illumination and depth of field. NLM and RPF remove the depth of field noise, but overblur the geometry of the steps and texture in the water. SBF and WLR do not weight the features appropriately and thus their results contain residual noise in the depth of field regions. Although both our approach and RDFC handle the depth of field regions well, RDFC produces visible artifacts in the smooth regions due to the low sampling rate. Meanwhile, we are able to generate a smooth result that is closer to ground truth.

Finally, San Miguel Balcony is a path-traced scene with severe noise at 8 spp. Again, NLM overblurs the textures on the floor and the wall. Moreover, SBF and WLR produce results that are overblurred or contain residual noise.

Although RPF, RDFC, and our method preserve the floor texture, RDFC and RPF cannot properly remove the noise in the smooth regions, including the back wall. Despite having a slightly higher relative MSE than RPF and RDFC, we produce a relatively noise-free result that is better than the other algorithms both visually and in terms of SSIM.

To test the convergence of our method as the sampling rate increases, we provide convergence plots in Fig. 5.7 using relative MSE and SSIM metrics for the DOWNTOWN scene and provide comparisons against low-discrepancy samples, NLM, SBF, RDFC, and WLR. Since RPF is computationally expensive, we omit its comparison. As can be seen, our relative MSE consistently decreases even at the sampling rates that the network has not been trained on, demonstrating the network's ability to extrapolate. Although higher sampling rates could be included in the training set, we found our set to be sufficient for the purposes of this work. Moreover, the perceptual quality of our method is higher than others based on SSIM index, particularly at lower sampling rates. Our method is a consistent estimator because the distances in the color term are normalized by the variances. Thus, as the sampling rate increases, these variances are reduced and the filter approaches identity.

We also demonstrate how other filters can be used in our framework by training with the cross non-local means filter [56, 82]. Since every filter has a different

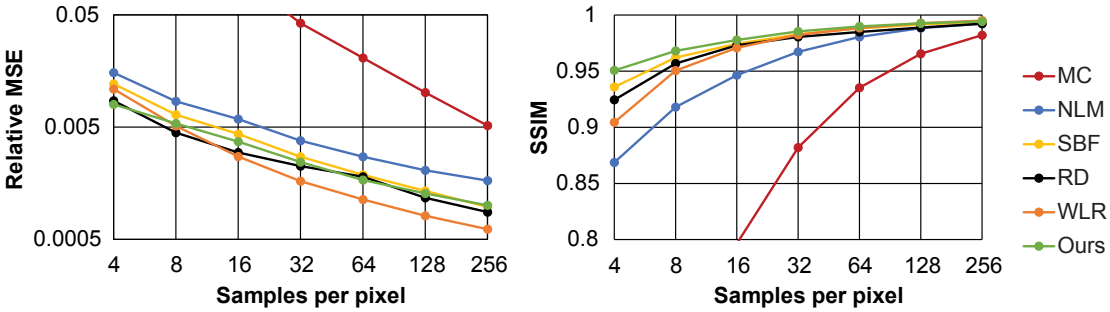


Figure 5.7: On the left, we show the convergence plot in terms of relative MSE for the Downtown scene with our approach and several state-of-the-art approaches. Our relative MSE consistently decreases despite the fact that the network has not been trained on sampling rates higher than 64 samples per pixel. We show the convergence plot in terms of SSIM index on the right. Our method produces results with higher perceptual quality, particularly at low sampling rates. Note that although WLR has strong relative MSE, the perceptual quality of its results is lower than SBF, RDFC, and our approach because of visible artifacts. Also, our algorithm is faster than the others, so equal number of samples is not equal time.

relationship between the secondary features and its filter parameters, we cannot simply use the network weights from the previous cross bilateral filter. In Fig. 5.8, we compare our method against NLM, SBF, and RDFC (all with non-local means filters) and show that our method produces a smoother result and preserves most of the scene detail.

Finally, to handle animated sequences, we simply use the same network from before *without retraining* and extend our cross-bilateral filter to operate on 3-D spatio-temporal volumes. This modification to the filter is necessary to reduce the flickering that might appear if we filter each frame independently. We tested our extension on the Sci-fi City, Kitchen, Sibenik, and San Miguel video

sequences, which can be found in the accompanying video. In all cases, we used only three neighboring frames on each side of the current frame (7 frames total) for spatio-temporal filtering. Our approach is able to generate high-quality, temporally-coherent videos from noisy input sequences with low sampling rates.

In terms of timing, the overhead of our algorithm is roughly independent of the sampling rate. In general, our algorithm with a cross-bilateral filter removes noise from a 1200×800 image in roughly 8.4 seconds, where calculating the features takes approximately 5 seconds and evaluating the network and filtering takes 3.4 seconds. For video sequences, the spatio-temporal filter increases our timings to roughly one minute for each 1200×800 frame, which is reasonable given the quality of our videos.

5.5 Discussion

In our method, we estimate $\mathcal{G}$ by training a network through direct minimization of the error between the filtered and ground truth images (Eq. 5.3). An alternative approach might be to approximate $\mathcal{G}$ by minimizing the error between the estimated and ground truth filter parameters, i.e., $E(\mathcal{G}(\mathbf{x}_i), \boldsymbol{\theta}_i^*)$. However, this approach has two major problems. First, it provides a suboptimal solution to $\mathcal{G}$ since it does not minimize the error between the filtered and ground truth

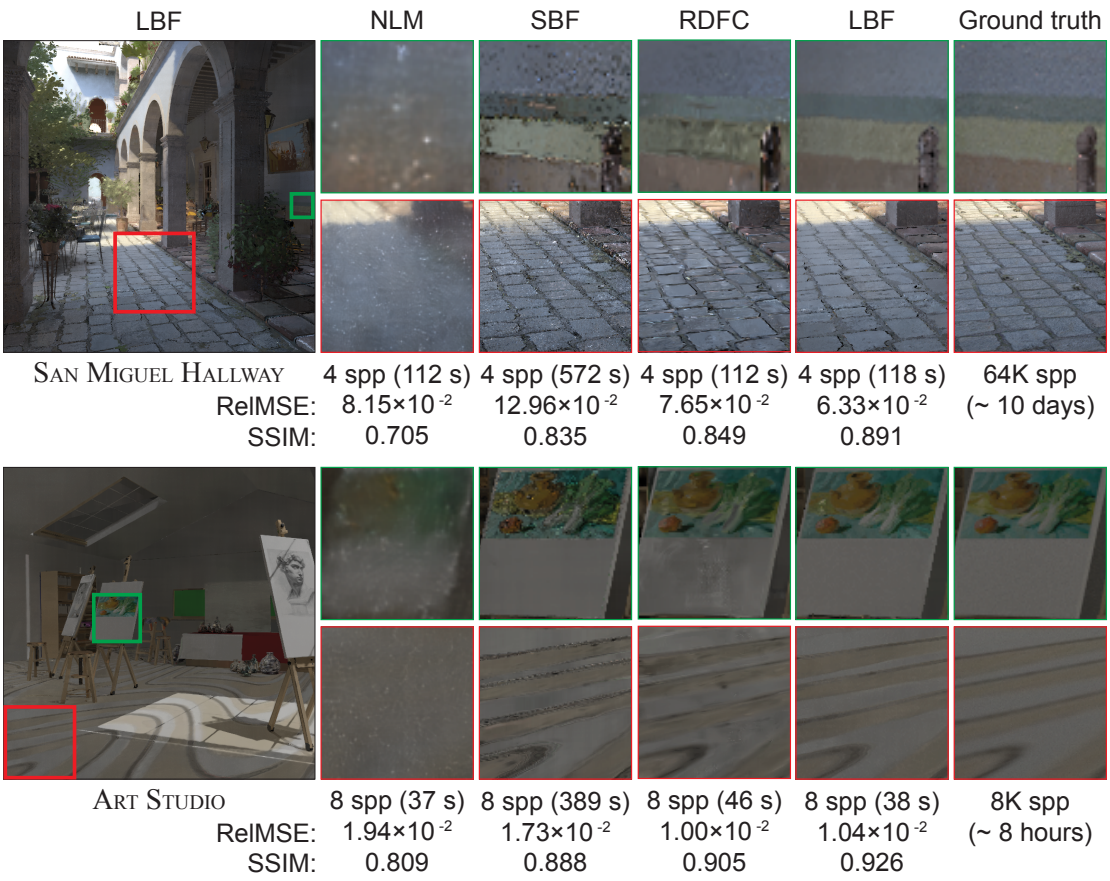


Figure 5.8: Comparison between our approach (LBF) and three other methods all using non-local means filters. In both scenes, non-local means filtering (NLM) overblurs structure and textures while SURE-based filtering (SBF) and robust denoising using feature and color (RDFC) have noticeable artifacts in both smooth and textured regions.

images, which is our final goal. In some cases, a small error in filter parameters might result in a large error in the final image. Second, it requires calculating the optimum filter parameters $\boldsymbol{\theta}_i^*$ for each pixel in every image of the training set. This requires an expensive, brute-force search in the parameter space to perform the error minimization in Eq. 4.4.

Note that the concept of training a machine learning model (e.g., a neural network) in combination with a function (e.g., a filter) has been introduced in the image processing community and was used for applications like image denoising [96, 85, 41] and deblurring [86]. Tappen et al. [96] first introduced this idea to optimize the parameters of a Gaussian conditional random field in an image denoising system. Jancsary et al. [41] and Schmidt et al. [86] extended this idea by training local regression tree fields for image denoising and deblurring applications. Here, we used this idea to train a neural network in combination with a matching filter for removing MC noise.

As discussed in Sec. 5.3.1, network size affects performance and thus it is important to size it appropriately. We evaluate this effect by training five networks of different sizes on the first training set of the robustness experiment from Fig. 5.4. Fig. 5.9 shows convergence plots for the different network sizes based on the average relative MSE of 8 test scenes (also from the robustness experiment) evaluated after each epoch. Since the network with one hidden layer of size 10 produces slightly better results than the other configurations, we use it for all the results in this work.

The foundation of our approach lies in having an effective training stage that can generate high-quality results on the test scenes. Therefore, the training set should include scenes with a variety of distributed effects so that the network

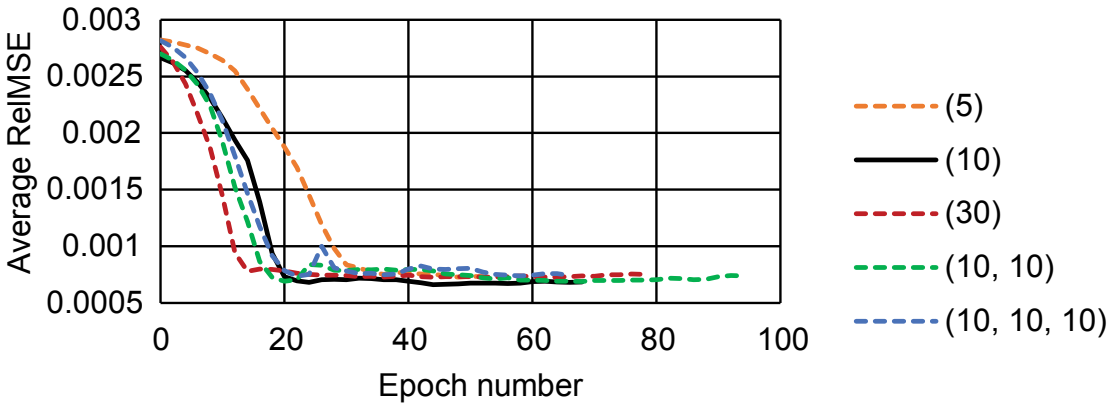


Figure 5.9: We evaluate the effect of network size on learning performance. Here, we train 5 networks of different sizes and compute the average relative MSE on 8 test images at every epoch. The sizes differ in the number of hidden layers and the number of nodes in each layer. For example, (10, 10) is a network with 2 hidden layers and 10 nodes in each layer. We found that the network with one hidden layer of size 10 provides slightly better performance than the other sizes. The sizes of the input and output layers, which correspond to the number of secondary features and the number of output filter parameters, respectively, are both fixed.

can learn how to robustly handle diverse test scenes. To evaluate the effect of a particular distributed effect, we remove all scenes with glossy reflections from the training set used in Fig. 5.6 and replace them with non-glossy scenes containing other distributed effects. We show the result of the new trained network on the Teapot Room test scene in Fig. 5.10. Although the glossy reflections are preserved better than most other approaches (see Fig. 5.6), the result has lower quality compared to the network trained with scenes containing glossy effects.

Our network uses various secondary features to estimate filter parameters and perform filtering. We evaluated the impact of the secondary features in each of

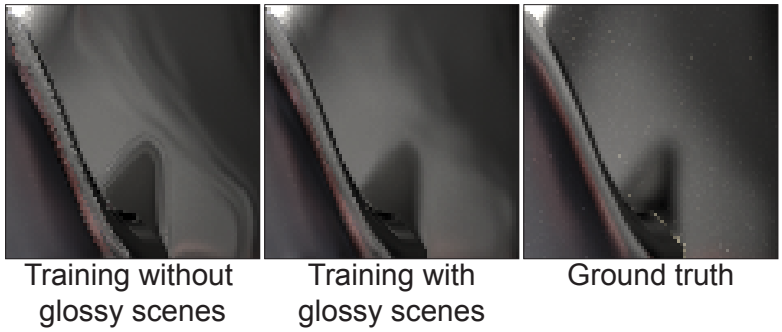


Figure 5.10: We show the effect of training on scenes without glossy reflections on the TEAPOT ROOM test scene. Even without glossy scenes in the training set, our method is able to capture the glossy reflections on the teapot. However, the result has fewer artifacts when the glossy effect is present in the training set.

the 5 categories defined in Section 5.2.3 by excluding all the features from each category one at a time and training the network. Based on this experiment, we observed that feature statistics have the most impact on the quality of the result while removing the other features only slightly reduced the quality. However, since these other features are fast to compute, we used them all in our framework for improved quality.

Furthermore, to show that the network cannot predict optimal filter parameters from the color information, we trained a network by computing all the secondary features, except the sampling rate, only on the color image. As seen in Fig. 5.11, the additional features such as world positions and shading normals are required to predict optimal filter parameters.

Like many other MC filtering approaches, the main limitation of our algorithm is that it produces biased but consistent results. As such, it may not be an ade-

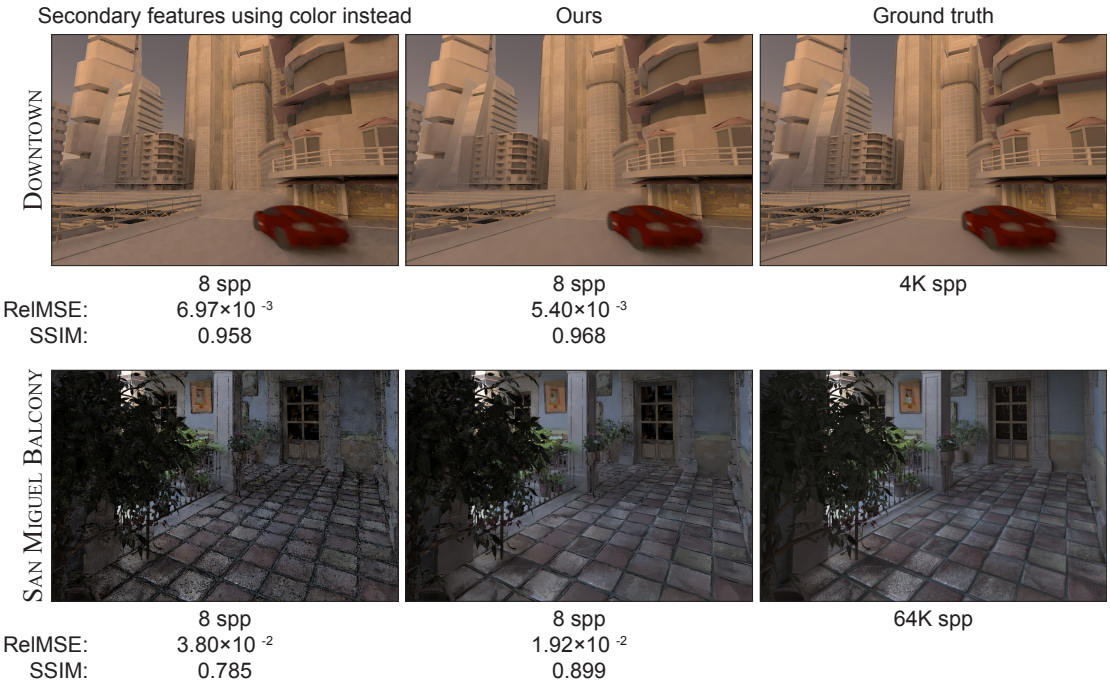


Figure 5.11: We computed all the secondary features, except the sampling rate, only on color image and trained the network. As seen, the trained network in this case cannot predict the optimal filter parameters and produces results with severe artifacts.

quate solution for applications requiring complete physical accuracy. However, we see strong potential in fields that focus more on perceptual quality such as digital film production, particularly for previsualization when quick rendering times are important and very low sampling rates are used. However, as shown in Fig. 5.12, our approach produces reasonable results at high sampling rates as well, which are comparable in quality to other methods.

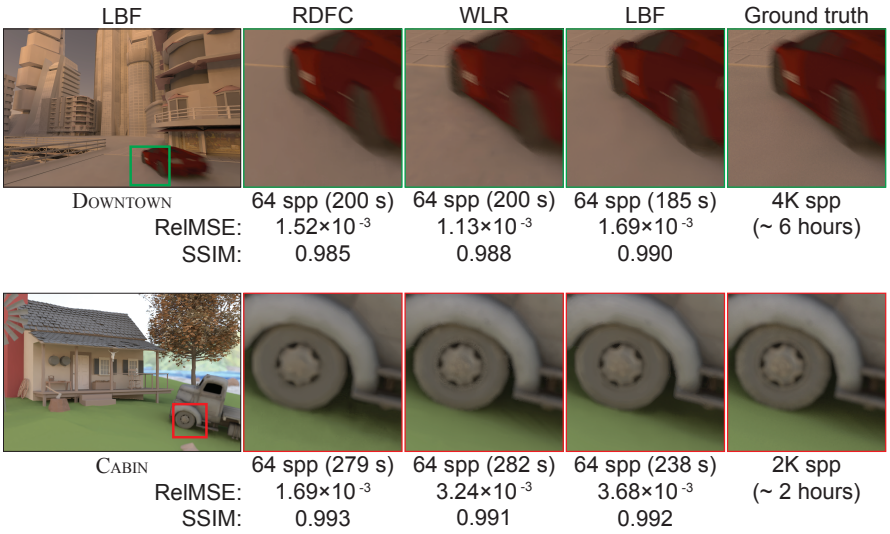


Figure 5.12: Comparison between our approach (LBF), robust denoising using feature and color (RDFC) [82], and weighted local regression (WLR) [66] using a cross-bilateral filter at 64 samples per pixel. Our method produces results with comparable quality to other algorithms at this high sampling rate. Note that these results are produced using the network trained with 4, 8, 16, 32, and 64 samples per pixel, as in Fig. 5.6.

5.6 Conclusion

We have presented a machine learning approach to estimate filter parameters in a direct estimation framework. In order to model the complex relationship between the ideal filter parameters and a set of features extracted from the input noisy samples, we use a multilayer perceptron (MLP) neural network as a non-linear regression model. To effectively train the network, we combine the MLP network with a filter such that the standard MLP takes in a set of secondary features extracted from a local neighborhood at each pixel and outputs a set of filter parameters. The filter takes these parameters and the noisy samples as in-

puts and generates a filtered pixel that is compared to the ground truth pixel during training. We train our proposed system on a set of scenes with a variety of distributed effects and then test it on different scenes containing motion blur, depth of field, area lighting, glossy reflections, and global illumination. Our results show that this simple approach demonstrates significant improvement over existing state-of-the-art methods.

Chapter 6

Training for optimal filter parameter sets

As discussed in Chapter 4, error-minimization techniques filter the noise by attempting to minimize the error between the filtered and the ground truth images [56, 66, 80, 82]. Since the ground truth image is not available at run-time, these error-minimization algorithms approximate this error using no-reference error estimation metrics. To do this, they consider a set of predefined *candidate* filter parameters and at every pixel they effectively select the filter that minimizes this error estimate. However, these approaches all define the candidate filter parameters heuristically, which is sub-optimal and often leads to artifacts in the final result.

To avoid this problem, in this chapter we propose to optimize these candidate filter parameters in order to produce images that are close to the ground truth. Specifically, we optimize the parameters of a set of N candidate filter parame-

ters by minimizing an objective function that matches the filtered result to the ground truth over a set of training scenes. Moreover, we improve upon the filters commonly used by adding an extra parameter to the cross non-local means filter, allowing it to smoothly transition to a cross-bilateral filter and thereby leverage the power of both filters. This additional parameter is also optimized to allow our system to utilize the power of both filters if necessary. Once our system has been optimized, we can use these optimal candidate filter parameters in our error minimization framework at run-time to denoise new test scenes.

6.1 Error-minimization with optimal filter sets

As discussed in Chapter 4, the goal of *all* post-process filtering algorithms for MC noise is to find filter parameters that minimize the error between the filtered and ground truth images (see Eq. 4.4). Since the ground truth image is not available at run-time, the error-minimization filtering methods require a no-reference error estimation metric such as Stein's unbiased risk estimator (SURE) [94] to approximate the actual error directly from the filtered pixel color.

Therefore, the goal of these error-minimization frameworks is to perform the energy minimization in Eq. 4.6. At this point, one could try to find continuous filter parameters to satisfy Eq. 4.6, but this is usually difficult and computationally

expensive. Thus, the typical approach is to discretize the parameter space and define a set of N candidate filter parameters $\Theta = \{\boldsymbol{\theta}_1, \cdots, \boldsymbol{\theta}_N\}$. This is usually done *a priori* during the design of the algorithm, using logical (but heuristic) rules.

Once the set of N candidate filter parameters are known, we can use Eq. 4.6 to find the optimal one at each pixel by selecting the filter parameters that produce a filtered pixel with the lowest estimated error. However, since no-reference error estimation metrics are typically noisy (especially at low sampling rates), choosing the filter with the lowest error as done in Eq. 4.6 is not always optimal. Therefore, to make the filter more robust, we need a more sophisticated approach that reconstructs the final image using the N filtered images and their corresponding error. This can be formalized as follows,

$$\hat{c}_p = R\left[h(\bar{\mathbf{s}}_{\mathcal{N}(p)}, \Theta), \hat{E}\left(h(\bar{\mathbf{s}}_{\mathcal{N}(p)}, \Theta)\right)\right], \tag{6.1}$$

where R is the reconstruction function, h is the filter, and $\hat{E}$ is the no-reference error estimation metric. Moreover, $\bar{\mathbf{s}}_{\mathcal{N}(p)}$ is the collection of mean primary features in the square neighborhood $\mathcal{N}(p)$ of pixel p. Note that in Eq. 6.1 we are effectively applying all N filter parameters to obtain a set of N filtered pixels, $h(\bar{\mathbf{s}}_{\mathcal{N}(p)}, \Theta)$, and computing the estimated error for each. Furthermore, R is a function that, rather than selecting the one with the minimum estimated error, might, for example, blend them together with weights instead [82]. Of course, if we had an accurate

error estimation metric, the optimal R would simply output the result of the filter with the lowest error.

Our key observation is that the choice of candidate filter parameters significantly affects the results. While previous approaches implement Eq. 6.1 using heuristically-defined candidate filter parameters, Θ, that perform sub-optimally, we propose to optimize these candidate filter parameters in an offline process and use the optimal ones at run-time to achieve a significant improvement over previous techniques. In the following sections, we explain our proposed optimization system to find a set of optimal candidate filter parameters Θ (Sec. 6.1.1), our filter h (Sec. 6.1.2), the error estimation metric $\hat{E}$ (Sec. 6.1.3), and the reconstruction function R (Sec. 6.1.4).

6.1.1 Finding optimal candidate filter parameters

For a set of N candidate filter parameters, our goal is to find the M parameters of each filter parameter set, $\boldsymbol{\theta}_i$, where $i = \{1, \cdots, N\}$, that minimizes the error between the final reconstructed and ground truth images. We propose to do this as a pre-process optimization that minimizes the following objective function,

$$\Theta^* = \underset{\Theta=\{\boldsymbol{\theta}_1,\cdots,\boldsymbol{\theta}_N\}}{\arg\min} E\left(R\left[h(\bar{\mathbf{s}}_{\mathcal{N}(p)}, \Theta), \hat{E}\left(h(\bar{\mathbf{s}}_{\mathcal{N}(p)}, \Theta)\right)\right], c_p\right), \tag{6.2}$$

The advantage of posing the problem in this way is that we can perform an offline optimization on a set of training scenes to obtain N optimum parameter sets $\boldsymbol{\theta}_1, \cdots, \boldsymbol{\theta}_N$. This energy function can be minimized iteratively, where at every iteration each parameter is updated according to the derivative of the error with respect to that parameter, i.e., $\partial E/\partial\theta_{ij}$ where θ_{ij} refers to the j^{th} parameter of the i^{th} filter, where $i = \{1, \cdots, N\}$ and $j = \{1, \cdots, M\}$.

Since our reconstruction function R and filter, discussed in Secs. 6.1.2 and 6.1.4 respectively, contain functions with discontinuities (e.g., max function), the error is non-differentiable. However, we found that numerical differentiation works well in practice. To numerically evaluate $\partial E/\partial\theta_{ij}$, we add a small number, ε, to the j^{th} parameter of the i^{th} filter (i.e., $\theta_{ij} + \varepsilon$) to generate the filter parameters set $\widetilde{\Theta}$. We then calculate the derivative by evaluating the effect of adding ε to each parameter on the error in Eq. 6.2 as follows,

$$\frac{\partial E}{\partial \theta_{ij}} \approx \frac{\widetilde{E} - E}{\varepsilon}, \tag{6.3}$$

where $\widetilde{E}$ is the error from Eq. 6.2 obtained using $\widetilde{\Theta}$ and E is the original error using Θ. This derivative is calculated for every j^{th} parameter in each of the N filters. Once the derivatives are calculated, we update the parameters at each iteration based on their corresponding derivative. We use the resilient backpropagation

(RPROP) algorithm [79] instead of more common techniques like gradient descent since it results in faster convergence.

As for the training set, we first render noisy images with low sampling rates and their corresponding ground truth images rendered with many samples. The noisy images are rendered with 4, 8, 16, 32, and 64 samples per pixel to ensure the obtained candidate filters are optimal for different sampling rates. Once this offline optimization is performed and the optimal filter parameters set Θ is obtained, we can use Eq. 6.1 to remove noise from any new test image.

6.1.2 Filter

The choice of filter is an important decision of any MC filtering approach. As discussed in Chapter 4, a filter generates a filtered pixel through a weighted average of the pixels in the neighborhood around it (see Eq. 4.1). Essentially, different filters differ in how they compute the weights between pixels, $d_{p,q}$.

The two most commonly used filters for MC denoising include the cross-bilateral and cross non-local means filters [45, 56, 82, 90]. In a non-local means filter, the weight between two pixel colors are computed based on the distance of the two patches surrounding them. A bilateral filter can be seen as a special case of this where the patch size is 1×1, so it computes the weight simply based on the distance of the two pixel colors.

We observe that each of these two filters has different advantages for handling specific regions of the noisy image. For example, a cross-bilateral filter can better handle the regions with intricate scene detail, while cross non-local means is more effective in removing noise from smooth regions. Although most previous approaches have used only one of these two filters at a time, we utilize the general cross non-local means filter introduced in Buades et al. [7] that uses a Gaussian kernel in the distance function. We expose this as an additional parameter that enables our filter to smoothly transition from a cross non-local means to a cross-bilateral filter and allow our method to leverage the power of both.

We first explain the general non-local means component used for the pixel colors, and later describe the other cross terms applied to the additional scene features. For the non-local means component, we define the weight between pixels p and q as,

$$d_{p,q}^{\mathrm{NLM}} = \exp\left[-\frac{\max(D(\bar{\mathbf{P}}, \bar{\mathbf{Q}}, \sigma_g), 0)}{2\beta^2}\right], \tag{6.4}$$

where β is the standard deviation of the filter which defines its width, and $\mathbf{P}$ and $\mathbf{Q}$ are small patches (7×7 in our implementation) centered around pixels p and q, respectively. Moreover, $D(\bar{\mathbf{P}}, \bar{\mathbf{Q}}, \sigma_g)$ computes the distance between the two patches as,

$$D(\bar{\mathbf{P}}, \bar{\mathbf{Q}}, \sigma_g) = \frac{1}{|\bar{\mathbf{P}}|} \sum_{j \in \mathcal{J}} \left[\exp\left(-\frac{\|j\|^2}{2\sigma_g^2} \right) \times \frac{\|\bar{\mathbf{P}}_j - \bar{\mathbf{Q}}_j\|^2 - \psi[\bar{\mathbf{P}}_j] - \psi[\bar{\mathbf{P}}_j, \bar{\mathbf{Q}}_j]}{\psi[\bar{\mathbf{P}}_j] + \psi[\bar{\mathbf{Q}}_j] + \varepsilon} \right],$$

where $|\bar{\mathbf{P}}|$ is the number of pixels in patch $\bar{\mathbf{P}}$, $\bar{\mathbf{P}}_j$ is the patch color at position j and $\mathcal{J}$ is the collection of offsets from the patch center. Moreover, $\psi[\bar{\mathbf{P}}_j]$ is the variance of all the sample colors in pixel $\bar{\mathbf{P}}_j$, $\psi[\bar{\mathbf{P}}_j, \bar{\mathbf{Q}}_j]$ is $\min(\psi[\bar{\mathbf{P}}_j], \psi[\bar{\mathbf{Q}}_j])$, and ε is 10^{-10} to avoid division by zero.

Here, the key difference with respect to typical non-local means filters used for MC denoising [81, 82] is in the additional exponential weight, controlled by the parameter σ_g. As σ_g decreases, the effective size of the patch becomes smaller and, in the limit, converges to a simple bilateral filter. In our approach, σ_g is one of the parameters optimized for the candidate filter with the method discussed in Sec. 6.1.1. This allows the system to utilize the power of both bilateral and non-local means filters when necessary.

Finally, the weight between pixels p and q for the other features (the cross terms of the filter) is defined as,

$$d_{p,q}^{\text{cross}} = \prod_{k=1}^{K} \exp\left[-\frac{D(\bar{f}_{p,k}, \bar{f}_{q,k})}{2\gamma_k^2} \right], \tag{6.5}$$

where $\bar{f}_{p,k}$ and $\bar{f}_{q,k}$ are the k^{th} additional features in pixels p and q, respectively, and γ_k^2 is the variance which sets the width of the filter for this feature. As in

Rousselle et al. [82], our distance function between features, $D(\bar{f}_{p,k}, \bar{f}_{q,i})$, is given by

$$D(\bar{f}_{p,k}, \bar{f}_{q,k}) = \frac{\|\bar{f}_{p,k} - \bar{f}_{q,k}\|^2}{\zeta[\bar{f}_{p,k}] + \zeta[\bar{f}_{q,k}]},$$

where $\zeta[\bar{f}_{p,k}] = \max(\psi[\bar{f}_{p,k}], \text{grad}[\bar{f}_{p,k}], \varepsilon)$ where $\psi[\bar{f}_{p,k}]$ is the variance of the k^{th} feature of all samples at pixel p, $\text{grad}[\bar{f}_{p,k}]$ is the squared magnitude of the gradient of the k^{th} feature in both x and y directions at pixel p, and ε is 10^{-3} to avoid division by zero. The final weight $d_{p,q}$ between pixel p and q that is used in Eq. 4.1 to compute the final filtered pixel is then,

$$d_{p,q} = d_{p,q}^{\text{NLM}} \times d_{p,q}^{\text{cross}}. \tag{6.6}$$

In our implementation, we use $K = 5$ additional features which include world coordinates, shading normals, texture colors for the first and second intersections, and direct illumination visibility. These features are pre-filtered using a non-local means filter in an 11×11 window with patch size 7×7.

In our system, $\boldsymbol{\theta}_i = \{\sigma_g^2, \beta^2, \gamma_1^2, \cdots, \gamma_K^2\}$ are the parameters that are optimized for each candidate filter using the technique discussed in Sec. 6.1.1. Note that when optimizing these parameters, they can become negative which makes the optimization unstable. Therefore, we introduce an auxiliary variable $\boldsymbol{\lambda}_i$ which

is related to the parameters through $\boldsymbol{\theta}_i = \exp(\boldsymbol{\lambda}_i)$. Optimizing $\boldsymbol{\lambda}_i$, avoids this problem as the exponential function is in the range $(0, \infty)$. Moreover, we used a constant learning rate ($\mu = 0.1$) for all parameters except the patch weight, σ_g, for which we used $\mu = 1.0$, since we found that this parameter was less sensitive to large changes relative to the other terms.

6.1.3 No-reference error metric

No-reference error metrics have been used in the previous approaches to approximate the mean squared error (MSE) between the filtered and ground truth images. For example, Rousselle et al. [80] and Moon et al. [66] used the definition of bias and variance to estimate MSE. However, in this work we use the more successful metric proposed by Li et al. [56] for MC denoising, which is Stein's unbiased risk estimator (SURE) [94].

6.1.4 Final image reconstruction

As previously discussed, the SURE estimate tends to be noisy and, thus, simply choosing the filter with the lowest estimated error at each pixel is not optimal. For the final image reconstruction (function R in Eq. 6.1), we use the reconstruction approach from Rousselle et al. [82]. Specifically, we apply a small non-local means filter (7×7 patches in a 3×3 window) to the initial SURE estimate. Next, we

compute an integer selection map denoting which of the N candidate filters has the lowest error at each pixel. Finally, we apply a large non-local means filter (7×7 patches in an 11×11 window) to smooth the selection map. We then obtain the filtered image by performing a weighted average of the candidate filters according to our selection map. Note that other ways of performing filter selection are possible, such as the graph cuts approach recently proposed by Bauszat et al. [4], and it is an interesting avenue of future work. After the filtered selection map is applied, we perform a quick second pass filter to remove small residual noise and obtain our final filtered result.

Furthermore, we eliminate high energy pixels, or spikes, since they are difficult to filter properly and result in artifacts. We use the approach of Kalantari et al. [45] (discussed in Chapter 5) for handling spikes, but instead perform it as a pre-process step rather than a post-process one. First, we find the mean and standard deviation of the noisy RGB colors for pixels in a surrounding block of size 41×41. Center pixels with any color channel more than 10 standard deviations away from the block mean are replaced by the block median. Furthermore, for each pixel detected as a spike, we also replace the corresponding pixel variance with the variance of the median pixel. Alternative approaches to deal with spikes are left for future investigation.

6.2 Extensions

We also extend our framework to perform adaptive sampling. We use the adaptive sampling scheme of Rousselle et al. [81, 82] where we do uniform sampling initially on a subset of the total budget and then use adaptive sampling for the remaining samples. For each adaptive iteration, we dedicate a portion of the total number of samples to evaluate regions with the highest reconstruction error. The adaptive sampling map is generated based on the SURE error obtained with one of the candidate filters for efficiency. The SURE error is then weighted by the filter weights in the neighborhood around a given pixel to evaluate the amount that the error can be reduced by sending another sample at that pixel. Finally, we filter this map, using the parameters from Rousselle et al. [82], to get a smooth sampling map. Note that we found that different choices for the candidate filter during the adaptive step did not make a significant difference quantitatively or qualitatively, so we simply choose the first filter candidate during adaptive sampling.

To handle animated sequences while maintaining temporal coherency, we extend our filter to work on spatio-temporal volumes. At each frame, we use the five previous frames and the five next frames (11 frames total) to generate the corresponding filtered frame. This simple extension greatly reduces the flickering and low frequency residual noise in the final videos.

6.3 Results

We implemented our method, called optimal filter parameter sets (OFS), in C++ as an extension to PBRT2 [73] and wrote the filters in CUDA for GPU acceleration. All timings were obtained on an Intel quad-core 3.7 GHz machine with 24GB of memory and a GeForce GTX TITAN GPU. We used a set of *two* candidate filters ($N = 2$) and performed the offline optimization over a set of 10 training scenes (see supplementary materials) with a variety of distributed effects that were a subset of the ones used by Kalantari et al. [45] (discussed in Chapter 5). As discussed in Sec. 6.1.1, we rendered the noisy scenes with 4, 8, 16, 32, and 64 samples per pixel for our training set. The optimization converged after approximately 25 iterations, which took roughly 1.5 days. This optimization was done a single time to generate the two optimal candidate filters, and we then used them to produce all the results. Naturally, the results shown here are of scenes that were not part of the training set.

We use two objective metrics to evaluate the quality of our results: 1) relative MSE [80] which is defined as RelMSE $= (\hat{c} - c)^2/(c^2 + \varepsilon)$, where $\hat{c}$ and c are the filtered and ground truth pixel colors, respectively, and ε is a small number (0.01) to avoid division by zero, and 2) structural similarity (SSIM) index [105] which is



Figure 6.1: We present equal-time comparisons of our approach (OFS) with the most recent state-of-the-art algorithms and equal-time Monte Carlo (ETMC) on scenes with a variety of distributed effects. Specifically, we compare our method against robust denoising using feature and color (RDFC) [82], weighted local regression (WLR) [66], and the learning-based filtering (LBF) [45] (discussed in Chapter 5). The relative MSE (lower is better), SSIM index (higher is better), and timings (in seconds) are also listed. Please see supplementary materials for full-size images and additional comparisons.

a value from 0 to 1, where 1 indicates perfect quality with respect to the ground truth image.

Fig. 6.1 compares our approach to several state-of-the-art methods on four scenes with different distributed effects. Specifically, we compare our optimal filter set (OFS) method against equal-time Monte Carlo (ETMC), robust denoising using feature and color (RDFC) [82], weighted local regression (WLR) [66], and the learning-based filtering (LBF) [45] (discussed in the previous chapter). We also compare against random parameter filtering (RPF) [90] and SURE-based filtering (SBF) [56] in the supplementary materials. For all comparisons, we used the publicly available implementation by the original authors themselves. Furthermore, except for RPF and LBF which are only post-process filtering approaches, all algorithms (including our own) were run with adaptive sampling enabled.

First, we examine the Sci-Fi City scene containing motion blur and global illumination. Both LBF and WLR have artifacts around the edges of highly textured structures like the spaceship and sphere. While RDFC is able to generate a smooth result, it fails to preserve fine texture detail such as the ridges on the spaceship and the edges of the walls. Our approach is able to keep textures while still being noise and artifact free.

The path traced Crytek Sponza scene also contains both motion blur and global illumination. The previous approaches all have artifacts throughout the

image. For example, although LBF is able to keep the textures on the floor, it has objectionable artifacts in the motion blur region of the plane. On the other hand, RDFC is able to smooth the plane appropriately, but has distracting blotches on the ground. As shown, our method closely resembles the ground truth in both areas.

Next, we compare our approach on the KITCHEN scene containing global illumination noise. None of the other approaches can efficiently remove the noise from the wall and the lamp without introducing artifacts. Moreover, other methods overblur the texture details on the cabinet and the wallpaper, while our result closely resembles the ground truth. Finally, the TEAPOT ROOM scene is an extremely noisy and challenging path-traced scene that includes global illumination and glossy reflections. Our approach keeps the details on the floor and the glossy reflection on the teapot better than all the other methods.

As discussed in Sec. 6.2, we can extend our filter spatio-temporally to be able to filter videos while maintaining temporal coherency. We use the same filter candidates from Fig. 6.1 on the CRYTEK SPONZA, SCI-FI CITY, and CONFERENCE video sequences at 16 samples per pixel, which can be found in the supplemental video. Despite the low sampling rates, our approach generates temporally coherent, high-quality results on very difficult scenes.

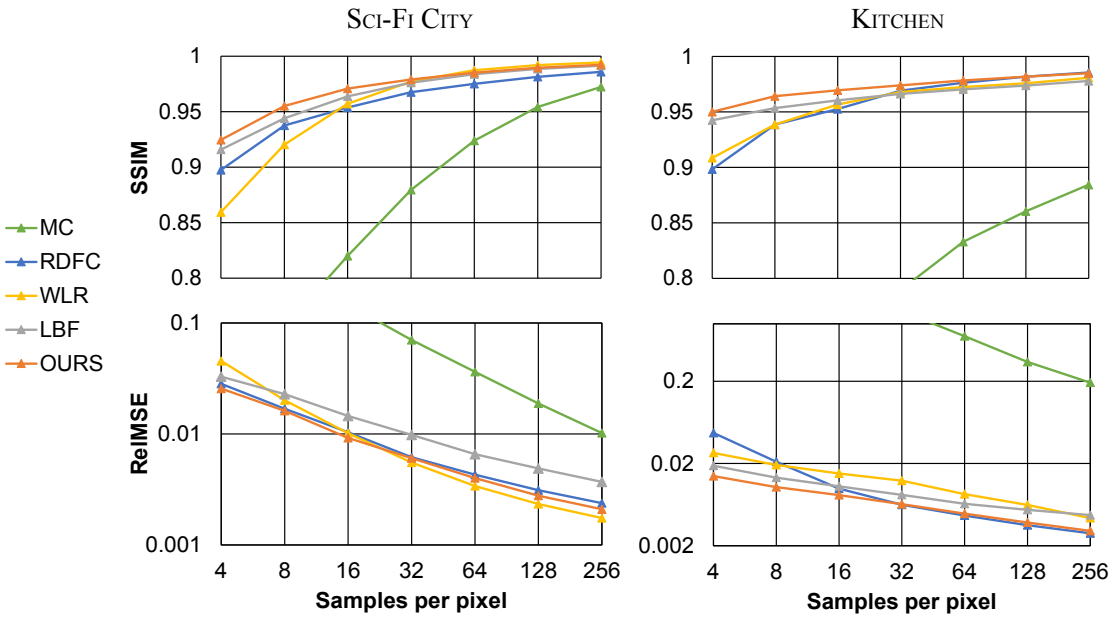


Figure 6.2: We show the convergence plot in terms of relative MSE and SSIM for the Sci-Fi City scene (left) and the Kitchen scene (right). Although we optimized our system on scenes with sampling rates 64 samples per pixel and below, our results consistently improve even for higher sampling rates, demonstrating the robustness of our optimization. Our approach produces results with better relative MSE and SSIM at low sampling rates. As the sampling rate increases, all the approaches converge to the ground truth. However, as shown in Fig. 6.3, our results have better visual quality at these sampling rates.

6.4 Discussion

Our algorithm produces high-quality results that outperform state-of-the-art algorithms despite using only two candidate filters. Previous approaches such as Li et al. [56] and Rousselle et al. [82] use seven and three candidates, respectively, yet still contain artifacts. This demonstrates that previous approaches do not utilize the full potential of their extra candidate filters, which are heuristically chosen

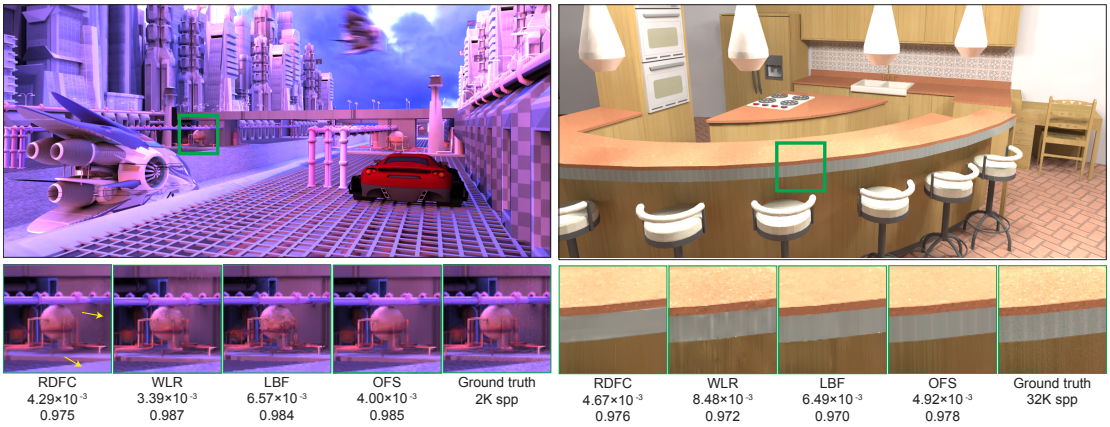


Figure 6.3: We compare the results of our method (OFS) against robust denoising using feature and color (RDFC) [82], weighted local regression (WLR) [66], and the learning-based filtering (LBF) [45] on the Sci-Fi City (left) and Kitchen (right) scenes rendered at 64 samples per pixel. Note that although WLR's result on the Sci-Fi City scene is slightly better numerically, it contains disturbing residual noise. Moreover, despite having a slightly better relative MSE on the Kitchen scene, RDFC cannot keep the textures on the counter properly. Our approach produces results with better visual quality and fewer artifacts on both scenes.

and, thus, are sub-optimal. Our proposed optimization pre-process is therefore critical in these error-minimization filtering approaches to gain the most benefit from each of the individual filters. Indeed, we tested our framework with $N = 3$ candidate filters and did not notice a significant difference in quality, indicating that $N = 2$ candidate filters, if optimized properly, is sufficient for producing high-quality results.

Table 6.1 shows the final optimized filter parameters for the two candidate filters in our approach. The two candidates have significantly dissimilar values and ranges for the various terms allowing them to successfully filter different

Filter Term	Parameter	Candidate 1	Candidate 2
Color	(σ_c^2)	1.05e+07	0.23
Patch Size	(σ_g^2)	1.44	7.00e+03
World Coordinates	$(\sigma_{f_1}^2)$	0.01	0.68
Shading Normals	$(\sigma_{f_2}^2)$	0.08	2.60
1st Bounce Texture	$(\sigma_{f_3}^2)$	0.05	0.49
2nd Bounce Texture	$(\sigma_{f_4}^2)$	3.02	0.22
Direct Illumination Visibility	$(\sigma_{f_5}^2)$	12.55	0.39

Table 6.1: Final filter parameters for each filter term in our $N = 2$ candidate filters.

regions of the noisy image, as demonstrated for the Crytek Sponza scene in Fig. 6.4. Here, candidate 1 is able to keep the scene detail on the curtains, pillars, and floor because of the 1st bounce texture term, while candidate 2 overblurs the textures but is able to clean up the residual noise. In general, for our two filter candidates, one filter is tighter than the other and will filter a particular feature more aggressively. Specifically, candidate 1 has a wide filter for the color dimension. In other words, the filter ignores color when filtering. It is interesting to note that RDFC has one of its three filter candidates ignore the color term as well. However, this candidate was designed heuristically, while our method established such a relationship systematically. Moreover, the patch size is made very wide in candidate 2, effectively turning it into a box filter. Meanwhile, for the other filter, the patch size is much smaller thus making it more closely resemble a cross-bilateral filter.

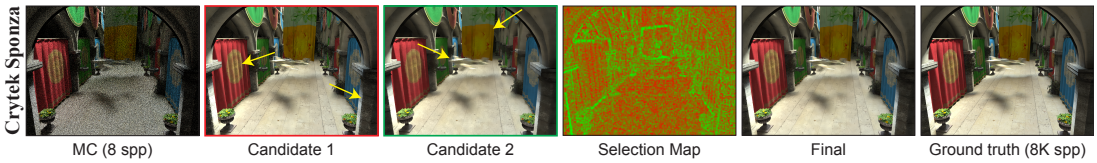


Figure 6.4: We demonstrate the importance of having our two candidate filters to appropriately denoise different regions of the image. In particular, candidate 1 is able preserve high frequency scene detail at the cost of some residual noise. On the other hand, candidate 2 overblurs these texture to effectively remove noise from the smooth regions. Each result on its own is unacceptable, but our final reconstruction uses the selection map to generate a high-quality output by combining the best regions of the two candidates.

To test the convergence of our method as the sampling rate increases, we provide convergence plots in Fig. 6.2 using relative MSE and SSIM metrics for the Sci-Fi City and Kitchen scenes and provide comparisons against low-discrepancy samples, RDFC, WLR, and LBF. As can be seen, our result continues to improve even at the sampling rates that we did not optimize on, demonstrating the generality and robustness of our approach. Moreover, our algorithm produces better results than others in terms of both relative MSE and SSIM at low sampling rates. Although all these algorithms converge to the ground truth as sampling rate increases (i.e., they are consistent), our method typically produces results with better visual quality, as shown in Fig. 6.3.

6.5 Conclusion

We have presented an approach for obtaining optimal candidate filter parameter sets in an error-minimization framework for filtering Monte Carlo noise. First, we perform offline optimization to find a set of optimal candidate filters by minimizing the error between the filtered and ground truth images on a set of training scenes. Moreover, by adding an extra parameter to the cross non-local means filter to allow for a smooth transition to a cross-bilateral filter, we utilize the power of both filters in our system. Once the optimal candidate filters are obtained, we can then use them to denoise any new test image. Using only two filter candidates, we have demonstrated substantial improvement over state-of-the-art methods both visually and numerically.

Chapter 7

Discussion and future work

As discussed in Chapter 4, predicting optimal filter parameters is the main challenge and the goal of all filtering approaches. Moreover, the approaches in the two categories of direct estimation (e.g., learning-based filtering (LBF) discussed in Chapter 5) and error-minimization (e.g., optimal filter sets (OFS) discussed in Chapter 6), differ in the way of estimating these parameters at each pixel. Despite this difference, we observe that there is a connection between the two approaches and, thus, analyze them under the unified framework shown in Fig. 7.1. This helps us identify the advantages and disadvantages of each approach and discover the potential avenues for future work.

All the filtering approaches have the following three main components: 1) a parameter estimator outputs a set of N filter parameters at each pixel, 2) a filter takes these parameters as well as the local primary features to produce a set of N filtered pixel colors, and 3) a "reconstructor" combines these filtered pixels to

produce the final pixel value. In this system, the filter is fixed and every approach considers a specific general form for it. The main difference between the direct estimation and error-minimization approaches is in the design of the parameter estimation and reconstruction units.

Error-minimization approaches like OFS define a fixed set of N filter parameters ($N > 1$) for all the images and the reconstructor chooses the best filter parameters from this set of N candidate filter parameters. On the other hand, direct estimation approaches like LBF estimate a single set of filter parameters ($N = 1$) at each pixel independently and, thus, the reconstructor simply outputs the filtered pixel color. At a high level, the error-minimization techniques select the optimal filter parameters from a fixed discrete set of N filtered parameters, while the direct estimation approaches attempt to estimate the optimal filter parameters directly from the output of the MC renderer.

Therefore, at first glance, it may seem like LBF is more flexible and powerful than OFS. However, as discussed in Chapter 5, LBF relies heavily on an effective training stage which is difficult due to a major practical limitation. Although a big network can be used to model the complex function $\mathcal{G}$ for estimating the optimal filter parameters at each pixel, a large number of training scenes are required to train this network appropriately. With a small number of scenes, however, a large network overfits to the training scenes and performs poorly on the test

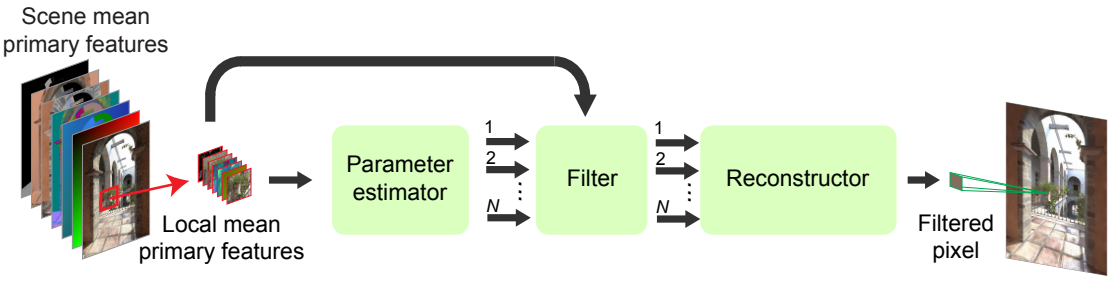


Figure 7.1: We show the general framework for all filtering approaches. A filter parameter estimator first predicts a set of N filter parameters at each pixel. These parameters as well as the local primary features are then given to the filter to output a set of N filtered pixels. Finally, the reconstructor combines these filtered pixels based on their error to produce the final pixel value. In this thesis, we observe that the main drawback of previous approaches is that they heuristically design the filter parameter estimation unit. Therefore, we propose to use machine learning to address this challenging problem for the two major classes of approaches. In the future, it would be interesting to use machine learning to improve other components of the filtering system, i.e., filter and reconstructor.

data. Since in our system we had access to a limited number of training scenes (20 scenes), we used a reasonably small network to avoid overfitting. Therefore, our network produced a smooth approximation to the true complex function $\mathcal{G}$ to create a balance between all the training data without overfitting. As a result, the estimated parameters on the test scenes may not be always optimal. However, as shown in Chapter 5, our system is very effective in removing noise from MC images despite being trained on a small number of scenes.

On the other hand, OFS only requires a set of N fixed filter parameters for all the images. Therefore, training this system is easier than the neural network in LBF and, thus, even 10 training scenes were sufficient for effective optimization.

Note that in OFS the reconstructor selects the best optimal filter parameters and, thus, the training stage is less critical than LBF. However, the main limitation of this approach is that it requires a no-reference error estimation metric, which could be noisy at low sampling rates. Moreover, although the set of filter parameters are optimized over a training set, they are fixed for all the images and the optimum filter parameters at each pixel may not be in the set.

In this thesis, we made an initial attempt to use machine learning for removing MC noise. However, there are several interesting avenues for future work. We believe machine learning can be used in the reconstruction stage to address the problem with noisy and inaccurate error estimation metrics. We observe that there are multiple ways to do this and it would be beneficial to investigate these approaches in the future. One way is to replace the error estimation metric with a machine learning model (e.g., a neural network). The model takes a set of secondary features (e.g., SURE, noise estimate, variances, etc.) from the filtered image and predicts the error. In this case, since the model utilizes more features, the prediction could be more accurate than the analytical error estimation metrics like SURE.

Another way to improve the reconstructor using machine learning is to train a learning model to simply choose the best filtered image at each pixel among a set of N filtered images. Note that, the goal of the reconstructor is to choose the

best filter parameters and the error estimation metric is simply used to evaluate the quality of the filtered images. Therefore, computing the actual error of each filtered image is not necessary and, as a result, we expect this training to be simpler than the first case. Finally, we can compute the final pixel value by computing a weighted average of all the filtered images, as is done in OFS and Rousselle et al. [82]. However, instead of using an error estimation metric to compute the weights, it is possible to train a model to predict them. In this case, the output of the model would be N floating point values which are used to weight the pixel values from the N filtered images. This is potentially the best way to perform the reconstruction in a more systematic way since the training is performed by minimizing the error between the final reconstructed and ground truth images, the actual goal of the reconstruction.

Currently, all the error-minimization techniques, including OFS, use a fixed set of N candidate filter parameters, which can be problematic as explained above. In the future, it is possible to use machine learning to predict a set of N filter parameters at each pixel and then use the reconstruction stage to select the best one among them. Here, instead of having a fixed set of candidate filter parameters for all the images, the model predicts them based on the features extracted from a local neighborhood around each pixel.

Moreover, all the filtering techniques, including our approaches, consider a general form for the filter (e.g., cross-bilateral or cross non-local means filters), which could be suboptimal. In the future, it would be interesting to train a machine learning model to take the output of the MC renderer and directly predict the final pixel value. This is, however, challenging and might not be possible in the near future due to several practical limitations. First, training such a complicated system requires a big model with lots of parameters and, thus, as discussed earlier, a significant number of training scenes is needed. However, obtaining a huge database of training scenes is challenging. Furthermore, since the network performs the filtering in this case, the number of inputs are large (e.g., all the primary features for a 55×55 block). Therefore, training this system is computationally expensive and, given the current available hardware, is almost intractable. Finally, although similar systems using neural networks have been tried for image denoising [8], they have been applied in constrained situations due to difficulties in the training. For example, Burger et al. train a single neural network for a specific noise standard deviation to achieve reasonable performance. Since in MC rendering noise is spatially-varying, this could be even more problematic.

In LBF, we used a neural network as our nonlinear regression model since it was simple, effective, and fast. However, it might be possible to replace the neural network with another nonlinear regression model, such as a support vector

machine [23, 95]. Moreover, in this thesis we only focused on using machine learning for the filtering stage, but it would be interesting to investigate possibility of using a learning approach for adaptive sampling as well.

There is significant future work beyond the application of machine learning. First, most of the MC filtering techniques, including ours, focus on image filtering. For animated sequences, they simply extend the filter to work in spatio-temporal volumes. However, we believe these current techniques do not fully utilize the additional information which is available in the video. Although recently there has been an attempt to specifically handle videos [109], we think there is still room to explore this fairly untouched problem. This could be possible by borrowing ideas from the image processing community which have been looking at the video denoising problem for years. For example, Liu and Freeman [57] proposed to use optical flow to estimate motion between frames to guide the filtering process and a similar strategy can be used to remove noise from MC rendered videos.

Second, recently there has been a lot of interest in real-time ray tracing rendering systems with the introduction of powerful GPU ray tracers such as OptiX [71] and Embree [104]. However, for complex scenes, these renderers produce noisy images and a filtering approach is required to produce a smooth, noise-free video. The approaches proposed in this thesis are designed for offline filtering, but developing fast filtering approaches is a great avenue for future research.

Finally, most of the current filtering approaches perform the process on the full resolution image. However, we believe there is a significant benefit in performing the filtering and adaptive sampling process at multiple resolutions. This has been proven to be beneficial in image denoising with a few recent approaches [58, 110]. For example, Zontak et al. [110] proposed a directional pyramid technique to find patch correspondences across scales. A similar strategy could be investigated in MC filtering to improve the quality of the filtered results. Moreover, adaptive sampling can be performed in multiple scales to avoid dedicating a lot of samples to the smooth regions of the image. These regions usually can be sampled and reconstructed at the coarser scales and the result can be used at finer scales.

Chapter 8

Conclusion

In this thesis, we first studied the application of standard image denoising techniques to the problem of MC noise reduction. To do this, we proposed a general framework to use any powerful image denoising technique to remove general MC noise and showed that our approach can produce better results than the previous color-based schemes. We then demonstrated that this approach cannot produce satisfactory results in challenging cases and it is important to utilize additional features, such as world positions, shading normals, and texture values, in the filtering process.

Therefore, in Chapter 4, we presented a new way of looking at the MC filtering problem and identified filter parameter estimation to be the main challenge for all filtering techniques. We then categorized the previous filtering techniques into two main classes based on how they estimate the filter parameters. To address the challenging filter parameter estimation problem, we proposed to use machine

learning in two major MC filtering frameworks. First, in Chapter 5, we proposed to use machine learning to directly set the filter parameters of feature-based filters such as the cross-bilateral and cross non-local means filters in a systematic way. We proposed to train a multilayer perceptron neural network to produce filtered results close to ground truth images according to a specific error metric. Second, in Chapter 6, we proposed to optimize the candidate filter parameters of an error-minimization filtering framework on a set of training scenes.

Finally, we showed the connection between the two proposed approaches and discussed the advantages and disadvantages of each method in Chapter 7. We then described possible future work to use machine learning for addressing the current issues with these systems. Furthermore, we proposed several general research directions and open problems for future investigation.

Overall, our key contribution to the field of MC filtering was to demonstrate the first use of machine learning in removing MC noise. We hope this thesis is the beginning of a new trend in the rendering community and other researchers can build upon the ideas presented here.

Bibliography

[1] Timo Aila and Samuli Laine. Understanding the efficiency of ray traversal on gpus. In *Proceedings of the Conference on High Performance Graphics 2009*, HPG '09, pages 145–149, New York, NY, USA, 2009. ACM.

[2] Kavita Bala, Bruce Walter, and Donald P. Greenberg. Combining edges and points for interactive high-quality rendering. *ACM Trans. Graph.*, 22(3):631–640, July 2003.

[3] P. Bauszat, M. Eisemann, S. John, and M. Magnor. Sample-based manifold filtering for interactive global illumination and depth of field. *Computer Graphics Forum*, 34(1):265–276, 2015.

[4] Pablo Bauszat, Martin Eisemann, Elmar Eisemann, and Marcus Magnor. General and robust error estimation and reconstruction for Monte Carlo rendering. *Computer Graphics Forum (Proc. of Eurographics EG)*, 34(2), January 2015.

[5] Pablo Bauszat, Martin Eisemann, and Marcus Magnor. Guided image filtering for interactive high-quality global illumination. *Computer Graphics Forum*, 30(4):1361–1368, 2011.

[6] Laurent Belcour, Cyril Soler, Kartic Subr, Nicolas Holzschuch, and Fredo Durand. 5d covariance tracing for efficient defocus and motion blur. *ACM Trans. Graph.*, 32(3):31:1–31:18, July 2013.

[7] A. Buades, B. Coll, and J. M. Morel. A review of image denoising algorithms, with a new one. *Multiscale Modeling & Simulation*, 4(2):490–530, 2005.

[8] H.C. Burger, C.J. Schuler, and S. Harmeling. Image denoising: Can plain neural networks compete with BM3D? In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2392–2399, June 2012.

[9] Nathan A. Carr, Jesse D. Hall, and John C. Hart. The ray engine. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, HWWS '02, pages 37–46, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[10] Jin-Xiang Chai, Xin Tong, Shing-Chow Chan, and Heung-Yeung Shum. Plenoptic sampling. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, pages 307–318, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[11] Jiating Chen, Bin Wang, Yuxiang Wang, Ryan S. Overbeck, Jun-Hai Yong, and Wenping Wang. Efficient depth-of-field rendering with adaptive sampling and multiscale reconstruction. *Computer Graphics Forum*, 30(6):1667–1680, 2011.

[12] Petrik Clarberg, Wojciech Jarosz, Tomas Akenine-Möller, and Henrik Wann Jensen. Wavelet importance sampling: Efficiently evaluating products of complex functions. *ACM Trans. Graph.*, 24(3):1166–1175, July 2005.

[13] Robert L. Cook. Stochastic sampling in computer graphics. *ACM Trans. Graph.*, 5:51–72, January 1986.

[14] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. *SIGGRAPH Comput. Graph.*, 18(3):137–145, January 1984.

[15] Kostadin Dabov, Alessandro Foi, and Karen Egiazarian. Video denoising by sparse 3d transform-domain collaborative filtering. In *Proc. 15th European Signal Processing Conference*, pages 145–149, September 2007.

[16] Kostadin Dabov, Alessandro Foi, Vladimir Katkovnik, and Karen Egiazarian. Image denoising with block-matching and 3d filtering. In *Electronic Imaging'06*, 2006.

[17] C. Dachsbacher. Analyzing visibility configurations. *IEEE Transactions on Visualization and Computer Graphics*, 17(4):475–486, April 2011.

[18] Holger Dammertz, Daniel Sewtz, Johannes Hanika, and Hendrik P.A. Lensch. Edge-avoiding À-trous wavelet transform for fast global illumination filtering. In *Proceedings of High Performance Graphics 2010*, pages 67–75, 2010.

[19] Mauricio Delbracio, Pablo Musé, Antoni Buades, Julien Chauvier, Nicholas Phelps, and Jean-Michel Morel. Boosting Monte Carlo rendering by ray histogram fusion. *ACM Trans. Graph.*, 33(1):8:1–8:15, February 2014.

[20] Mark A. Z. Dippé and Erling Henry Wold. Antialiasing through stochastic sampling. *SIGGRAPH Comput. Graph.*, 19(3):69–78, July 1985.

[21] David L. Donoho and Jain M. Johnstone. Ideal spatial adaptation by wavelet shrinkage. *Biometrika*, 81(3):425–455, 1994.

[22] D.L. Donoho. De-noising by soft-thresholding. *IEEE Transactions on Information Theory*, 41(3):613–627, may 1995.

[23] Harris Drucker, Chris J. C. Burges, Linda Kaufman, Alex Smola, and Vladimir Vapnik. Support vector regression machines. In *Advances in Neural Information Processing Systems 9*, pages 155–161. MIT Press, 1997.

[24] Frédo Durand, Nicolas Holzschuch, Cyril Soler, Eric Chan, and François X. Sillion. A frequency analysis of light transport. *ACM Trans. Graph.*, 24(3):1115–1126, July 2005.

[25] Philip Dutré, Kavita Bala, Philippe Bekaert, and Peter Shirley. *Advanced Global Illumination*. AK Peters Ltd, 2006.

[26] Kevin Egan, Frédo Durand, and Ravi Ramamoorthi. Practical filtering for efficient ray-traced directional occlusion. *ACM Trans. Graph.*, 30(6):180:1–180:10, December 2011.

[27] Kevin Egan, Florian Hecht, Frédo Durand, and Ravi Ramamoorthi. Frequency analysis and sheared filtering for shadow light fields of complex occluders. *ACM Trans. Graph.*, 30(2):9:1–9:13, April 2011.

[28] Kevin Egan, Yu-Ting Tseng, Nicolas Holzschuch, Frédo Durand, and Ravi Ramamoorthi. Frequency analysis and sheared reconstruction for rendering motion blur. *ACM Trans. Graph.*, 28(3):93:1–93:13, July 2009.

[29] Elmar Eisemann and Frédo Durand. Flash photography enhancement via intrinsic relighting. *ACM Trans. Graph.*, 23(3):673–678, August 2004.

[30] A. Fujimoto, T. Tanaka, and K. Iwata. Arts: Accelerated ray-tracing system. *Computer Graphics and Applications, IEEE*, 6(4):16–26, April 1986.

[31] Kirill Garanzha and Charles Loop. Fast ray sorting and breadth-first packet traversal for gpu ray tracing. *Computer Graphics Forum*, 29(2):289–298, 2010.

[32] Andrew S. Glassner, editor. *An Introduction to Ray Tracing*. Academic Press Ltd., London, UK, UK, 1989.

[33] A.S. Glassner. Space subdivision for fast ray tracing. *Computer Graphics and Applications, IEEE*, 4(10):15–24, Oct 1984.

[34] Christiaan Gribble and Alexis Naveros. Gpu ray tracing with rayforce. In *ACM SIGGRAPH 2013 Posters*, SIGGRAPH '13, pages 98:1–98:1, New York, NY, USA, 2013. ACM.

[35] Radek Grzeszczuk, Demetri Terzopoulos, and Geoffrey Hinton. Neuroanimator: Fast neural network emulation and control of physics-based models. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, pages 9–20, New York, NY, USA, 1998. ACM.

[36] Toshiya Hachisuka, Wojciech Jarosz, Richard Peter Weistroffer, Kevin Dale, Greg Humphreys, Matthias Zwicker, and Henrik Wann Jensen. Multidimensional adaptive sampling and reconstruction for ray tracing. *ACM Trans. Graph.*, 27:33:1–33:10, August 2008.

[37] Trevor Hastie, Robert Tibshirani, Jerome Friedman, T Hastie, J Friedman, and R Tibshirani. *The elements of statistical learning*, volume 2. Springer, 2009.

[38] J. Havel and A. Herout. Yet faster ray-triangle intersection (using sse4). *Visualization and Computer Graphics, IEEE Transactions on*, 16(3):434–438, May 2010.

[39] Vlastimil Havran. *Heuristic Ray Shooting Algorithms*. Ph.d. thesis, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, November 2000.

[40] Wenzel Jakob, Christian Regg, and Wojciech Jarosz. Progressive expectation-maximization for hierarchical volumetric photon mapping. *Computer Graphics Forum*, 30(4):1287–1297, 2011.

[41] Jeremy Jancsary, Sebastian Nowozin, and Carsten Rother. Loss-specific training of non-parametric image restoration models: A new state of the art. In *Proceedings of the 12th European Conference on Computer Vision - Volume Part VII*, ECCV'12, pages 112–125, Berlin, Heidelberg, 2012. Springer-Verlag.

[42] Henrik Wann Jensen. Global illumination using photon maps. In *Proceedings of the Eurographics Workshop on Rendering Techniques '96*, pages 21–30, London, UK, UK, 1996. Springer-Verlag.

[43] Henrik Wann Jensen and Niels Jorgen Christensen. Optimizing path tracing using noise reduction filters. In *Winter School of Computer Graphics (WSCG) 1995*, pages 134–142, 1995.

[44] James T. Kajiya. The rendering equation. In *SIGGRAPH '86*, pages 143–150, New York, NY, USA, 1986. ACM.

[45] Nima Khademi Kalantari, Steve Bako, and Pradeep Sen. A machine learning approach for filtering Monte Carlo noise. *ACM Transactions on Graphics (TOG) (Proceedings of SIGGRAPH 2015)*, 34(4), 2015.

[46] Nima Khademi Kalantari and Pradeep Sen. Efficient computation of blue noise point sets through importance sampling. *Computer Graphics Forum*, 30(4):1215–1221, 2011.

[47] Nima Khademi Kalantari and Pradeep Sen. Removing the noise in Monte Carlo rendering with general image denoising algorithms. *Computer Graphics Forum*, 32(2pt1):93–102, 2013.

[48] Timothy L. Kay and James T. Kajiya. Ray tracing complex scenes. *SIGGRAPH Comput. Graph.*, 20(4):269–278, August 1986.

[49] Johannes Kopf, Daniel Cohen-Or, Oliver Deussen, and Dani Lischinski. Recursive wang tiles for real-time blue noise. *ACM Trans. Graph.*, 25:509–518, July 2006.

[50] Eric P Lafortune and Yves D Willems. Bi-directional path tracing. In *Proceedings of CompuGraphics*, volume 93, pages 145–153, 1993.

[51] Samuli Laine, Hannu Saransaari, Janne Kontkanen, Jaakko Lehtinen, and Timo Aila. Incremental instant radiosity for real-time indirect illumination. In *Proceedings of the 18th Eurographics Conference on Rendering Techniques*, EGSR'07, pages 277–286, Aire-la-Ville, Switzerland, Switzerland, 2007.

[52] Yann Le Cun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural Networks, Tricks of the Trade*, Lecture Notes in Computer Science LNCS 1524. Springer Verlag, 1998.

[53] M.E. Lee and R.A. Redner. A note on the use of nonlinear filtering in computer graphics. *IEEE Computer Graphics and Applications*, 10(3):23–29, May 1990.

[54] Jaakko Lehtinen, Timo Aila, Jiawen Chen, Samuli Laine, and Frédo Durand. Temporal light field reconstruction for rendering distribution effects. *ACM Trans. Graph.*, 30(4):55:1–55:12, August 2011.

[55] Jaakko Lehtinen, Timo Aila, Samuli Laine, and Frédo Durand. Reconstructing the indirect light field for global illumination. *ACM Trans. Graph.*, 31(4):51:1–51:10, July 2012.

[56] Tzu-Mao Li, Yu-Ting Wu, and Yung-Yu Chuang. Sure-based optimization for adaptive sampling and reconstruction. *ACM Trans. Graph.*, 31(6):194:1–194:9, November 2012.

[57] Ce Liu and William T. Freeman. A high-quality video denoising algorithm based on reliable motion estimation. In *Proceedings of the 11th European Conference on Computer Vision Conference on Computer Vision: Part III*, ECCV'10, pages 706–719, Berlin, Heidelberg, 2010. Springer-Verlag.

[58] Ziwei Liu, Lu Yuan, Xiaoou Tang, Matt Uyttendaele, and Jian Sun. Fast burst images denoising. *ACM Trans. Graph.*, 33(6):232:1–232:9, November 2014.

[59] Michael D. McCool. Anisotropic diffusion for Monte Carlo noise reduction. *ACM Trans. Graph.*, 18(2):171–194, April 1999.

[60] Soham Mehta, Brandon Wang, and Ravi Ramamoorthi. Axis-aligned filtering for interactive sampled soft shadows. *ACM Trans. Graph.*, 31(6):163:1–163:10, Nov 2012.

[61] Soham Uday Mehta, Brandon Wang, Ravi Ramamoorthi, and Fredo Durand. Axis-aligned filtering for interactive physically-based diffuse indirect lighting. *ACM Trans. Graph.*, 32(4):96:1–96:12, July 2013.

[62] Soham Uday Mehta, JiaXian Yao, Ravi Ramamoorthi, and Fredo Durand. Factored axis-aligned filtering for rendering multiple distribution effects. *ACM Trans. Graph.*, 33(4):57:1–57:12, July 2014.

[63] Don P. Mitchell. Generating antialiased images at low sampling densities. *SIGGRAPH Comput. Graph.*, 21(4):65–72, August 1987.

[64] Don P. Mitchell. Consequences of stratified sampling in graphics. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 277–280, New York, NY, USA, 1996. ACM.

[65] Tomas Möller and Ben Trumbore. Fast, minimum storage ray-triangle intersection. *J. Graph. Tools*, 2(1):21–28, October 1997.

[66] Bochang Moon, Nathan Carr, and Sung-Eui Yoon. Adaptive rendering based on weighted local regression. *ACM Trans. Graph.*, 33(5):170:1–170:14, September 2014.

[67] Bochang Moon, Jose A. Iglesias-Guitian, Sung-Eui Yoon, and Kenny Mitchell. Adaptive rendering with linear predictions. *ACM Transactions on Graphics (TOG) (Proceedings of SIGGRAPH 2015)*, 34(4), 2015.

[68] Derek Nowrouzezahrai, Evangelos Kalogerakis, and Eugene Fiume. Shadowing dynamic scenes with arbitrary BRDFs. *Computer Graphics Forum*, 28(2):249–258, 2009.

[69] Ryan Overbeck, Ravi Ramamoorthi, and William R. Mark. Large ray packets for real-time Whitted ray tracing. In *IEEE/EG Symposium on Interactive Ray Tracing (IRT)*, pages 41–48, Aug 2008.

[70] Ryan S. Overbeck, Craig Donner, and Ravi Ramamoorthi. Adaptive wavelet rendering. *ACM Trans. Graph.*, 28(5):140:1–140:12, December 2009.

[71] Steven G. Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, and Martin Stich. Optix: A general purpose ray tracing engine. *ACM Trans. Graph.*, 29(4):66:1–66:13, July 2010.

[72] Georg Petschnigg, Richard Szeliski, Maneesh Agrawala, Michael Cohen, Hugues Hoppe, and Kentaro Toyama. Digital photography with flash and no-flash image pairs. *ACM Trans. Graph.*, 23(3):664–672, August 2004.

[73] Matt Pharr and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, second edition, 2010.

[74] Stefan Popov, Iliyan Georgiev, Rossen Dimov, and Philipp Slusallek. Object partitioning considered harmful: Space subdivision for bvhs. In *Proceedings of the Conference on High Performance Graphics 2009*, HPG '09, pages 15–22, New York, NY, USA, 2009. ACM.

[75] J. Portilla, V. Strela, M.J. Wainwright, and E.P. Simoncelli. Image denoising using scale mixtures of Gaussians in the wavelet domain. *Image Processing, IEEE Transactions on*, 12(11):1338 – 1351, November 2003.

[76] Timothy J. Purcell, Ian Buck, William R. Mark, and Pat Hanrahan. Ray tracing on programmable graphics hardware. *ACM Trans. Graph.*, 21(3):703–712, July 2002.

[77] Ravi Ramamoorthi and Pat Hanrahan. A signal-processing framework for inverse rendering. In *Proceedings of the 28th Annual Conference on Com-*

puter Graphics and Interactive Techniques, SIGGRAPH '01, pages 117–128, New York, NY, USA, 2001. ACM.

[78] Peiran Ren, Jiaping Wang, Minmin Gong, Stephen Lin, Xin Tong, and Baining Guo. Global illumination with radiance regression functions. *ACM Trans. Graph.*, 32(4):130:1–130:12, July 2013.

[79] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: the RPROP algorithm. In *IEEE International Conference on Neural Networks*, pages 586–591 vol.1, 1993.

[80] Fabrice Rousselle, Claude Knaus, and Matthias Zwicker. Adaptive sampling and reconstruction using greedy error minimization. *ACM Trans. Graph.*, 30(6):159:1–159:12, December 2011.

[81] Fabrice Rousselle, Claude Knaus, and Matthias Zwicker. Adaptive rendering with non-local means filtering. *ACM Trans. Graph.*, 31(6):195:1–195:11, November 2012.

[82] Fabrice Rousselle, Marco Manzi, and Matthias Zwicker. Robust denoising using feature and color information. *Computer Graphics Forum*, 32(7):121–130, 2013.

[83] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, October 1986.

[84] Holly E. Rushmeier and Gregory J. Ward. Energy preserving non-linear filters. In *ACM SIGGRAPH '94*, pages 131–138, New York, NY, USA, 1994.

[85] K.G.G. Samuel and M.F. Tappen. Learning optimized map estimates in continuously-valued mrf models. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 477–484, June 2009.

[86] Uwe Schmidt, Jeremy Jancsary, Sebastian Nowozin, Stefan Roth, and Carsten Rother. Cascades of regression tree fields for image restoration. *CoRR*, abs/1404.2086, 2014.

[87] B. Segovia, J. C. Iehl, R. Mitanchey, and B. Péroche. Non-interleaved deferred shading of interleaved sample patterns. In *Proceedings of the 21st ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware*, GH '06, pages 53–60, New York, NY, USA, 2006. ACM.

[88] Pradeep Sen and Soheil Darabi. Compressive estimation for signal integration in rendering. In *Proceedings of the 21st Eurographics Conference on*

Rendering, EGSR'10, pages 1355–1363, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association.

[89] Pradeep Sen and Soheil Darabi. Compressive rendering: A rendering application of compressed sensing. *IEEE Transactions on Visualization and Computer Graphics*, 17:487–499, 2011.

[90] Pradeep Sen and Soheil Darabi. On filtering the noise from the random parameters in Monte Carlo rendering. *ACM Trans. Graph.*, 31(3):18:1–18:15, June 2012.

[91] Peter Shirley. Discrepancy as a quality measure for sample distributions. In *Proc. Eurographics*, volume 91, pages 183–194, 1991.

[92] Peter Shirley, Timo Aila, Jonathan Cohen, Eric Enderton, Samuli Laine, David Luebke, and Morgan McGuire. A local image reconstruction algorithm for stochastic rendering. In *Symposium on Interactive 3D Graphics and Games*, I3D '11, pages 9–14 PAGE@5, New York, NY, USA, 2011. ACM.

[93] Cyril Soler, Kartic Subr, Frédo Durand, Nicolas Holzschuch, and François Sillion. Fourier depth of field. *ACM Trans. Graph.*, 28(2):1–12, 2009.

[94] Charles M. Stein. Estimation of the mean of a multivariate normal distribution. *The Annals of Statistics*, 9(6):pp. 1135–1151, 1981.

[95] J.A.K. Suykens and J. Vandewalle. Least squares support vector machine classifiers. *Neural Processing Letters*, 9(3):293–300, 1999.

[96] M.F. Tappen, Ce Liu, E.H. Adelson, and W.T. Freeman. Learning gaussian conditional random fields for low-level vision. In *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pages 1–8, June 2007.

[97] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Sixth International Conference on Computer Vision*, pages 839–846, Jan 1998.

[98] Yun-Ta Tsai, Markus Steinberger, Dawid Pajak, and Kari Pulli. Fast ANN for high-quality collaborative filtering. In *High-Performance Graphics*, 2014.

[99] Eric Veach and Leonidas J. Guibas. Optimally combining sampling techniques for monte carlo rendering. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, pages 419–428, New York, NY, USA, 1995. ACM.

[100] Eric Veach and Leonidas J. Guibas. Metropolis light transport. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, pages 65–76, New York, NY, USA, 1997. ACM.

[101] Ingo Wald, Solomon Boulos, and Peter Shirley. Ray tracing deformable scenes using dynamic bounding volume hierarchies. *ACM Trans. Graph.*, 26(1), January 2007.

[102] Ingo Wald, Thiago Ize, Andrew Kensler, Aaron Knoll, and Steven G. Parker. Ray tracing animated scenes using coherent grid traversal. *ACM Trans. Graph.*, 25(3):485–493, July 2006.

[103] Ingo Wald, Philipp Slusallek, Carsten Benthin, and Markus Wagner. Interactive rendering with coherent ray tracing. *Computer Graphics Forum*, 20(3):153–165, 2001.

[104] Ingo Wald, Sven Woop, Carsten Benthin, Gregory S. Johnson, and Manfred Ernst. Embree: A kernel framework for efficient cpu ray tracing. *ACM Trans. Graph.*, 33(4):143:1–143:8, July 2014.

[105] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, April 2004.

[106] Li-Yi Wei. Parallel Poisson disk sampling. *ACM Trans. Graph.*, 27:20:1–20:9, August 2008.

[107] Turner Whitted. An improved illumination model for shaded display. *Commun. ACM*, 23:343–349, June 1980.

[108] Ruifeng Xu and Sumanta N. Pattanaik. A novel Monte Carlo noise reduction operator. *IEEE Computer Graphics and Applications*, 25:31–35, 2005.

[109] Henning Zimmer, Fabrice Rousselle, Wenzel Jakob, Oliver Wang, David Adler, Wojciech Jarosz, Olga Sorkine-Hornung, and Alexander Sorkine-Hornung. Path-space motion estimation and decomposition for robust animation filtering. *Computer Graphics Forum*, 34(4):131–142, 2015.

[110] M. Zontak, I. Mosseri, and M. Irani. Separating signal from noise using patch recurrence across scales. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 1195–1202, June 2013.

[111] Matthias Zwicker, Wojciech Jarosz, Jaakko Lehtinen, Bochang Moon, Ravi Ramamoorthi, Fabrice Rousselle, Pradeep Sen, Cyril Soler, and Sung-Eui Yoon. Recent advances in adaptive sampling and reconstruction for Monte Carlo rendering. *Computer Graphics Forum (Proceedings of Eurographics)*, 34(2):667–681, May 2015.