Image Space Adaptive Rendering

Inauguraldissertation
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Fabrice Rousselle

von Kanada

Leiter der Arbeit:

Prof. Dr. M. Zwicker
Universität Bern

Dr. J. Lehtinen
Aalto University

Image Space Adaptive Rendering

Inauguraldissertation
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Fabrice Rousselle

von Kanada

Leiter der Arbeit:

Prof. Dr. M. Zwicker
Universität Bern

Dr. J. Lehtinen
Aalto University

Von der Philosophisch-naturwissenschaftlichen Fakultät
angenommen.

Bern, 2014

Der Dekan:
Prof. Dr. S. Decurtins

Acknowledgments

Many people have helped me in various ways during the duration of my thesis. First, I would like to thank my advisor, Matthias Zwicker, who has been a great guide and inspiration over the last four years. The extent and breadth of his knowledge and his understanding for the constraints of having a family made this PhD a pleasant and fruitful endeavor, for which I am deeply grateful. I would also like to thank the external reviewer, Jaakko Lehtinen, for taking the time to read and evaluate this thesis.

I would like to take the opportunity to thank the various members of the CGG. In particular, Claude Knaus who contributed to this thesis more than I can thank him for. I have greatly benefited from his skills and knowledge, and enjoyed his company. I would also like to particularly thank Marco Manzi, who also contributed greatly to this thesis. Finally I would like to thank the other members of the CGG, Daniel Donatsch for organizing the monthly PhD student meetings, Daljit Singh Dhillon for sharing our office so pleasantly, Sonja Schär, Wan-Yen Lo, and Peter Bertholet.

This work was financed in part by the Swiss National Science Foundation under grant no. 200021 127166. The various implementations of the framework presented in this thesis were done by extending the open source renderer PBRT by Greg Humphreys and Matt Pharr, which proved to be tremendously helpful.

I would like to thank my parents, Marie-France and Delphis Rousselle, who have supported me throughout the years, as well as my mother-in-law, Sarita Goutorbe, whose enthusiasm for my studies and generous personality have been greatly appreciated. Lastly, I would like to thank my wife, Cynthia, for her relentless support and for brightening my life. This thesis would not have existed without her, and I dedicate it to her.

Abstract

In this thesis, we develop an adaptive framework for Monte Carlo rendering, and more specifically for Monte Carlo Path Tracing (MCPT) and its derivatives. MCPT is attractive because it can handle a wide variety of light transport effects, such as depth of field, motion blur, indirect illumination, participating media, and others, in an elegant and unified framework. However, MCPT is a sampling-based approach, and is only guaranteed to converge in the limit, as the sampling rate grows to infinity. At finite sampling rates, MCPT renderings are often plagued by noise artifacts that can be visually distracting.

The adaptive framework developed in this thesis leverages two core strategies to address noise artifacts in renderings: adaptive sampling and adaptive reconstruction. Adaptive sampling consists in increasing the sampling rate on a per pixel basis, to ensure that each pixel value is below a predefined error threshold. Adaptive reconstruction leverages the available samples on a per pixel basis, in an attempt to have an optimal trade-off between minimizing the residual noise artifacts and preserving the edges in the image. In our framework, we greedily minimize the relative Mean Squared Error (MSE) of the rendering by iterating over sampling and reconstruction steps. Given an initial set of samples, the reconstruction step aims at producing the rendering with the lowest rMSE on a per pixel basis, and the next sampling step then further reduces the MSE by distributing additional samples according to the magnitude of the residual MSE of the reconstruction. This iterative approach tightly couples the adaptive sampling and adaptive reconstruction strategies, by ensuring that we only sample densely regions of the image where adaptive reconstruction cannot properly resolve the noise.

In a first implementation of our framework, we demonstrate the usefulness of our greedy error minimization using a simple reconstruction scheme leveraging a filterbank of isotropic Gaussian filters. In a second implementation, we integrate a powerful edge aware filter that can adapt to the anisotropy of the image. Finally, in a third implementation, we leverage auxiliary feature buffers that encode

scene information (such as surface normals, position, or texture), to improve the robustness of the reconstruction in the presence of strong noise.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

> There is this tremendous mess of waves all over in space, which is the light bouncing around the room, and going from one thing to the other.
>
> Richard Feynman

The aim for realism has, to various degrees, always been a part of the figurative arts, and culminated in the realist and hyperrealist movements. Throughout history, artists and scientists have attempted to formalize the rules of nature, in order to better represent it and understand it, and it is not surprising that the line between artist and scientist would sometimes blur, such as with Leonardo da Vinci. A striking example would be the discovery of linear perspective, which lead to the famous experiment of Filippo Brunelleschi, an architect, who painted around 1420 an exact replica of the Florentine Baptistery as viewed from a specific viewpoint.

While Brunelleschi was interested in architectural studies, his experiment actually had a profound impact on the art of the time, due to the unprecedented realism it allowed. As an example, in Figure 1.1, we have the study of a chalice in perspective by Paolo Uccello done in the 15th century, which is strangely reminiscent of the wireframe renderings of today's modeling software.

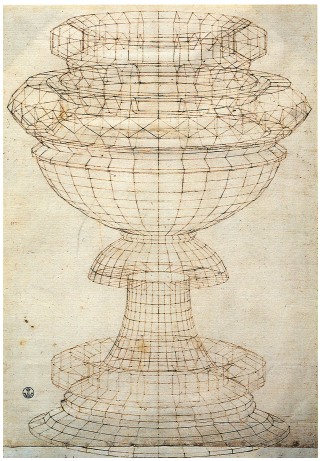


Figure 1.1: Paolo Uccello, *Perspective study of a chalice*, 15th century.

The 1960s, thanks to the rising power of computers, saw the first efforts in the field of computer graphics, and it is perhaps not coincidental that one of the pioneers in computer graphics, Donald Greenberg, is, like Brunelleschi, also an architect. As with linear perspective, computer graphics went on to have a profound impact on the arts and data visualization, and are now commonly used in the movie industry, scientific visualization, video games, virtual museums, etc.

As the need for photo-realistic renderings (i.e. renderings that would be indistinguishable from photographs) became more pressing, in good part motivated by the growing use of digital special effects in movies, the laws of optics were tightly integrated into rendering systems. This resulted in today's field of physically-based rendering, which aims at faithfully modeling all aspects of light transport, including the optics of the camera itself.

In order to make rendering tractable, it is common to consider only geometrical optics, where light is modeled as a particle and propagates along rays. This model cannot model wave optics effects such

as diffraction or interference, but otherwise offers a fairly accurate approximation of light transport. One can also model the interaction of light with the medium it propagates through using radiative transfer theory, that considers the medium emission, absorption, and scattering properties. These principles, geometrical optics and radiative transfer, have been elegantly combined in the shape of the rendering equation, as proposed in Kajiya's seminal work [Kaj86]. The geometrical optics model at the heart of the rendering equation naturally gave way to the concept of path tracing, where light is gathered at a receiver along specific directions. In practice, solving the rendering equation essentially consists in integrating, at every point of the scene, all the light that is coming from all directions, and which could have been emitted, reflected, scattered, etc. towards the receiver. However, despite the simplifying assumptions of geometrical optics, it is not possible to solve the rendering equation in the general case, and researchers have turned to numerical methods to evaluate it. One algorithm, Monte Carlo path tracing (MCPT) and its extensions, has proven to be particularly effective.

1.1 Monte Carlo Path Tracing

As the name suggests, MCPT combines Monte Carlo integration methods with the concept of path tracing. Monte Carlo integration is a well know approach to numerically integrate functions, and is based on random sampling. Essentially, it consists in averaging several samples taken from a function to estimate its integral. In path tracing, a sample is the contribution of a light path, i.e. the radiance carried along that path towards the camera. A light path is build by recursively tracing lights rays through the scene. Ray tracing is conceptually very simple: a light ray originates from a given point and is traced until it somehow interacts with the scene (by reflecting off a surface, refracting through glass, scattering in the fog, etc.). An early example of ray tracing, illustrating the long history of this technique in the context of realistic rendering, is given in Figure 1.2. As can be seen in

this figure, ray tracing faithfully captures the perspective view of complex objects. Ray tracing can be easily generalized to build complex paths that will handle a wide variety of light transport effects such as area lights, participating media (fog, water, etc.), depth of field (due to camera lens optics), indirect illumination, etc. To illustrate this flexibility, we show in Figure 1.3 an image generated using MCPT with the PBRT renderer [PH10] (PBRT being an acronym for Physically Based Ray Tracing). This figure also illustrates the scalability of Monte Carlo integration, since this scene has a 9-dimensional sampling space, where the sampling space dimensionality corresponds to the number of parameters of the simulation: 2d sampling of the image plane for anti-aliasing, 2d sampling of the lens to simulate depth of field, 2d sampling of the hemisphere to capture the indirect illumination, 1d sampling inside the fog to account for single scattering, and 2d sampling of the area light producing soft shadows.

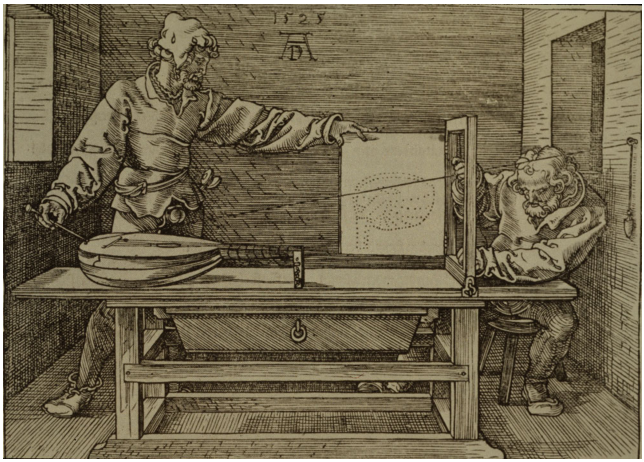


Figure 1.2: Albrecht Dürer, *Man drawing a lute*, 1525. Early application of the principle of path tracing to produce a realistic rendering.

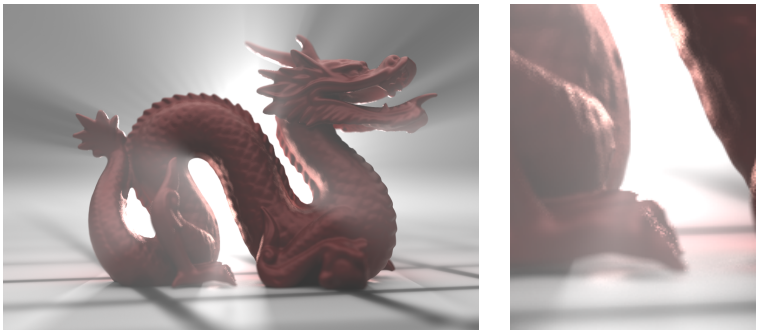


Figure 1.3: Physically-based rendering using Monte Carlo Path Tracing. This scene features: area lighting, participating media (fog), depth of field (simulating camera optics), and indirect illumination. This rendering was produced using 32000 paths per pixel, and took nearly 12 hours to compute.

The Appeal of Monte Carlo Path Tracing. There are three key features to MCPT that make it particularly appealing for photo-realistic rendering: (1) it is physically based, (2) it can simulate a wide variety of effects in a unified framework, and (3) it is guaranteed to converge (except in some pathological cases) to the right result. The first characteristic, being physically based, refers to the fact that MCPT directly builds upon the laws of geometrical optics. For instance, the direction of a refracted ray of light can be computed using the indices of refraction on both sides of the refractive surface, or the attenuation of light through a medium will be computed using Beer's law. Relying on the laws of physics allows for a faithful reproduction of the appearance of real objects, provided we have accurate measurements of an object attributes. The second characteristic, the applicability to a wide range of effects, comes from the flexibility of the path tracing concept and the generality of the Monte Carlo integration, which can handle integration domains of arbitrary dimensionality. The third characteristic, the convergence to the right result, is also inherited from the use of

Monte Carlo integration, which is both consistent and unbiased. Being consistent means that Monte Carlo methods converge to the right result as the number of samples grows to infinity. Being unbiased means that the expected value of the Monte Carlo estimation is the true integral value, in other terms, averaging multiple Monte Carlo estimations also converge to the right integral value as the number of estimations goes to infinity. While all these three characteristics are essential (being physically based, the wide applicability, and the convergence), the second one, that is that it can simulate a wide variety of effects in a unified framework, is arguably the most appealing because it leads to a relatively simple implementation.

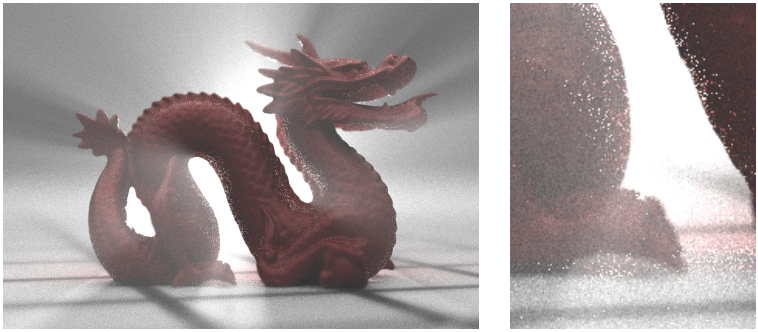


Figure 1.4: Same rendering as in Figure 1.3, but using 32 paths per pixel, instead of 32000. The rendering time is considerably reduced (42s instead of nearly 12 hours), but the output is now noticeably noisy.

The Curse of Dimensionality. Leaving aside the issue of convergence in pathological cases, which can be mitigated using more advanced path sampling algorithms, the main disadvantage of MCPT is that it is inherently a noisy process, which converges rather slowly to the right result. The noise in Monte Carlo integration correspond to the estimation error due to considering only a finite number of

samples, which leads to integral values that can be too high or too low (or pixels that are too dark or too bright in the rendering context). This noise can be reduced by increasing the number of samples and vanishes in the limit, as the number of samples grows to infinity, but the noise reduction is not linear with respect to the sampling rate. Actually, to reduce the magnitude of noise by a factor two, we need to increase the sampling rate by a factor of four. This leads to rapidly increasing sampling budgets for non trivial scenes. For instance, the rendering in Figure 1.3 was generated using 32000 paths per pixel, and took nearly 12 hours to compute on a powerful workstation with 12 CPU cores. As shown in Figure 1.4, using only 32 paths per pixel for the same scene yields a much more reasonable rendering time of 42 seconds, but the output is now noticeably noisy. This high computational cost, compared to the much more computationally efficient systems based on rasterization, is the main factor slowing down the adoption of MCPT. However, rasterization-based systems trade off computational cost for an implementation cost, since they rely on a collection of *ad hoc* hacks to approximate various light transport effects. The exponential growth of computational power is progressively shifting the outcome of this trade-off between computational and implementation cost, and leading the industry to move to MCPT for off-line rendering. This shift is perhaps best illustrated by Pixar's RenderMan renderer, with its seminal rasterization based rendering architecture, REYES [CCC87], which has now added support for ray tracing and used it in their movie Cars [CFLB06]. Yet, the computation cost of MCPT is still problematic for off-line rendering, and prohibitive for interactive or realtime rendering.

1.2 Problem Statement

In this thesis, we are concerned with tackling the high computational cost of MCPT, in order to mitigate the curse of dimensionality. Put simply, our goal is to reduce the amount of noise in a rendering given a fixed sample budget or vice-versa, improve image quality given a

fixed time budget.

It should be noted that we do not necessarily aim for an unbiased rendering, but rather a rendering that strikes an optimal balance between residual noise and bias. Ideally, the final output should be visually indistinguishable from an actual converged rendering for a human observer at a reasonable distance.

We wish to preserve the three key features of MCPT: (1) being physically based, (2) handling of a wide variety of light transport effects in a unified framework, and (3) estimation consistency at increasing sampling rate. Ideally, the solution should also apply to more sophisticated variants of MCPT such as photon mapping [Jen96], Metropolis light transport [VG97], etc. It should also be applicable with variance reduction methods (quasi MCPT [KK02a], multiple importance sampling [VG95b], photon mapping [Jen96], etc.) that are commonly used in the industry.

1.3 Image-Space Adaptive Rendering

We propose to reduce the noise in MCPT by employing an image-space adaptive rendering framework. Working in image-space decouples our framework from the dimensionality of the sampling space. This implicitly shields our approach from the curse of dimensionality, and ensures a constant overhead, independent of the scene complexity. Adaptive rendering refers to the two basic strategies to remove noise in a rendering: filtering and sampling. The adaptive sampling strategy simply consists in increasing the sampling rate, i.e. evaluating more light paths, in regions with a larger error (for instance pixel variance, though we could consider perceptual errors, or some other heuristic). The adaptive filtering strategy consists of estimating a pixel as a weighted average of its neighborhood, but using weights specially defined for this pixel and its neighborhood. This allows to have edge-aware filters that prevent blurring over edges. In practice, the adaptive filtering strategy is by far more computationally effective at reducing noise, and we use adaptive sampling to complement the

filtering strategy in regions where it is failing.

The proposed framework operates iteratively, using a greedy error minimization approach, and is illustrated in Figure 1.5. In each iteration, we sample the image plane, and then filter the result. We then proceed to estimate the error in the filtered rendering, which is used to define the adaptive sampling map (i.e. the per-pixel sampling density) that will be used in the sampling step of the next iteration. By adapting the sampling rate to the errors in the filtered rendering, we tightly couple the sampling and filtering strategies, and ensure that we only sample densely regions that cannot be properly reconstructed through filtering.

The main two design decisions in this framework are what filter to use, and what error metric should drive the adaptive sampling rate. In this thesis, we only consider the Mean Squared Error (MSE) as error metric, but experiment with filters of increasing complexity, and efficiency.

1.4 Summary of Contributions

We present a framework for adaptive sampling and reconstruction based on minimizing per-pixel MSE. In a greedy error minimization procedure, we iterate over two steps: applying filters specifically adapted to each individual pixel of the image and its neighborhood, and distributing additional samples in an attempt to maximally reduce MSE in each iteration. This framework tightly couples the sampling and filtering steps, ensuring that they complement each other's strength (namely, the low computation cost of filtering, and the unbiased reconstruction of dense sampling). In the sampling step, our main contribution is a strategy to distribute samples in each iteration, in order to minimize MSE on a per-pixel basis.

In this thesis, we develop three variants of this framework, each one using filtering approaches of increasing complexity and efficiency. The first variant leverages scale selection using a filterbank of isotropic Gaussian filters. Our main contribution in this approach is a robust

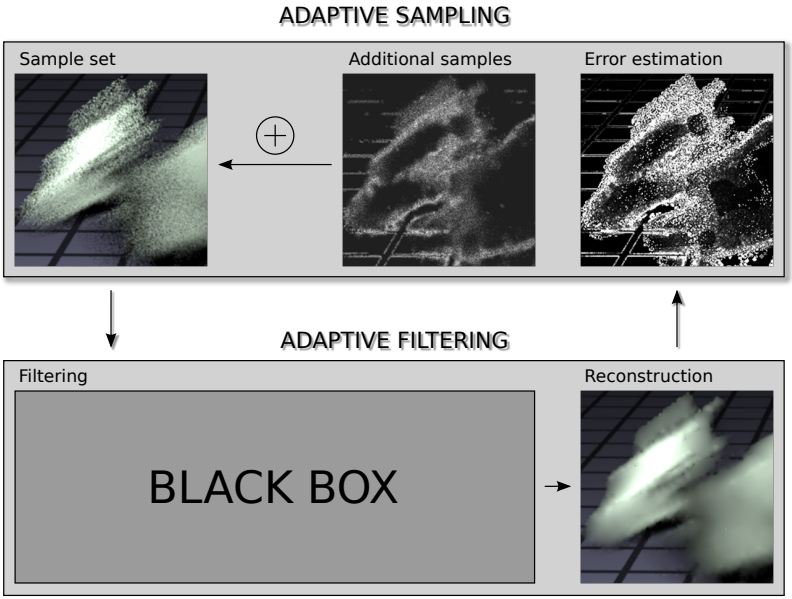


Figure 1.5: Proposed framework overview. Our framework tightly couples two core strategies: adaptive sampling and adaptive filtering. In each iteration, we sample the image plane, and then filter the result. We then estimate the error of the reconstruction, which we use to define where additional samples should be taken. By increasing the sampling rates in regions where adaptive filtering is less effective, we ensure that the two strategies complement each other.

method to select the filter scale and estimate MSE. The second approach uses a generalized Non-Local Means (NL-Means) filter, which is a robust non-linear filter that adapts itself to the signal. Our main contributions in this variant are: (1) a generalization of the NL-Means filter to handle data with non-uniform variance, (2) a dual-buffer strategy that allows us to obtain error estimates to drive adaptive sampling, and to extend our method to low-discrepancy sampling. The third

variant extends the second one, by integrating scene feature buffers (such as surface normals, position, direct visibility, etc.) to constrain the filtering. Our main contributions in this third approach are: (1) a pre-filtering strategy to handle noisy features, (2) a scale selection strategy to robustly combine pixel color and feature constraints.

All three variants of our framework were implemented and compared to other state of the art adaptive rendering methods on a variety of scenes demonstrating a wide range of materials and light transport effects.

1.5 Thesis Organization

The rest of the thesis is organized as follows:

- Chapter 2: brief overview of the rendering theory.
- Chapter 3: statistical tools used in our framework.
- Chapter 4: previous works, overview of the various adaptive rendering strategies that have been explored throughout the years.
- Chapter 5: previous works, overview of image space denoising techniques that have been used in adaptive rendering frameworks.
- Chapter 6: first implementation of our framework, leveraging nonlinear isotropic filtering [RKZ11].
- Chapter 7: second implementation of our framework, leveraging nonlinear anisotropic filtering [RKZ12].
- Chapter 8: third implementation of our framework, leveraging nonlinear anisotropic filtering and accounting for scene features to increase robustness [RMZ13].
- Chapter 9, conclusion and avenues for further research.

1.6 Related Problems

In this thesis, we develop a framework that aims to reduce the noise of MCPT through adaptive rendering. There are, however, many other techniques that have been proposed and which are often orthogonal (and therefore can be combined) to the one we propose. We will briefly present these techniques in this section. Lastly, we will present alternative rendering methods that have been developed over the years, including methods based on rasterization that have prevailed in the industry for many years.

1.6.1 Improving ray tracing based methods

Sampling Patterns. At its core, MCPT is a sampling problem, and the quality of sampling patterns is naturally a fundamental concern. Initial research in this topic was concern in minimizing artifacts. Stochastic sampling, and jittered sampling in particular [Coo86, DW85], was introduced to address aliasing artifacts of regular sampling patterns, by turning them into noise. Later work however, concentrated specifically on improving the properties of the sample sets, namely their discrepancy (which measures how evenly they are distributed in the sampling domain). This has lead to work on stratified sampling [Mit96], low-discrepancy sampling [KK02b], and Poisson-disk sampling patterns [Mit87, Mit91, ODJ04].

Importance Sampling. In MCPT, so paths may be hard to sample while having a large contribution to the overall shading. This would be the case of an sky map, where the sun would carry almost all the energy, while covering a very small solid angle. Uniformly sampling such functions lead to very large energy spikes that yield very noisy outputs. Additionally, if the paths carrying a large amount of energy are very rare, they could be missed altogether. A common solution for this type of problem is to use importance sampling, a technique that samples more densely regions that contribute more to the integral. This approach has been use to sample reflection func-

tions [LRR04], or environment maps [KK03, ARBJ03, ODJ04]. Methods have been designed to directly importance sample products of functions [CJAMJ05, CETC06, RCL+08] (for instance, the product of the reflection and lighting function) to great effect, but are generally restricted in their applicability. One pitfall of importance sampling is that it yields very bad results if the importance function is inappropriately chosen. Multiple importance sampling [VG95b] addresses this problem by combining multiple sampling techniques in a way that minimizes the contribution of inefficient techniques. All results presented in this thesis were obtained using the PBRT renderer, which employs multiple importance sampling.

Advanced Sampling Techniques. MCPT builds paths starting from the camera. This is based on the observation that only paths reaching the camera contribute to the final image. However, in cases where the light source is hard to reach through random sampling, it is more efficient to sample paths starting from the light source. Bidirectional path tracing [LW93, VG95a] combines the two approaches, by separately tracing paths from the camera and from the light sources, and connecting these. This results in a much more robust rendering technique that preserves the key characteristics of MCPT. Commercial renderers based on path tracing usually implement the bidirectional variant. Metropolis light transport [VG97] was proposed to handle scene where the light source is very difficult to reach, for instance when light goes through a small slit. In such cases, bidirectional path tracing is not sufficiently robust, since it is unlikely that randomly sampled camera and light paths could be connected together. Metropolis sampling solves this problem by mutating paths instead of randomly generating paths from scratch. As soon as a valid path is found, i.e. a path that connects the light to the camera, mutations allow to explore the sampling space in the vicinity of this valid path. Lastly, photon mapping [Jen96] proposes a idea similar to bidirectional path tracing, by tracing individual photons emitted by the lights throughout the scene. Tens of millions can be shot, and then stored in a data structure to be reused to render all pixels of this image, offsetting the cost

of shooting photons over many pixels. While we use mostly use MCPT in this thesis, our proposed framework is also compatible with bidirectional path tracing, Metropolis light transport, and photon mapping, and we present some results using photon mapping in Chapter 8.

1.6.2 Alternative Rendering Methods

Rasterization. Methods based on rasterization have been, by far, the most prevalent in the computer graphics industry. The core idea in rasterization is to draw primitives (usually triangles) to the screen. Rasterization algorithms can easily be parallelized on specialized hardware, and produce high quality renderings at a low cost, which makes them ideally suited for interactive or realtime applications, such as games where they are used almost exclusively. Pixar's REYES architecture, which dates back to the 1980s, also made use of rasterization, instead of ray tracing, for performance reasons. The main issue with rasterization based methods, is that they rely on a collection of tricks to approximate various effects (depth of field, soft shadows, etc.), and typically operate over multiple passes. Additionally, some effects, such as reflection of nearby objects on curved surfaces, cannot be easily approximated using rasterization, limiting its applicability.

Point Based Rendering. Global illumination is now commonly used to enhance the realism of renderings, but is problematic to render using rasterization methods. A simple solution would be to render the scene from the point of view of each pixel of the image, but this would obviously be prohibitive. Point based global illumination [Chr08] algorithms, where the scene is approximated using a hierarchical point cloud (where high level nodes of the hierarchy represent a coarser approximation of the geometry), offers an efficient solution. They operate by rasterizing appropriate nodes of the point cloud hierarchy, which greatly improves efficiency, while offering a reasonably accurate approximation of the global illumination solution. A similar idea was previously used to approximate all lights in a scene using a hierarchical cloud of lights [WFA+05, WABG06].

Chapter 2

Rendering Theory

Since this thesis is concerned with improving the efficiency of MCPT, we propose in the current chapter to introduce the problem MCPT was designed to solve, the rendering equation [Kaj86], as well as the source of MCPT's main artifact, the noise in rendered images.

The rendering equation was formulated in 1986 by James Kajiya in his seminal paper [Kaj86]. In this work, Kajiya not only gave a concise mathematical formulation of the rendering problem, but also presented a survey showing how previous work on ray tracing [Whi79], distributed ray tracing [CPC84], and radiosity [GTGB84], could be interpreted as approximate solutions to the rendering equation.

This chapter will present three different formulations of the rendering equation: the directional formulation in Section 2.1, the three-point formulation in Section 2.2, and the path integral formulation 2.3. Each new formulation offers an increasingly flexible interpretation of the rendering equation, and consequently offers an increasingly varied set of sampling schemes, with the path integral formulation being the most flexible.

2.1 The Rendering Equation

Consider a point $\mathbf{x}$ on $\mathcal{M}$, the union of all surfaces in the scene. The outgoing radiance at $\mathbf{x}$ in direction ω_o, $L_o(\mathbf{x}, \omega o)$ is the sum of the emitted radiance $L_e(\mathbf{x}, \omega o)$, and the scattered radiance $L_s(\mathbf{x}, \omega o)$,

$$L_o(\mathbf{x}, \omega o) = L_e(\mathbf{x}, \omega o) + L_s(\mathbf{x}, \omega o).$$

The emitted radiance L_e, that is, the radiance emitted by light sources, is given as part of the scene description, while the scattering L_s is computed by integrating all incoming light and applying the Bidirectional Reflectance Distribution Function (BRDF). The BRDF $f(\mathbf{x}, \omega_i \rightarrow \omega_o)$ is the ratio of reflected radiance along direction ω_o to the irradiance reaching $\mathbf{x}$ from direction ω_i, and the scattering is

$$L_s(\mathbf{x}, \omega o) = \int_{\mathcal{H}} L_i(\mathbf{x}, \omega_i) f(\mathbf{x}, \omega_i \rightarrow \omega_o) \cos\theta_i d\omega_i,$$

where $\mathcal{H}$ is the positive hemisphere of the surface at $\mathbf{x}$, L_i the incoming radiance, f the BRDF, and θ_i the angle between ω_i and the surface normal at $\mathbf{x}$. This leads to the directional form of the rendering equation,

$$L_o(\mathbf{x}, \omega o) = L_e(\mathbf{x}, \omega o) + \int_{\mathcal{H}} L_i(\mathbf{x}, \omega_i) f(\mathbf{x}, \omega_i \rightarrow \omega_o) \cos\theta_i d\omega_i. \quad (2.1)$$

Consider a function $\mathbf{x}_M$, such that $\mathbf{x}_M(\mathbf{x}, \omega)$ is the first intersection point with the surfaces of the scene $\mathcal{M}$, when tracing a ray from $\mathbf{x}$ in direction ω. Is is possible to rewrite the incoming radiance L_i from direction ω, as the outgoing radiance in direction $-\omega$ from the first intersection point $x_M(\mathbf{x}, \omega)$,

$$L_i(\mathbf{x}, \omega) = L_o(\mathbf{x}_M(\mathbf{x}, \omega), -\omega). \quad (2.2)$$

Equation 2.2 holds because radiance is constant along a ray in a vacuum. Combining Equations 2.1 and 2.2, we get the following form

for the rendering equation:

$$L_o(\mathbf{x}, \omega o) = L_e(\mathbf{x}, \omega o) + \int_{\mathcal{H}} L_o(\mathbf{x}_M(\mathbf{x}, \omega), -\omega) f(\mathbf{x}, \omega_i \rightarrow \omega_o) \cos\theta_i d\omega_i.$$

This formulation has the advantage of making the recursive nature of light transport more explicit. In particular, we see that the rendering equation is actually an integral equation, that is, an equation where one of the unknowns is an integral. Such equations generally do not have an analytical solution, and must be solved numerically. Kajiya therefore proposed the MCPT algorithm, based on a Neumann series decomposition, which solves the rendering equation by recursively sampling it: a ray is shot from the camera through the image plane until it intersects the scene, and then a ray is shot from that intersection in a random direction until it intersects the scene, and then another ray is shot from that latest intersection, and so on until a stopping criterion has been reached. The key advantage of MCPT, compared to previous approximations to the solution of the rendering equation, is that it considers all light transport effects, whereas previous methods only considered a subset of effects. For instance, Whitted's ray tracing algorithm could not handle indirect illumination, apart for specular reflection and refraction.

Lastly, we introduce the *measurement equation*, which we use to compute the value I_j of a pixel j. The value of a pixel is the integral of all incident radiance at the eye location $\mathbf{e}$, multiplied by the pixel importance function W^j that weights the contribution along each direction ω_i to pixel j,

$$I_j = \int_{\mathcal{H}} W^j(\omega_i) L(\mathbf{e}, \omega_i) \cos\theta_i d\omega_i.$$

2.2 The Three-Point Formulation

The directional formulation of the rendering equation leads to recursive algorithms, such as MCPT, but some paths are very difficult

to sample recursively, in particular in the presence of specular surfaces. The three-point formulation of the rendering equation is a step towards the path integral formulation, which can be used to derive more robust MC rendering algorithm beyond MCPT, such as bidirectional path tracing, where paths are traced concurrently from the camera and from the light sources.

Equation 2.1 formulates the rendering equation using a directional integral, the three-point form instead formulates it as a surface integral. In this formulation, instead of considering radiance from a point $\mathbf{x}$ in direction ω, we consider the radiance along a ray from $\mathbf{x}$ to another point $\mathbf{x}'$ of the scene,

$$L(\mathbf{x} \to \mathbf{x}') = L(\mathbf{x}, \omega),$$

where ω is the unit vector pointing from $\mathbf{x}$ to $\mathbf{x}'$. The BRDF can be similarly rewritten as

$$f(\mathbf{x} \to \mathbf{x}' \to \mathbf{x}'') = f(\mathbf{x}', \omega_i \to \omega_o),$$

where ω_i is the unit vector pointing from $\mathbf{x}'$ to $\mathbf{x}$, and ω_o is the unit vector pointing from $\mathbf{x}'$ to $\mathbf{x}''$.

We can now rewrite the rendering equation as an integral over the scene surface area $\mathcal{M}$,

$$\begin{aligned} L_o(\mathbf{x}' \to \mathbf{x}'') = L_e(\mathbf{x}' \to \mathbf{x}'') \\ + \int_{\mathcal{M}} L_o(\mathbf{x} \to \mathbf{x}') f(\mathbf{x} \to \mathbf{x}' \to \mathbf{x}'') G(\mathbf{x} \leftrightarrow \mathbf{x}') d\mathbf{x}, \end{aligned} \tag{2.3}$$

where $G(\mathbf{x} \leftrightarrow \mathbf{x}')$ is a geometry term accounting for the change of integration parameters. Figure 2.1 illustrates the setup.

We can easily derive the relation between a directional element $d\omega_i$ and a surface element $d\mathbf{x}$,

$$d\omega_i = \frac{\cos\phi}{||\mathbf{x} - \mathbf{x}'||^2} \, d\mathbf{x}. \tag{2.4}$$

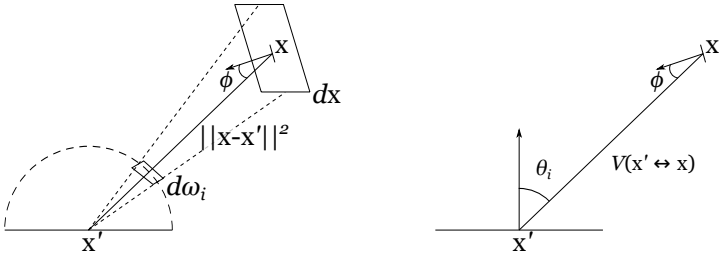


Figure 2.1: The left image illustrates the change of variable, going from directional integration to surface integration, as well as the relation of the area of elements on the hemisphere $d\omega_i$ and on the scene surfaces $d\mathbf{x}$. The right image illustrates the components of the geometry term (see Equation 2.4). The geometry term models the relation between the area of a surface element, $d\mathbf{x}$, and a directional element, $d\omega_i$. The visibility term, $V(\mathbf{x} \leftrightarrow \mathbf{x}')$, is 1 if $\mathbf{x}$ and $\mathbf{x}'$ are mutually visible and 0 otherwise.

The geometry term $G(\mathbf{x} \leftrightarrow \mathbf{x}')$ in Equation 2.3 combines Equation 2.4, with the visibility term $V(\mathbf{x} \leftrightarrow \mathbf{x}')$ seen in Figure 2.1, and the cosine attenuation term $\cos\theta_i$,

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

Finally, the change of variable is used to rewrite the measurement equation that gives the value I_j of pixel j as

$$I_j = \int_{\mathcal{M}} W^j(\mathbf{x} \to \mathbf{e}) L(\mathbf{x} \to \mathbf{e}) G(\mathbf{x} \to \mathbf{e}) d\mathbf{x}. \tag{2.5}$$

2.3 Path Integral Formulation

The path integral formulation of the rendering problem was proposed by Veach [Vea97]. It shifts the point of view from that of recursively sampling the integral equation, as is done in MCPT, to the idea of

sampling paths themselves. This formulation was used to design powerful extensions of MCPT, namely bidirectional path tracing [VG95a] and Metropolis light transport [VG97].

While the results presented in this thesis make use of MCPT, and not bidirectional path tracing or Metropolis light transport, we still introduce the path integral formulation, since it offers a very simple interpretation of the light transport problem. In practice, our framework could be applied to bidirectional path tracing without any change. However, Metropolis light transport could not be handled directly, since it is substantially different from MCPT, and would therefore require a specifically designed adaptive framework, which could be an interesting avenue for future work.

A path is defined as a set of vertices that starts at a light source and, going through a pixel, ends at the camera. The set of all paths of length k is denoted Ω_k, and each path has the form

$$\bar{x} = \mathbf{x}_0 \mathbf{x}_1 \cdots \mathbf{x}_k,$$

with $1 \leq k \leq \infty$. Each point $\mathbf{x}_i$ is a path vertex on a surface of the scene (which could be a light source, the camera, or any other object of the scene). Vertex $\mathbf{x}_0$ is on a light source, while vertex $\mathbf{x}_k$ is the camera. The path space Ω, which is the set of all paths, is the union of all paths of all lengths,

$$\Omega = \bigcup_{k=1}^{\infty} \Omega_k.$$

Noting $f_j(\bar{x})$ the contribution of a path $\bar{x}$ to a pixel j, we can express the value of a pixel j as

$$I_j = \int_{\Omega} f_j(\bar{x}) d\bar{x}, \tag{2.6}$$

where $f_j(\bar{x})$ is the contribution of the path $\bar{x}$ to the value I_j of pixel j. We now write the path integral formulation, based on the measurement equation of the three-point formulation (Equation 2.5), and

recursively expand the radiance term:

$$I_j = \sum_{k=1}^{\infty} \int_{\mathcal{M}^k} L_e(\mathbf{x}_0 \to \mathbf{x}_1) G(\mathbf{x}_0 \leftrightarrow \mathbf{x}_1) \prod_{i=1}^{k-1} f(\mathbf{x}_{i-1} \to \mathbf{x}_i \to \mathbf{x}_{i+1}) \cdot G(\mathbf{x}_i \leftrightarrow \mathbf{x}_{i+1}) W^j(\mathbf{x}_{k-1} \to \mathbf{x}_k) d\mathbf{x}_0 \cdots d\mathbf{x}_{k_1}. \tag{2.7}$$

In Equation 2.6, the integration was done over Ω, the set of all paths of all lengths. In Equation 2.7, this translates to the sum $\sum_{k=1}^{\infty}$, which iterates over all path lengths, and the integral $\int_{\mathcal{M}^k}$, which corresponds to the set of all paths of length k. The integration domain is $\mathcal{M}^k$, since each vertex of the path could be placed at any point of the scene surfaces $\mathcal{M}$.

The path integral in Equation 2.7 cannot be solved analytically for non-trivial scenes, and must therefore be solved numerically. A common approach is Monte Carlo (MC) integration, which we present in Section 3.6, where paths are randomly sampled. It is precisely the random sampling of paths that introduces the rendering noise that we aim to reduce in this thesis.

Chapter 3

Statistical Tools

The adaptive rendering framework developed in this thesis is empirical in nature. We start by randomly sampling the path integral, and analyze the resulting set of samples, once the initial sampling is done. The output of this analysis is used to drive the subsequent adaptive filtering and sampling steps. This chapter presents the statistical tools that are used to perform the analysis of the samples.

3.1 Random Variable

A *random variable* models the output of an uncertain process. The value of the random variable is therefore not fixed, but can randomly take any of a set of potential output values, called the sample space. For instance, a random variable representing the output of rolling a dice with six faces can take one of six values.

In MCPT, the sampling space corresponds to the set of paths that light could follow through the scene, and each light path $\bar{x}$ is a realization of the high dimensional random variable $\bar{X}$ resulting from a random process that samples the high dimensional parameters of the path.

3.2 Probability Density Function

The probability density function (PDF) of a random variable indicates the relative likelihood of a given realization of the random variable, that is, the relative likelihood of observing a given value. A PDF is positive (or equal to zero) everywhere and integrates to one: $\int p(x) = 1$, and $p(x) >= 0, \forall x$.

The probability of a random variable to take a value in a given range is obtained by integrating its PDF over that range,

$$\Pr[a \leq X \leq b] = \int_a^b p(x)dx. \tag{3.1}$$

One interesting outcome of Equation 3.1 is that the probability of a specific realization of a continuous random variable is zero, that is, $\Pr[X = a] = 0$.

Equation 3.1 illustrates the PDF on a one dimensional sampling space. However, in rendering, paths are defined in a high dimensional sampling space. The dimensionality of the rendering sampling space depends on the light transport effects that are simulated, but for instance a scene featuring depth of field and motion blur would have a five dimensional sampling space (two dimensions for sampling the image plane, two dimensions for sampling the lens aperture, and one dimension for sampling time).

3.3 Expected Value of a Random Variable

The expected value of a random variable X is the integral of all the values it can take over the sampling space Ω, weighted by its PDF,

$$\mathrm{E}[X] = \int_\Omega x\, p(x)dx.$$

In the general case, one cannot directly evaluate the expected value of a random variable, since it may not have a closed form solution.

The expected value of a measurable function f of X is computed as

$$\mathrm{E}\left[f(X)\right] = \int_{\Omega} f(x)\, p(x) dx.$$

In rendering, the expected value of a function of a random variable is of particular interest, since each pixel value is actually the expected value of the path contribution function defined in Section 2.3, $f_j(\bar{X})$, which is a function of the random variable that represents the stochastic sampling process.

3.4 Variance of a Random Variance

Variance is a measure of the spread of a random variable, or how far it can deviate from the mean μ, and is computed as

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

The mean μ is the expected value of the random variable, and the variance can be reformulated as

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

3.5 Estimator Bias

An estimator $\hat{\theta}$ is a statistic that is used to estimate a parameter θ based on sampled data. The estimator $\hat{\theta}$ is to be distinguished from its output, the estimate. The bias of a an estimator is a measure of any systematic error that would prevent the estimator from converging to the parameter being estimated, even using an infinitely large data set. It is measured as the difference between the expected value of the estimator $\mathrm{E}[\hat{\theta}]$ and the parameter θ:

$$\mathrm{Bias}\left[\hat{\theta}\right] = \mathrm{E}[\hat{\theta}] - \theta.$$

An estimator is said to be unbiased if $\text{Bias}\left[\hat{\theta}\right] = 0$.

Let us now look at the bias of the mean of multiple independent estimates,

$$\text{Bias}\left[\frac{1}{n}\sum_{i=1}^{n}\hat{\theta}_i\right] = \frac{1}{n}\sum_{i=1}^{n}\text{Bias}\left[\hat{\theta}_i\right] = \text{Bias}\left[\hat{\theta}\right], \tag{3.2}$$

and its variance,

$$\text{Var}\left[\frac{1}{n}\sum_{i=1}^{n}\hat{\theta}_i\right] = \frac{1}{n^2}\sum_{i=1}^{n}\text{Var}\left[\hat{\theta}_i\right] = \frac{1}{n}\text{Var}\left[\hat{\theta}\right]. \tag{3.3}$$

From Equations 3.2 and 3.3, we see that, if an estimator is unbiased, averaging a growing number of independent estimates converges to the right result. This result is useful for unbiased rendering algorithms such as MCPT, since it allows easy parallelization of the rendering process by computing multiple independent renderings on a cluster of machines, and averaging the results to get a single rendering with lower variance.

3.6 Monte Carlo Integration

We now very briefly present the concept of Monte Carlo integration, which is central to MCPT. Monte Carlo integration is a numerical approach to evaluating integrals by randomly sampling the integration domain. In its simplest implementation, one draws samples of a random variable with uniform probability density, and averages their contributions. We call samples drawn with a uniform probability density *uniform samples*. A one-dimensional integral $\int_a^b f(x)dx$ could be estimated using N uniform samples with the following Monte Carlo estimator:

$$F_N = \frac{b-a}{N}\sum_{i=1}^{N} f(X_i),$$

where $f(X_i)$ is the contribution of sample X_i. The X_i term corresponds to independent and identically distributed random variables used to model the process of drawing N samples of X. It is easy to verify that the expected value of this estimator, $E[F_N]$, is an unbiased estimator of our one-dimensional integral. Since samples are drawn uniformly, the PDF is a constant, $p = 1/(b-a)$. By applying the equation for the expected value, we have

$$\begin{aligned}
\mathrm{E}[F_N] &= \mathrm{E}\left[\frac{b-a}{N}\sum_{i=1}^{N} f(X_i)\right] \\
&= \frac{b-a}{N}\sum_{i=1}^{N} \mathrm{E}\left[f(X_i)\right] \\
&= \frac{b-a}{N}\sum_{i=1}^{N} \int_b^a f(x)p(x)dx \\
&= \frac{1}{N}\sum_{i=1}^{N} \int_b^a f(x)dx \\
&= \int_b^a f(x)dx.
\end{aligned}$$

In practice, samples do not necessarily have to be generated using a uniform PDF, but could use arbitrary PDFs, as long as the density is non-zero and positive. This technique is called importance sampling, and the Monte Carlo estimator becomes

$$F_N = \frac{1}{N}\sum_{i=1}^{N} \frac{f(X_i)}{p(X_i)},$$

where $p(X_i)$ is the PDF used to draw the samples X_i. With the same reasoning as before, we can show that the expected value of the new estimator is the integral we want to evaluate. Importance sampling can significantly reduce the estimator variance if the sampling PDF (the importance function) is well chosen, that is if it has a similar

shape as the function we are integrating. The ideal importance function is proportional to the function itself, so that $p(x) = cf(x)$, with c a constant such that $\int p(x)dx = 1$, in which case the estimator has no variance since $f(X_i)/p(X_i) = c$, and c is actually the integral value. Note that importance sampling can actually increase the variance of the estimator (and sometimes greatly) if the importance function is badly chosen.

Let us go back to the problem of rendering, and in particular that of estimating the path integral formulation of light transport (Equation 2.6), which we reproduce here:

$$I_j = \int_\Omega f_j(\bar{x})d\bar{x},$$

where $\bar{x}$ is a path, $f_j(\bar{x})$ is the contribution of this path to pixel j, and Ω is the set of all paths of all lengths. We see that this integral can be directly evaluated using Monte Carlo integration,

$$I_j \approx \frac{1}{N}\sum_{i=1}^{N}\frac{f(\bar{X}_i)}{p(\bar{X}_i)}, \tag{3.4}$$

where $\bar{X}_i$ is a random sample path, and $p(\bar{X}_i)$ the probability density of the sample.

In rendering, each random sample path is a realization of a high dimensional random variable, and that random variable is the set of all the simulation parameters. The simulation parameters include the screen position, the lens position (to simulate depth of field), the time (to simulate motion blur), the light position, the polar coordinates of the direction taken at each bounce in the scene, etc. Given the high dimensionality of the sampling space, naive sampling would lead to highly noisy renderings, and multiple variance reduction techniques are commonly used. These variance reduction techniques can be roughly split in two categories: advanced sampling techniques and improved sample distributions. Note that most of these variance reduction techniques are orthogonal to our framework.

As examples of advanced sampling technique that are commonly used in rendering for Monte Carlo integration, we have importance

sampling [LRR04] or bidirectional path tracing [LW93]. However, we will not discuss such techniques since they are orthogonal to the the framework developed in this thesis.

Improved sample distributions aim at better covering the sampling domain by preventing clumps of samples. Two common techniques to obtain improved sample distributions are stratified sampling and quasi-random sampling. In stratified sampling, the sampling domain is partitioned in non-overlapping strata, and a single sample is put in each stratum. While this technique does not guarantee a minimum distance between nearby samples, it does lead to a better overall coverage of the sampling domain, and is also guaranteed to give lower or equal variance to standard random sampling. In quasi-random sampling, samples are taken from a predefined low-discrepancy sequence [Nie92], designed to maximize the distance between nearby samples. Unlike stratified sampling, quasi-random sampling is not guaranteed to lower the variance of the Monte Carlo estimator [Owe97], though it does most of the time. However, quasi-random sampling generally performs better than stratified sampling in practice, and sometimes significantly so. We discuss in Section 3.8, how to compute the variance of a random variable when using stratified and quasi-random sampling.

An important characteristic of Monte Carlo integration is that it is unbiased, which makes MCPT an unbiased rendering algorithm. This unbiasedness has two useful outcomes: 1) MCPT will converge to the right result given infinitely many samples, and 2) independent MCPT renderings could be averaged together to yields another rendering with lower variance, allowing trivial parallelization of the rendering process. In our framework, we do not preserve the unbiasedness of MCPT, but we do preserve its consistency. In other words, our framework will converge to the right result given infinitely samples, but averaging independent renderings obtained with our framework would not converge to the right result. The bias in our framework is due to the filtering occurring during the adaptive reconstruction, which is fundamentally biased as we show in the introduction of Chapter 5.

3.7 A Pixel Mean as a Gaussian Variable

A Gaussian variable is a random variable that has a Gaussian distribution. It is bell-shaped, and defined solely by its mean and variance. We propose to model each pixel value in an MCPT rendering using this Gaussian model.

While this may appear to be a gross approximation given the wide variety of sample distributions that can be observed in renderings, it is actually appropriate since we're not considering the sample distribution, but rather the distribution of the sample mean. Indeed, as can be seen in Equation 3.4, the value of each pixel is the mean of the normalized path contributions $f(\bar{X}_i)/p(\bar{X}_i)$ to that pixel. This fact validates the Gaussian model, since, according to the central limit theorem, the distribution of the mean of sufficiently many independent and identically distributed random variables is approximately Gaussian distributed, regardless of the actual distributions of the random variables.

3.8 Estimating Variance

In rendering, the variance of the random variable (the pixel mean in our case) is not known in advance, since it depends on the specificities of the scene being rendered, and the variance has to be estimated from the set of samples. In this section, we start by showing how to compute the sample variance, assuming we have independent and identically distributed samples. We then describe how we can estimate the variance of correlated sample sets obtained using stratified sampling or low discrepancy sampling. Stratified sampling and low discrepancy sampling are two important cases, since they are very commonly used in the computer graphics industry. Their use is motivated by the fact that they reduce the variance of Monte Carlo estimators, by ensuring that samples are more homogeneously distributed in the sampling space.

3.8.1 Random Samples

We are interested in computing the variance of a random variable using a set of sampled data, that is, a set of realizations of that random variable. We can compute the variance of our sampled data set as:

$$S^2 = \frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})^2.$$

We will now relate this sample variance to the variance of the random variable. First, we observe that the sample variance is itself a random variable, and we can therefore compute its expected value:

$$\begin{aligned}
S^2 &= \mathrm{E}\left[\frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})^2\right] = \mathrm{E}\left[\frac{1}{n}\sum_{i=1}^{n}\left((X_i - \mu) - (\bar{X} - \mu)\right)^2\right] \\
&= \mathrm{E}\left[\frac{1}{n}\sum_{i=1}^{n}\left((X_i - \mu)^2 - 2(X_i - \mu)(\bar{X} - \mu) + (\bar{X} - \mu)^2\right)\right] \\
&= \mathrm{E}\left[\frac{1}{n}\sum_{i=1}^{n}(X_i - \mu)^2\right] - 2\,\mathrm{E}\left[(\bar{X} - \mu)^2\right] + \mathrm{E}\left[(\bar{X} - \mu)^2\right] \\
&= \mathrm{E}\left[\frac{1}{n}\sum_{i=1}^{n}(X_i - \mu)^2\right] - \mathrm{E}\left[(\bar{X} - \mu)^2\right] = \mathrm{Var}\left[X\right] - \mathrm{Var}\left[\bar{X}\right] \\
&= \mathrm{Var}\left[X\right] - \frac{1}{n}\mathrm{Var}\left[X\right] = \frac{n-1}{n}\mathrm{Var}\left[X\right].
\end{aligned}$$

As we see, the sample variance is actually a biased estimator the variance of the random variable, which it underestimates by a factor $(n-1)/n$. We can easily correct this bias, which yields the corrected sample variance, or unbiased sample variance:

$$s^2 = \frac{n}{n-1}\,S^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2.$$

3.8.2 Stratified Sampling

Let us now consider the problem of estimating the variance of an estimator using stratified sampling. Stratification breaks down the overall variance in two: the variance within each stratum, and the variance between strata,

$$\mathrm{Var}_{total} = \mathrm{Var}_{within} + \mathrm{Var}_{between}.$$

Each stratum is evaluated independently, and we weight its contribution according to its size. For instance, to compute the mean μ of a random variable X, using H strata, we use the following unbiased estimator:

$$\begin{aligned}\bar{X}_{str} &= W_1\bar{X}_1 + W_2\bar{X}_2 + \cdots + W_H\bar{X}_H \\ &= \sum_{h=1}^{H} W_h\bar{X}_h.\end{aligned}$$

We can then directly compute the variance of this stratified estimator,

$$\begin{aligned}\mathrm{Var}\left[\bar{X}_{str}\right] &= \mathrm{Var}\left[\sum_{h=1}^{H} W_h\bar{X}_h\right] \\ &= \sum_{h=1}^{H} \mathrm{Var}\left[W_h\bar{X}_h\right] \\ &= \sum_{h=1}^{H} W_h^2\mathrm{Var}\left[\bar{X}_h\right].\end{aligned}$$

This derivation follows from the fact that the strata are sampled independently, which ensures that there is no covariance between the strata.

3.8.3 Quasi-Random Sampling

Quasi-random sampling leverages low-discrepancy sequences [Nie92], which attempt to maximize the distance between any pair of samples,

in order to maximize the coverage of the sampling space. This implicitly reduces the likelihood of having clump of samples, and therefore the likelihood of under-sampled regions.

In rendering, Keller and collaborators were early proponents of using quasi-random sampling, and explored a variety of applications. In particular, Kollig and Keller investigated quasi-Monte Carlo rendering, and demonstrated it to be potentially significantly more effective at reducing variance than stratified sampling [KK02a], in particular for relatively low-dimensional sampling problems. Today, quasi-Monte Carlo rendering is very widely used in the industry.

By construction, all samples in a quasi-random sequence are correlated (since their position is implicitly defined by that of the other samples of the sequence), and it is therefore not possible to rely on the traditional sample variance. We propose to use a dual-buffer approach to solve this, where two renderings are generated using independent low-discrepancy sequences. The per-pixel variance between the two renderings is an unbiased, though very noisy, estimator of the sample mean variance of each pixel. We describe our approach in more detail in Section 7.1.1, and an improved implementation building on the same idea in Section 8.2.

3.9 Mean Squared Error

The Mean Squared Error (MSE) of an estimator $\hat{\theta}$ is the metric used in our framework to evaluate the quality of a reconstruction. While the MSE is not a perfectly reliable predictor of perceptual errors [WBSS04b], a recent survey of image quality metrics found it to be similarly useful to other metrics [CHM+12].

The MSE of an estimator is directly related to its variance and squared bias:

$$\mathrm{MSE}\left[\hat{\theta}\right] = \mathrm{Var}\left[\hat{\theta}\right]^2 + \mathrm{Bias}\left[\hat{\theta}\right]^2. \tag{3.5}$$

This follows from the definition of the MSE:

$$\begin{aligned}
\mathrm{MSE}\left[\hat{\theta}\right] &= \mathrm{E}[(\hat{\theta}-\theta)^2] = \mathrm{E}[((\hat{\theta}-\mathrm{E}[\hat{\theta}])+(\mathrm{E}[\hat{\theta}]-\theta))^2] \\
&= \mathrm{E}[(\hat{\theta}-\mathrm{E}[\hat{\theta}])^2] + 2\mathrm{E}[(\hat{\theta}-\mathrm{E}[\hat{\theta}])(\mathrm{E}[\hat{\theta}]-\theta)] + \mathrm{E}[(\mathrm{E}[\hat{\theta}]-\theta)^2] \\
&= \mathrm{Var}\left[\hat{\theta}\right] + 2\mathrm{E}[\hat{\theta}\mathrm{E}[\hat{\theta}] - \mathrm{E}[\hat{\theta}]^2 - \hat{\theta}\theta + \mathrm{E}[\hat{\theta}]\theta] + \mathrm{Bias}\left[\hat{\theta}\right]^2 \\
&= \mathrm{Var}\left[\hat{\theta}\right] + \mathrm{Bias}\left[\hat{\theta}\right]^2
\end{aligned}$$

An interesting outcome of Equation 3.5 is that we can evaluate the accuracy of an estimator by independently studying its variance and its squared bias. This is of particular importance in the context of adaptive rendering, since we propose to reduce the variance of MCPT renderings using adaptive filtering techniques. A filter is actually an estimator, and we can therefore select the best of multiple filters if we have access to their variance and bias. This will be the key of the first implementation of our adaptive rendering framework, presented in Chapter 6.

3.9.1 Filter Variance Reduction and Bias

As we just saw, we can evaluate the accuracy of a filter, as an estimator, according to its variance and squared bias. Because of this, we would be interested in defining the variance of a filter output, based on the variance of its input. This amounts to measuring the filter variance reduction potential. Later on, we will look at the bias of the filter.

Let us start with the definition of an image space filter, which estimates each pixel value as a weighted average of its neighborhood. Consider the estimation at pixel p,

$$\hat{f}(p) = \sum_{q \in N(p)} w(p,q) f(q), \tag{3.6}$$

where q is a pixel in the neighborhood of p, $N(p)$, and $w(p,q)$ is the weight of this neighbors. The weights are constrained to the range $[0,1]$, $0 \leq w(p,q) \leq 1$, and should sum up to 1, $\sum_{q \in N(p)} w(p,q) = 1$.

If the weights of the filter are fixed, then the variance of the filtered image is

$$\operatorname{Var}\left[\hat{f}(p)\right] = \operatorname{Var}\left[\sum_{q\in N(p)} w(p,q)f(q)\right] = \sum_{q\in N(p)} w(p,q)^2 \operatorname{Var}\left[f(q)\right]. \tag{3.7}$$

Assuming a uniform variance, that is, $\operatorname{Var}\left[f(q)\right] = c$, the variance reduction potential of the filter is given by $\sum_{q\in N(p)} w(p,q)^2$. In practice, we are only guaranteed to lower the variance if the weights are constrained to the range $[0,1]$, and if none of the neighbor pixels has a larger variance than the central pixel.

The bias of a filter is linked to the difference between a pixel and its neighborhood. Based on Equation 3.6, we can get the bias of a filter,

$$\begin{aligned}
\operatorname{Bias}\left[\hat{f}(p)\right] &= \mathrm{E}[\hat{f}(p)] - \mathrm{E}[f(p)] \\
&= \mathrm{E}\left[\sum_{q\in N(p)} w(p,q)f(q)\right] - \mathrm{E}[f(p)] \\
&= \sum_{q\in N(p)} w(p,q)(\mathrm{E}\left[f(q)\right] - \mathrm{E}[f(p)]) \\
&= \sum_{q\in N(p)} w(p,q)\operatorname{Bias}\left[f(q)\right].
\end{aligned}$$

In practice, the bias of the filter is the weighted average of the bias of each neighbor used to estimate the center pixel. Given that bias can be either positive or negative, it is quite possible that the overall bias vanishes. For instance, an isotropic filter would not introduce any bias when filtering a smooth gradient, since radially symmetric neighbors would cancel out each other's bias. In practice, we do not actually know the bias induced by each neighbor contribution, and we have to resort to other means of computing the filter bias. In Section 6.2.2, we show how to estimate the bias when using isotropic Gaussian filters.

As an example, let us now consider the simple case of a box filter covering a $2r+1 \times 2r+1$ square neighborhood, where r is the radius of the filter. The weights of the filter are constant and sum up to 1:

$$w(p,q) = 1/(2r+1)^2.$$

According to Equation 3.7, and assuming a uniform variance, the variance reduction of the box filter is: $\sum_{q \in N(p)} w(p,q)^2 = 1/(2r+1)^2$. For instance, a 3×3 box filter would reduce the variance by a factor of 9, a 5×5 box filter by a factor 25, etc. To put things in perspective, reducing variance by a factor of 9 through sampling would require increasing the sampling rate by a factor of 9, a considerable computational effort. While it is tempting to use a larger filter for the improved variance reduction, it will also lead to an increased bias. In rendering, given the high dynamic range of the signal, the bias introduced by some neighbors can be arbitrarily large.

We have discussed the case of image space filters with fixed weights, but left aside the case of data-adaptive filters. A data-adaptive filter is a filter whose weights depend on the content of some guide data (which could be the data to be filtered itself), and usually aims at better preserving edges in the signal, which in turn reduces the bias of the filter. Unfortunately, our analysis of the variance of a filter does not hold for data-adaptive filters that adapt to noisy input data. Indeed, filter weights based on noisy data have variance, and are therefore random variables whose variance will affect the filtered output. If these noisy weights are applied to the noisy data they were derived from, then this builds dependence between the random variables (that is, a neighbor weight and its value), which further complexifies the analysis of the filtered output variance. Therefore, to compute the variance of data-adaptive filters, we instead propose to use a two buffer approach, where the filter variance is estimated using the variance between two filtered buffers, and present it in Section 7.1.2.

3.9.2 Stein's Unbiased Risk Estimator

In the previous section, we discussed means of independently estimating the bias and variance of an estimator, which are then summed up to get its MSE. Stein's Unbiased Risk Estimator (SURE) instead directly estimates the MSE [Ste81]. The key aspect of SURE is that it is an *unbiased* estimator of the MSE, and one can therefore minimize SURE as a proxy for minimizing MSE.

Given y, an observation of x with a normal distribution $N(x, \sigma_y^2)$, and F a weakly differentiable estimator, SURE is defined as:

$$\mathrm{SURE}(F(y)) = ||F(y) - y||^2 + 2\sigma_y^2 \frac{dF(y)}{dy} - \sigma_y^2.$$

To get a better intuition of SURE, let us consider two cases. First, we consider the case of the identity estimator, $I(y) = y$, with $\frac{dI(y)}{dy} = 1$. Applying SURE to I gives:

$$\begin{aligned}\mathrm{SURE}(I(y)) &= ||I(y) - y||^2 + 2\sigma_y^2 \frac{dI(y)}{dy} - \sigma_y^2 \\ &= 0 + 2\sigma_y^2 - \sigma_y^2 = \sigma_y^2.\end{aligned}$$

In other terms, the MSE of the identity estimator is simply the variance of the input itself. Now, let us consider the case of an estimator, G whose output does not depend on y at all, such that $\frac{dG(y)}{dy} = 0$. Applying SURE to this estimator gives:

$$\begin{aligned}\mathrm{SURE}(G(y)) &= ||G(y) - y||^2 + 2\sigma_y^2 \frac{dG(y)}{dy} - \sigma_y^2 \\ &= ||G(y) - y||^2 - \sigma_y^2.\end{aligned}$$

In other terms, the MSE of G is given by measuring the squared difference between the estimate $G(y)$ and the noisy input y, but subtracting the variance of y, σ_y^2, which is implicitly measured by $||G(y) - y||^2$.

We use SURE in the third implementation of our adaptive rendering framework, presented in Chapter 8, to select between various candidate filters on a per-pixel basis.

Chapter 4

Adaptive rendering algorithms

In practice, "rendering" consists in solving the rendering equation, introduced in Chapter 2. Since solving this equation involves evaluating high dimensional integrals with no analytical solution, researchers have turned to numerical integration, which lead to MCPT, an algorithm based on Monte Carlo methods. The two core parts of MCPT are sampling and reconstruction. Sampling consists of generating paths that span the whole integration domain, while reconstruction consists in computing the image value at a given position as a weighted average of the contribution of nearby sampled paths, where each sample weight is given by the pixel filter kernel.

The standard approach to MCPT is to perform uniform sampling and reconstruction, that is, paths are generated with a uniform density over the image plane, and a fixed reconstruction filter is used over the whole image. Typical choices of pixel filters are the box or Gaussian filters, or Mitchell's filter [MN88]. While this standard approach to MCPT could be considered to have solved the rendering problem (at least for non pathological scenes), this is only true for infinite sampling rates. In practice, sampling rates are finite and therefore

MCPT renderings are plagued with high frequency noise that can be visually distracting. The adaptive rendering algorithms that will be presented in this chapter all aim at reducing the level of noise in MCPT renderings, and do so using two main strategies: adaptive sampling and adaptive reconstruction. These strategies can be employed on their own, but are often combined.

The first strategy for reducing rendering noise, adaptive sampling, consists in increasing the sampling rate to the point where noise artifacts are below a predefined error threshold. Since noise is not uniform across the image plane (neither from a variance, nor from a perceptual point of view), this naturally leads to non uniform sampling rates, where some regions are sampled more densely than others, hence the term adaptive sampling. Alternatively, instead of distributing samples until convergence, we can distribute a fixed sample budget proportionally to the locally estimated error, which ensures that the overall error is minimized. The key attribute and main differentiation point between adaptive sampling methods is the error metric they use. The seminal work of Don Mitchell [Mit87] considered a simple contrast metric, inspired by studies on human perception. Since then, other works have considered more advanced perceptual models [BM98, RPG99]. Ward et al.'s irradiance caching algorithm [WRC88] used the magnitude of gradients to decide where to put more samples. Many recent methods are based on frequency analysis theory, leveraging Nyquist's sampling theorem [DHS+05], while recent empirical methods often make use of relative MSE [RKZ11, LWC12].

The second strategy for reducing rendering noise, adaptive reconstruction, consists in using a reconstruction scheme adapted to the local characteristics of each reconstruction location, with the goal of generating the optimal reconstruction given a finite set of samples. The relevant local characteristics are the level of noise and the frequencies of the signal. For instance, noisy regions with a constant intensity would be reconstructed with a filter with a wide support to minimize variance, while regions with high frequencies would use a narrow support to prevent bias. Filter-based reconstruction, where a pixel value

is directly computed as a weighted average of nearby samples, is just one approach among many. Alternative means of reconstruction have also be considered in adaptive rendering, such as splatting [RW94], transform domain denoising [ODR09], Poisson solvers [LKL+13], local regression [BEM11], or anisotropic diffusion [McC99].

Surprisingly, the bulk of previous work in adaptive rendering has been done in the last ten years, probably motivated by the rise in popularity of MCPT and its derivatives. The recent surge of adaptive rendering methods have arguably been sparked by Durand et al.'s frequency analysis of light transport [DHS+05], and Hachisuka et al.'s multidimensional sampling and reconstruction method [HJW+08]. These two papers are also representative of two fundamentally different approaches to adaptive rendering. The former would be an *a priori* method, while the later would be an *a posteriori* method.

The main point differentiating *a priori* and *a posteriori* methods is that *a priori* methods are analytical, and their behavior is governed by an underlying model of the rendering problem, whereas *a posteriori* methods are empirical, and their behavior is governed by the sampled data gathered during the rendering process.

The rest of this chapter will attempt to give a broad overview of the various adaptive rendering methods that have been proposed throughout the years, based on our *a priori* vs. *a posteriori* classification, as illustrated in Figure 4.1. Section 4.1 will cover *a priori* methods, and the exposition will be split over Sections 4.1.1 to 4.1.3 according to the type of analysis used (gradient, frequency, or light field structure analysis). Section 4.2 will cover *a posteriori* methods, and we will start by describing the various error metrics used in Section 4.2.1, and then the methods themselves in Section 4.2.2, classified according to the strategies they use (adaptive sampling, adaptive reconstruction, or both). Our exposition will be at a relatively high level, but Chapter 5 will present in more details the main image space denoising techniques used in adaptive rendering that are relevant to our own work.

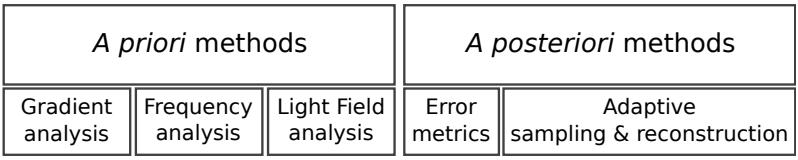


Figure 4.1: Overview of the structure of this chapter.

4.1 *A Priori* Methods

There is a wide array of information that *a priori* methods can leverage. For instance, the properties of the reflection of diffuse materials has been extensively exploited in various methods. But other types of information can be leveraged, such as the scene geometry, the camera optics, the motion vectors, light field structure, and others.

By exploiting known aspects of specific light transport effects, *a priori* methods can propose some strikingly efficient algorithms. This is however a double edged sword, and these methods are in practice constrained to a subset of light transport effects, in order to ensure the analysis remains tractable. Consequently, a wide array of methods have been proposed to handle an equally wide array of light transport effects. Methods have been designed to handle indirect illumination, depth of field, motion blur, soft shadows, sometimes restricted to diffuse scenes, etc. While most methods handle only a single effect at a time, some more recent ones have more general solutions that can handle combinations of effects.

We will now present the main *a priori* methods, classified according to the type of analysis they rely upon. We distinguish three main types: gradient analysis, frequency analysis, and light field structure analysis.

4.1.1 Gradient Analysis

Let us first define the gradient of a function, and then see how it can be used to perform the two core tasks of adaptive rendering, that is,

adaptive sampling and adaptive reconstruction.

The gradient of a function f is a vector whose components are the partial derivatives of f. For instance, if f is defined in the Cartesian system, its gradient is defined as

$$\nabla f = \frac{\delta f}{\delta x}\mathbf{i} + \frac{\delta f}{\delta y}\mathbf{j} + \frac{\delta f}{\delta z}\mathbf{k},$$

where **i**, **j**, and **k** are the standard unit vectors.

If we consider the problem of sampling, a small gradient indicates that the function is locally constant, whereas a large gradient indicates that the function value is changing rapidly. This suggests a straightforward sampling scheme where regions with a large gradient are sampled more densely than regions with a small gradient. This will naturally lead to sparse sampling in constant regions, where local correlation can be easily exploited in the reconstruction step. If we consider the problem of reconstruction, the gradient can be used to perform local extrapolation based on Taylor series, instead of simple interpolation.

Irradiance Caching. In the late 80s, Ward et al. proposed an algorithm for efficiently estimating diffuse interreflection that became known as irradiance caching (IC) [WRC88]. This algorithm had a very significant impact and has led to a long stream of publications, and widespread industry use.

IC builds upon two facts, (1) indirect irradiance varies smoothly in most regions, (2) Lambertian surfaces reflect light equally in all directions. The Lambertian assumption simplifies the rendering equation to

$$L_o(\mathbf{x}, \omega_o) = \frac{\rho(\mathbf{x})}{\pi} \int_\Omega L_i(\mathbf{x}, \omega_i)\, \cos(\theta)\, d\omega_i = \frac{\rho(\mathbf{x})}{\pi} E(\mathbf{x}),$$

where $\rho(\mathbf{x})$ is the diffuse reflectance of the surface at position $\mathbf{x}$, $L_i(\mathbf{x}, \omega_i)$ is the irradiance at position $\mathbf{x}$ from direction ω_i, θ is the angle between ω_i and the surface normal at position $\mathbf{x}$, and $E(\mathbf{x})$ is

the irradiance at position $\mathbf{x}$. We see that we can integrate the irradiance, $E(\mathbf{x})$, in a separate step, and only compute the shading (i.e. the multiplication with the surface reflectance) afterwards.

The IC algorithm builds upon these two facts, by proposing to interpolate high quality irradiance samples at sparse locations to reconstruct irradiance at all scene positions visible from the camera. An upper-bound of the irradiance gradient, $\epsilon_i(\mathbf{x})$, is then used in this interpolation scheme to inversely weight nearby irradiance samples, so that an irradiance sample weight is computed as

$$w_i(\mathbf{x}) = \frac{1}{\epsilon_i(\mathbf{x})}.$$

The irradiance gradient upper-bound is computed using a simple model, called the "split-sphere" model because it considers the lighting from a half dark sphere. To detect when a new irradiance sample should be computed, instead of interpolating the existing ones, a validity threshold is defined. If no nearby neighbor has a weight larger than this validity threshold, irradiance is computed explicitly at the current position, and the result is added to the irradiance cache for later use. In practice, IC has a moderate memory overhead, while offering drastic computation improvements. Figure 4.2 illustrates this using the results from Ward et al.'s paper. In particular, we see that irradiance is sampled densily along edges, but very sparsely in uniform regions. The result is significantly smoother than the one using path tracing.

Since its introduction, IC saw multiple improvements. Ward and Heckbert derived the actual irradiance gradient estimate, instead of an upper-bound, that they used to get smoother reconstruction using linear extrapolation with Taylor series [WH92]. Křivánek et al. extended IC to handle low-frequency glossy materials [KGPB05], by storing irradiance using spherical harmonics, instead of scalar, to preserve the directional irradiance information. Jarosz et al. recently proposed Hessian based irradiance extrapolation [JSKJ12], leveraging second order derivatives for more accuracy in the Taylor expansion, and this approach was further improved by Schwarzhaupt et al. [SJJ12]. Despite these improvements, IC remains a specialized method targeted

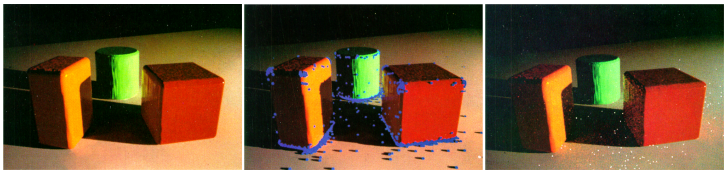


Figure 4.2: Irradiance caching (left), compared to path tracing (right). Irradiance caching yields a much smoother result thanks to the interpolation of high quality cache samples. Cache points are marked using blue spheres (middle). Discontinuities are densely sampled, while smoothly varying regions are sparsely sampled. These images are reproduced from [WRC88].

at handling indirect illumination constrained to diffuse or moderately glossy materials. In contrast, our framework is meant to handle arbitrary light transport effects, as well as arbitrary materials.

4.1.2 Frequency Analysis

Adaptive rendering methods based on frequency analysis all aim to leverage Nyquist's sampling theorem, which states that, for a bandlimited signal, a sampling rate of at least twice the signal bandwidth ensures that no aliasing will occur. Nyquist's sampling theorem also solves the adaptive reconstruction part, since the filter should simply be chosen so as not to overlap spectral replicas. This is a fundamental result in signal processing, and a very strong theoretical basis, since it guarantees the accuracy of the reconstruction. Figure 4.3 illustrates how Nyquist's theorem can be leveraged to handle motion blur and derive a proper space-time filter and sampling rate, that achieves the sparsest sampling density while preventing aliasing.

In rendering, the signal studied by frequency analysis methods is the light field, and the analysis is usually done locally around an individual light ray. All recent methods build on Chai et al's work on sampling for light field rendering [CTCS00], and Durand et al.'s

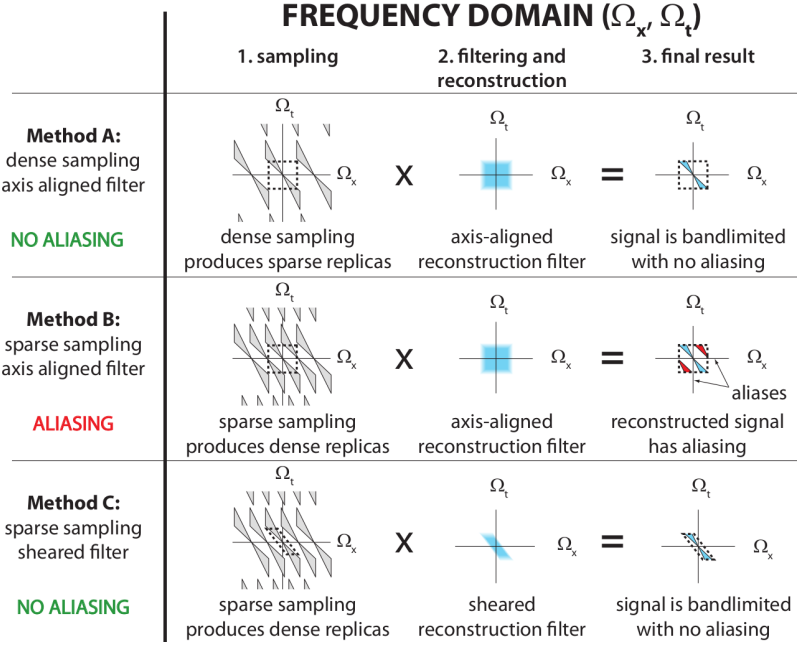


Figure 4.3: Frequency analysis of motion blur showing spectral replicas due to sampling in the space-time spectral domain. The two axes, Ω_x and Ω_t respectively correspond to the space and time dimensions. Increasing the spatial sampling rate spreads the replicas along Ω_x in the spectral domain, ensuring they don't overlap within the support of the reconstruction filter (top row). Using a lower sampling rate compacts the replicas, which causes aliasing artifacts due to spectral overlap (middle row). By using a sheared reconstruction filter, the signal is more tightly covered, which prevents aliasing (bottom row). This figure is reproduced from [ETH+09].

frequency analysis of light transport [DHS+05].

The goal of frequency analysis is to define the bandlimits of the light field spectrum. For instance, reflection off a Lambertian surface

yield a low-frequency signal, whereas occlusions from nearby geometry produces hard shadows with high frequencies. In order to derive the actual bounds of the spectrum, it is common to restrict the analysis to a single light transport effect.

For instance, in the case of motion blur, Egan et al. [ETH+09] have shown that the signal energy is mostly contained in a wedge defined by the minimum and maximum velocities. Once the signal bandlimits have been found, a corresponding filter, that covers as tightly as possible the signal can be derived. Finally, the optimal sampling rate is the one that ensures that no overlap occurs in the support of the chosen filter. We reproduce in Figure 4.4 a result from Egan et al. [ETH+09] illustrating the wedge bounding the motion blurred signal in the frequency domain.

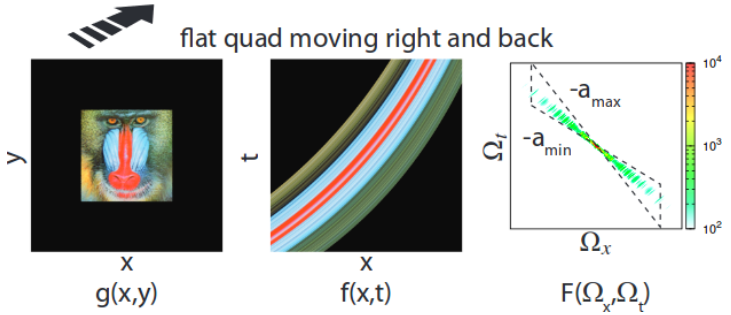


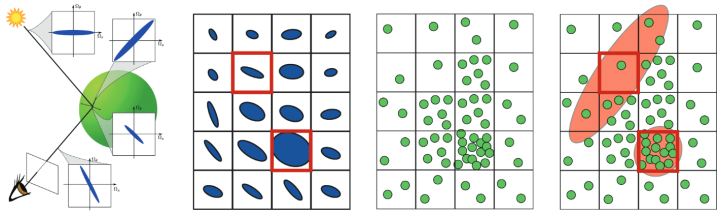
Figure 4.4: Space-time and Fourier domain plots of a moving textured 2d plane. The space-time and Fourier plots correspond to a single scan line of the image on the left. Because of perspective, velocities in the image plane change across time, but a wedge based on the minimum and maximum velocities bounds the resulting frequency spectrum. These images are reproduced from [ETH+09].

Frequency analysis has been applied to many different effects. Egan et al. have also studied soft shadows [EHDR11] and ambient occlusion [EDR11], Soler et al. studied depth of field [SSD+09], Mehta et al. studied soft shadows [MWR12] and diffuse indirect illumina-

tion [MWRD13].

One disadvantage of the frequency analysis methods just mentioned is that they only consider a single effect at a time, and offer no way of handling combinations of effects (for instance, combining motion blur and depth of field). However, recent work by Belcour et al. [BSS+13] proposes a more general approach to frequency analysis. Their approach models the frequency spectrum as a multidimensional Gaussian and traces it through the scene. Each light interaction (transport in free space, occlusion, reflection, etc.) is modeled as an operator applied to the frequency spectrum. This allows to progressively update the frequency spectrum, until the path reaches the image plane, where frequency spectra are accumulated. Once the overall spectrum is obtained, given sufficiently many individual spectra, it is projected in the image plane in order to derive the appropriate sampling rate, as well as the appropriate image space filter for the reconstruction. See Figure 4.5 for an illustration. This effectively samples the frequency spectrum, but allows to combine different effects, with the only constraint that an operator must be defined for each interaction. This method is a good example of a technique crossing the bridge between *a priori* and *a posteriori* methods, since the operators representing light interactions are defined using prior knowledge, but the actual frequency spectrum projected into each pixel is sampled, and therefore only known *a posteriori*.

There are two significant drawbacks to frequency analysis methods. The first comes from their nature: they require an extensive amount of prior information. For instance, for shading the frequency spectra of all BRDFs and texture at every point of the scene are needed. This imposes a large burden on the rendering pipeline and restricts the applicability of the method. The second inconvenience is that the analysis is only locally valid and fails near object boundaries, though this issue can be mitigated through additional heuristics. In contrast, the framework developed in this thesis builds exclusively on *a posteriori* information, and is therefore much less intrusive for the rendering pipeline and applicable to any scene. Additionally, in our framework, filters are defined considering the contributions of a large neighbor-



(a) propagation of 5D covariance matrices (b) per - pixel 5D covariance matrices (c) 5D samples are distributed on the image plane using the density derived from the 5D covariance matrix (d) 2D reconstruction

Figure 4.5: Tracing the frequency spectrum, modeled using 5d Gaussians represented as covariance matrices, through the scene (a). The resulting covariance matrices are accumulated in the image plane (b), and used to define appropriate sampling rate (c) and filter bandwidth (d). These images are reproduced from [BSS+13].

hood around each pixel, whereas frequency analysis only considers the immediate vicinity of each ray. This allows us to build robust filters that cover a large neighborhood, while still correctly adapting to the structures in the image.

4.1.3 Light Field Structure

Researchers have extensively studied the structure of light fields and found that it often is highly anisotropic. The slope of the anisotropy can be linked to various factors, such as depth [CTCS00], velocity [ETH+09], blocker depth [EHDR11], or reflector depth [MWRD13].

Methods have been designed to explicitly exploit this anisotropy by reprojecting samples in the high dimensional light field along these slopes. This allows to take a sparse set of samples, and upsample it to arbitrary rates. During reprojection, the sample value is untouched, which assumes that the value is valid at the new position, as is the case with Lambertian surfaces. For glossy materials, the reprojection is improved by restricting the distance over which samples can be reprojected according to the bandwidth of the material reflectance.

Lehtinen et al. proposed a method that could handle combinations of motion blur, depth of field, and soft shadows [LAC+11], while offering very good reconstructions, even for very sparse input sets. They reproject samples at specific lens-time coordinates, along the know anisotropy slopes, to obtain a scattered set of reprojected samples. The key of their method is a robust criterion to detect occlusions, that they use to accurately treat visibility of the reprojected samples. The reprojection is illustrated in Figure 4.6, where we see samples belonging to two separate surfaces reprojected along their individual slopes to a new position. As we see in the right image of Figure 4.6, the algorithm only considers samples falling within a small radius R of the reprojection location, corresponding to the dispersion of the sample set. The idea is that stochastic sampling will leave holes with a maximum radius R, and that these holes should be filled during reconstruction.

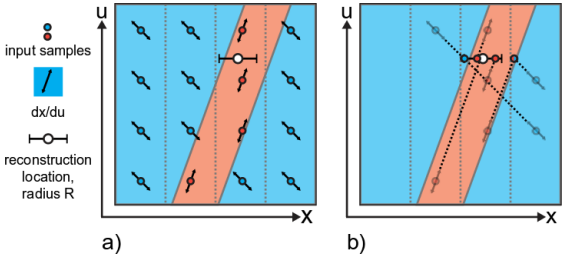


Figure 4.6: Reprojection of samples based on the local anisotropy of the light field (measured by the gradient $\frac{dx}{du}$). Only the samples that reproject within a radius R of the reconstruction location are considered. This figure is reproduced from [LAC+11].

Lehtinen et al. then extended this approach to reconstruction of indirect illumination [LALD12], again by exploiting local anisotropy in the light field.

Reprojection methods have two strong features. First, they are not susceptible to noise in the sample set since they are guided by a prior model of the light field anisotropy. Indeed Lehtinen et al. obtained

good reconstructions even with a single sample per pixel. Second, by operating directly on individual samples, they are able to accommodate conflicting anisotropy slopes. However, these methods are restricted to a specific subset of effects. Our own framework has opposite properties. One the one hand, our framework is susceptible to noise in the input, especially at very low sampling rates where statistics are very unreliable, and our framework cannot accommodate conflicting anisotropy slopes, since we operate on pixels rather than samples. For these reasons, we do not expect our framework to match the presented reprojection methods on the subset of scenes they can handle. On the other hand, our framework can be used on scenes featuring arbitrary light transport effects and arbitrary materials.

4.1.4 Summary Table

We summarize the main characteristics of the *a priori* methods cited in this section in Table 4.1. As can be seen in this table, many effects have been studied over the years, and methods based on studying the light field (either its frequency spectrum or its structure), have proved to be quite flexible. Still, all methods are restricted to a subset of effects, sometimes even a single effect, though recent work shows a clear trend towards generality.

4.2 *A Posteriori* Methods

Our own adaptive rendering framework belongs to the class of *a posteriori* methods. These methods are empirical in nature, in the sense that they start by producing a noisy rendering, which they then analyze. This analysis relies only on the samples' values (or the noisy pixel values), and is then used to define an adaptive sampling strategy, a locally adaptive filtering strategy, or both. The fact that the analysis relies only on the sample values, but not on the various effects (and potential combinations of effects) that lead to these observations, makes

Table 4.1: Summary of the main characteristics of the *a priori* methods cited. In the analysis types, "LF struct" stands for light field structure. In the light transport effects, "II" stands for indirect illumination (an additional "(G)" identifies methods that also support moderately glossy surfaces), "MB" for motion blur, "SS" for soft shadows, "AO" for ambient occlusion, and "DOF" for depth of field. In the sampling strategies, "reproj" stands for reprojection. In the reconstruction strategies, "IS" stands for image space, and "HD" for high dimensional space. For high dimensional reconstructions, the dimensionality depends on the effects being considered. For instance, reconstruction in [ETH+09] is done in the space-time domain (3D), but reconstruction in [LAC+11] is done in the space-lens-time domain (5D).

Method	Analysis	Effects	Sampling	Reconstruction
[WRC88]	gradient	II	heuristic	anisotropic IS
[WH92]	gradient	II	heuristic	anisotropic IS
[KGPB05]	gradient	II(G)	heuristic	anisotropic IS
[JSKJ12]	Hessian	II	heuristic	anisotropic IS
[SJJ12]	Hessian	II	heuristic	anisotropic IS
[ETH+09]	frequency	MB	Nyquist	anisotropic HD
[EHDR11]	frequency	SS	Nyquist	anisotropic HD
[EDR11]	frequency	AO	reproj.	anisotropic HD
[MWR12]	frequency	SS	Nyquist	axis-aligned IS
[MWRD13]	frequency	II(G)	Nyquist	axis-aligned IS
[BSS+13]	frequency	MB,DOF SS	Nyquist	anisotropic IS
[LAC+11]	LF struct.	SS,MB DOF	reproj.	anisotropic HD
[LALD12]	LF struct.	II(G),AO MB,DOF	reproj.	anisotropic HD

this class of method generally applicable.

Adaptive rendering methods can be roughly classified into three

categories: pure adaptive sampling methods, pure adaptive reconstruction methods, and hybrid methods that combine adaptive sampling and reconstruction. For adaptive sampling, the main differentiation factor is the error metric used, while for adaptive reconstruction, the main differentiation factor is the denoising technique used.

This section will present the main three error metrics used in *a posteriori* methods, followed by a high level presentation of the main adaptive rendering methods proposed over the years. However, we will not explicitly cover the denoising techniques used for adaptive reconstruction. Given the wealth of such denoising techniques, we chose to cover this subject separately in Chapter 5, where we will focus on the main image space techniques.

4.2.1 Error Metrics

The analysis in *a posteriori* methods relies on an error metric, which will then guide the adaptive sampling, the adaptive reconstruction, or both. For the sampling step, regions with a larger error will be sampled more densely, and for the reconstruction step, regions with a larger error will be denoised more aggressively.

Three metrics in particular have been relatively widely used over the years: the contrast heuristic, perceptual models, and the MSE.

The Constrast Heuristic. The contrast heuristic was the metric used in Mitchell's seminal paper on adaptive sampling [Mit87], and later on in multiple adaptive frameworks [HJW$^+$08, ODR09, CWW$^+$11]. Mitchell argued for the use of contrast rather than sample variance, because contrast had been shown to be a better measure of the visual perception of local variation. This contrast metric is computed as

$$C = \frac{I_{max} - I_{min}}{I_{max} + I_{min}},$$

where I_{max} and I_{min} are, respectively, the maximum and minimum intensity values of the recorded samples.

This contrast metric is directly related to the relative standard deviation of the two-sample set $\{I_{max}, I_{min}\}$. To show this, we start from the uncorrected sample variance definition,

$$S_n^2 = \frac{1}{n}\sum_{i=1}^{n}(I_i - \bar{I})^2.$$

The term "uncorrected" refers to the fact that this computation of the sample variance is a biased estimator of the actual random variable variance (see Section 3.8 for more information). For the set $\{I_{max}, I_{min}\}$, we have $n = 2$, and

$$\bar{I}_* = \frac{I_{max} + I_{min}}{2}.$$

The subscript "$_*$" indicates that this estimation is for the two-sample set containing only the maximum and minimum sample values. We can then define the corresponding uncorrected sample variance,

$$\begin{aligned} S_*^2 &= (I_{max} - \bar{I}_*)^2 + (I_{min} - \bar{I}_*)^2 \\ &= \left(\frac{I_{max} - I_{min}}{2}\right)^2 + \left(\frac{I_{min} - I_{max}}{2}\right)^2 \\ &= \left(\frac{I_{max} - I_{min}}{2}\right)^2, \end{aligned}$$

and the corresponding uncorrected relative sample standard deviation, or coefficient of variation,

$$c_{v*} = \frac{S_*}{\bar{I}_*} = \frac{I_{max} - I_{min}}{I_{max} + I_{min}} = C.$$

The contrast metric therefore corresponds to a conservative estimation of the uncorrected relative standard deviation that considers only the two extreme sample values.

The main inconvenience of this metric is that it is very sensitive to measurement noise, since outliers dominate the evaluation, i.e. the

metric degenerates to $C = 1$ if $I_{max} \gg I_{min}$. In particular, $C = 1$ whenever $I_{min} = 0$. Given the large magnitude of outliers when computing indirect illumination, and the relatively high rate of paths carrying a null energy in scenes with light sources that are hard to reach, we believe this metric is ill-suited for MCPT since it tends to converge to $C = 1$ over the whole image plane for complex scenes.

Perceptual Metrics. Given that renderings are usually meant to be viewed by human observers, it is natural to consider errors from a perceptual point of view. The logic is that if an error is not noticeable by a human observer, we should concentrate the rendering effort elsewhere. This principle is at the core of many lossy compression schemes for audio or video files and has also been leveraged by some adaptive rendering scheme.

Bolin and Meyer [BM98] proposed a perceptually based adaptive sampling algorithm. Their goal was two-fold: first, they wanted to have an automated image evaluation that would allow a program to detect and correct rendering artifacts without any input from the user; second, they wanted to have the ability of prescribing the desired visual quality, which would allow to mix rendering algorithms while ensuring a uniform visual quality. They propose to use a simplified vision model, based on the Haar wavelet decomposition, that can be refined efficiently while rendering the scene, as samples are added. Their system estimates the perceptual error by comparing to boundary images which are obtained from the wavelet coefficients and their variance, by adding or subtracting the noise standard deviation from the coefficients. Samples are then placed in regions having the largest predicted perceptual error.

Ramasubramanian et al. [RPG99] propose an adaptive sampling algorithm that predicts the threshold of detectable artifacts. Their model accounts for the fact that the human visual system is less sensitive in regions with strong background illumination, high spatial frequencies, and high contrast level. Ramasubramanian et al. observed that the cost of evaluating the perceptual metric is prohibitive, even for simplified models such as the one used by Bolin and Meyer, which

offsets the gain of the adaptive sampling. Consequently, Ramasubramanian et al. propose to use a pre-processing step that encodes the spatially-dependent component of their model. While rendering, they can then update the target luminance threshold without incurring the cost of spatial analysis. They apply this approach to global illumination. They pre-process the direct illumination solution, with the addition of an approximative ambient term, to encode the response to spatial frequencies and contrast, and then update the luminance information as they render the global illumination. The main limitation of this method is that it assumes a direct illumination solution is readily available.

Perceptual metrics are undeniably valid choices, especially for guiding adaptive sampling rates, however they are notoriously expensive, even though the method of Ramasubramanian et al. greatly mitigates their cost at the expense of some approximations. Our own framework does not make use of a perceptual metric, and this would certainly be an interesting extension.

The Mean Squared Error, an Objective Metric. The MSE is the metric we chose to use in our adaptive framework. It is very widely used, mostly because it is very easy to compute, and usually a good predictor of signal quality. Given an estimator $\hat{O}$, its MSE is

$$\mathrm{MSE}\left[\hat{O}\right] = \mathrm{E}[(\hat{O} - O)^2] = \mathrm{Var}\left[\hat{O}\right] + \mathrm{Bias}\left[\hat{O}\right]^2.$$

Please see Section 3.9 for a full derivation. The fact that the MSE is directly related to the variance and bias of the estimator is particularly useful, and was central to the first version of our adaptive framework (see Section 6.2). Also, given that the variance of the sample mean decreases linearly with the number of samples, i.e. $\mathrm{Var}\left[s_n^2\right] = \sigma^2/n$, we can predict the variance of the reconstructed image with more samples (and therefore its MSE, assuming the same filter), which can be used to estimate the expected error reduction of additional samples.

In our framework, we make use of the relative MSE (rMSE), rather than the MSE, to prevent bright regions from dominating the

error estimation. This coarsely models the fact that human observers are less sensitive to errors in very bright regions. The relative MSE is computed as

$$\mathrm{rMSE}\left[\hat{O}\right] = \frac{\mathrm{E}[(\hat{O} - O)^2]}{O^2 + \epsilon},$$

where ϵ is a small value to prevent overweighting very dark regions, as well as divisions by zero.

There has been some strong (and valid) criticism of MSE as an error metric, as well as similar metrics such as absolute difference or peak signal-to-noise ratio (PSNR). The main argument against the MSE is that it is too sensitive to outliers. In their paper introducing the Structural SIMilarity (SSIM) metric, Wang et al. [WBSS04b] present a compeling illustration of various degraded images with very different artifacts that all have the same MSE that underlines the shortcomings of the MSE. Still, a recent study of state-of-the-art image quality metrics by Čadík et al. [CHM+12] showed that no image quality metric performs better than the other, even including the absolute difference metric, which is equivalent to the MSE.

4.2.2 Adaptive Sampling and Reconstruction

Historically, adaptive sampling long predated adaptive reconstruction. Indeed, Whitted's seminal paper on ray tracing [Whi79] in 1979 already proposed an adaptive sampling scheme. In contrast, the first notable paper on adaptive reconstruction was Rushmeier and Ward's splatting algorithm [RW94], which was published in 1994. This is most likely due to an aversion to the implicit bias introduced by the denoising techniques used in adaptive reconstruction. Yet, recent methods have shown that adaptive reconstruction can be strikingly efficient, often significantly so compared to adaptive sampling, and the current state of the art methods all combine adaptive sampling and reconstruction.

A notable point of *a posteriori* adaptive rendering is that the vast majority of methods proposed over the years operate on pixel values

in image space. This was a natural choice for the early methods, which were severely constrained by the hardware at the time, both from a computational point of view, and a memory point of view. However, even the latest state-of-the-art methods operate on pixel values and in image space. There are two known algorithms that operate on sample values, rather than pixels: Sen and Darabi's Random Parameter Filtering (RPF) algorithm [SD12], and Hachisuka et al.'s Multidimensional Adaptive Sampling and reconstruction (MDAS) algorithm [HJW+08]. The MDAS algorithm is also the only notable algorithm operating in a high dimensional space, rather than image space. Both of these algorithms (RPF and MDAS) are notoriously slow, which explains the popularity of pixel based image space methods. Interestingly, recent work on *a priori* methods, in particular the work of Mehta et al. [MWR12, MWRD13] has argued in favor of simpler image space reconstruction filters for performance reasons.

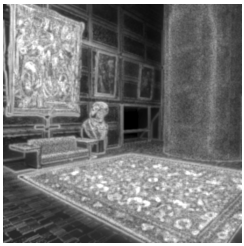
We will now present the main *a posteriori* adaptive rendering methods, starting with the adaptive sampling methods, then the adaptive reconstruction methods, and finally the hybrid methods leveraging both adaptive sampling and reconstruction, which are now prevalent.

Adaptive Sampling Methods. As previously mentioned, Whitted's paper on ray tracing already proposed an adaptive sampling scheme. The idea was to evaluate the signal value at edges of a square region, and subdivide it if the values differed too much. Kajiya's rendering equation paper [Kaj86], which introduced the path tracing algorithm, mentioned the possibility of adaptive hierarchical integration. The core idea was to recursively subdivide the sampling space according to some subdivision threshold. Kajiya proposed a few threshold functions, including one that would measure the variance in a subnode of the hierarchy, but noted that no adaptive criteria had been found to be particularly effective.

The following year, in 1987, Mitchell proposed his seminal work on adaptive sampling [Mit87]. He noted that variance was not a good adaptive criterion, since it could accurately indicate high frequency, but not measure the perceptual impact. Instead, Mitchell proposed to

use the contrast heuristic which more closely modeled the response of the human eye to rapid changes in light intensity. Mitchell used this contrast metric in a simple two-level sampling strategy, where pixels are either sampled with a low density, or with a high density. The image is then divided in *supersampling cells* containing eight or nine low-density samples. The contrast metric is evaluated, in image space, separately on the red, green, and blue channels, and if any channel is higher than a predefined threshold (0.4, 0.3, and 0.6 respectively), the whole cell is selected for high-density sampling. A lower contrast threshold was used for green, since the human visual system is more sensitive to green (the corresponding cone cells are more common in the human retina). This simple adaptive sampling strategy was shown to give good results combined with a basic ray tracer. Mitchell observed that the adaptive sampling rate caused a problem during reconstruction, since densely sampled pixels would have a larger weight (if the reconstruction pixel spans more than a single pixel). To solve this issue, Mitchell rendered images on a subpixel grid, and then applied a recursive box filter which ensured gave uniform importance to all pixels, no matter their sampling rate. We make use of a similar subpixel grid strategy in our own framework (see Section 6.4).

While Mitchell's contrast metric was inspired by studies on the human visual system, it was still a very simple one. Logically, later work proposed to use perceptual metrics that attempted to accurately measure the human visual system response. This lead to the work of Bolin and Meyer [BM98], Ramasubramanian et al. [RPG99], and Farrugia and Péroche [FP04]. All perceptually based methods share the same common idea, that sampling density should be locally increased until all pixels are below a pre-defined error threshold. Alternatively, samples could be distributed to give a uniform error level across the whole image plane, for a fixed sampling budget. As an example, we reproduce in Figure 4.7 the result of Ramasubramanian et al's method which uses a perceptually based error metric to detect convergence. Their method improves the indirect illumination solution until its artifacts are not perceptible when combined with the direct illumination solution.



threshold map

sample density

adaptive indirect illumination

direct illumination + adaptive indirect illumination

Figure 4.7: Adaptive sampling of indirect illumination using a perceptually based error metric. A per-pixel luminance threshold under which artifacts are not visible is computed (top-left). The resulting sampling density (top-right) yields an indirect illumination solution (bottom-left) with artifacts, that are not visible once added to the direct illumination solution (bottom-right).These images are reproduced from [RPG99].

Perceptual models are obviously a very sound choice to guide adaptive sampling from a theoretical point of view. However, in practice, their computational cost and inherent complexity mitigates their contribution. Because of this, recent methods reverted to simpler metrics to guide adaptive sampling, such as Mitchell's contrast heuristic or, more recently, the relative MSE.

Adaptive Reconstruction Methods. Initial work on reconstruction was not concerned with minimizing rendering noise, but instead with minimizing aliasing artifacts. The focus was mainly on the spectral properties of the reconstruction filter. Based on signal processing theory, the goal of these early works was to approximate the ideal sinc filter (which corresponds to a box filter in frequency space). A well known result is Mitchell's piecewise cubic filter [MN88]. Reconstruction in itself was not adaptive, but uniform. A single pixel filter was chosen, and then used to reconstruct all pixels in the image.

While initial work on reconstruction was focused on aliasing artifacts, the focus of adaptive reconstruction is on minimizing noise artifacts. As its name implies, adaptive reconstruction does not use a fixed reconstruction filter for the whole image, but adapts the reconstruction process to the local characteristics of the image. Rushmeier and Ward proposed to distribute the energy of noise samples to their neighborhood [RW94]. Given a measure of the amount of excess energy, a filter of appropriate size is defined. The size is determined so that the amount of energy redistributed to each neighbor pixel does not exceed a predefined threshold. The approach of Rushmeier and Ward is an example of parameter estimation, with the parameter estimated being the size of the filter in this case. The first implementation of our filter employs a similar reconstruction scheme, based on a filterbank of isotropic Gaussian filters. One particularity of Rushmeier and Ward's algorithm, is that the filter is used to splat samples, instead of gathering contribution from neighbor pixels. Since the filter weights integrate to 1, this implicitly ensures energy preservation.

While Rushmeier and Ward's method operated on samples, most methods instead operate on pixel values. In this case, a noisy rendering is first reconstructed using a standard uniform pixel filter, and this noisy rendering is then denoised. This is the approach employed by McCool, who used anisotropic diffusion to perform denoising [McC99]. Anisotropic diffusion was constrained by a conductance map, built using a color coherence metric between adjacent pixels. Because of noise in the rendering, the color coherence metric is not always reliable. To improve it, McCool proposed enhancing the con-

ductance map using scene information stored in two feature buffers: a depth buffer and a normal buffer, which respectively encoded the depth and normal at the primary intersection point of each pixel. These feature buffers, as well as the corresponding conductance maps, are shown in Figure 4.8. The idea of using feature buffer to increase the robustness of denoising proved to be particularly effective, and current state-of-the-art methods heavily rely on such buffers.

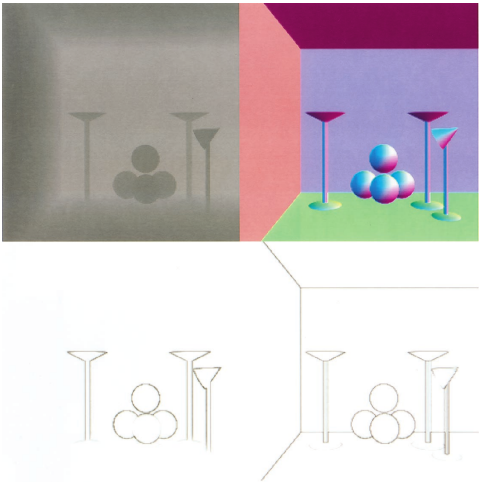


Figure 4.8: Feature buffers store the depth of the primary intersection point (top left), and the normal of the primary intersection point (top right). These buffers are then used to derive conductance maps (bottom left and bottom right), that encode the image space discontinuities contained in the feature buffers. Images reproduced from [McC99].

Xu and Pattanaik [XP05] proposed to denoise MCPT renderings using a bilateral filter, a structure aware filter (see Section 5.5 for a full description of the filter). The weights of the bilateral filter are defined by the affinity between pixels, which is a function of the spatial distance, but also the photometric distance. This naturally pre-

vents filtering across edges, since pixels on separate sides of an edge have a large photometric distance. Unfortunately, the photometric distance computation is unreliable in presence of noise, particularly in the presence of outliers. Xu and Pattanaik proposed to compute the photometric distance on a low-pass filtered version of the image, to mitigate the impact of outliers.

Dammertz et al. [DSHL10], like Xu and Pattanaik, proposed a method leveraging a structure aware filter. Instead of addressing the noise issue with a pre-filtering pass as in the work of Xu and Pattanaik, they instead used feature buffer as proposed by McCool. The feature buffers are used in a cross-filtering scheme, that is, the affinity between pixels is computed on the feature buffers, and the resulting weights are applied on the color buffer. This approach builds on the fact that feature buffers are noise free in the absence of distribution effects such as motion blur or depth of field. A similar idea was developed by Bauszat et al. [BEM11], using local regression (see Section 5.3 for a full description of local regression), based on the values of feature buffers. Because they assume feature buffers to be noise free, the methods of Dammertz et al. and Bauszat et al. have a limited applicability. Given that both methods are concerned with interactive rendering, this limitation is a decent trade-off. The goal of our framework is fundamentally different, since we are concerned with offline rendering where time budgets are much larger, but we want to handle all effects supported by MCPT.

While cross-filtering using feature buffers is a robust solution when the feature buffer is noise free, it breaks down when these buffers are themselves noisy, which happens with distribution effects such as motion blur or depth of field. Sen and Darabi [SD12] proposed a method that would detect noisy features, by measuring their dependency on the Monte Carlo random parameters. In their algorithm, the noisier a feature is, the less it contributes to the affinity computation. Sen and Darabi's algorithm operates on individual samples, which is computationally expensive. Our own framework operates on pixel values, and has a drastically lower overhead.

Hybrid Methods. The most successful adaptive rendering methods combine both adaptive sampling and adaptive reconstruction. The first method to explicitly leverage both strategies was Hachisuka et al.'s Multidimensional Adaptive Sampling and reconstruction (MDAS) algorithm [HJW+08]. The MDAS algorithm made use of Mitchell's contrast heuristic to guide the adaptive sampling, while reconstruction was performed using nearest neighbor interpolation. The MDAS algorithm is notable for being the only example of an *a posteriori* method operating directly in the high dimensional sampling space. Working in the high dimensional sampling space allowed sampling to explicitly target high dimensional discontinuities, without resorting to brute force sampling. This efficiency however, came at the cost of scalability, since the cost of performing reconstruction grows exponentially with the dimensionality in their algorithm. Consequently, while the MDAS algorithm could in theory be applied to arbitrary scenes, in practice it is restricted to scenes with a relatively low dimensionality (5D or less).

Overbeck et al. proposed an image space algorithm which proposed to render images directly in the wavelet domain. This allowed a multi-scale analysis of the rendering as sampling was performed. The adaptive sampling was also guided Mitchell's contrast metric, but combined with the wavelet coefficients. The contrast metric would identify noise in smooth regions, while wavelet coefficients would identify edge regions. Samples were then distributed at an appropriate scale of the wavelet decomposition (a coarse scale for noise in smooth regions, and a fine scale for edges). Finally, a smooth reconstruction was obtained by wavelet thresholding, that is, setting to zero small wavelet coefficients that encode noise (see Section 5.2 for a description of wavelet thresholding).

Overbeck et al.'s algorithm was the main inspiration for our framework. The first implementation of our framework used a similar multiscale approach, but using a filterbank of isotropic Gaussian filters instead of wavelet decomposition (see Chapter 6 for all details). For each pixel, we would select the filter yielding the smallest reconstruction MSE, and we estimated the MSE using an analytical model de-

rived specifically for Gaussian filters. Using the MSE, instead of a heuristic as in Overbeck et al.'s work, proved to be much more robust and lead to significantly improved results, despite using a simpler filtering scheme (isotropic Gaussians instead of wavelet thresholding). Li et al. [LWC12] extended our work by using a filterbank of cross-bilateral filters (leveraging feature buffers), instead of isotropic Gaussians. They estimated the MSE of their cross-bilateral filters using Stein's Unbiased Risk Estimate (SURE, see Section 3.9.2 for a full description). The use of cross-bilateral filtering yielded significantly improved results, but residual artifacts remained due to robustness issues with noisy feature buffers. Following Li et al.'s work, the latest implementation of our framework, presented in Chapter 8, uses SURE to estimate the MSE of the filtered output.

Kalantari and Sen [KS13] proposed a generic framework that could leverage state-of-the-art image denoising techniques, and apply them to MCPT renderings. The issue with classical image denoising techniques is that they assume uniform variance across the image plane, whereas the variance of MCPT renderings is highly non-uniform. In order to leverage image denoising techniques, one has first to reformulate them assuming non-uniform variance. Kalantari and Sen's approach proposed to simply perform multiple denoising passes assuming a range of variance values, and then locally interpolate the various filtered images according to the actual variance of each pixel. This algorithm gave good results, but could only be applied after tone mapping (since most image denoising algorithms assume low dynamic range images), and could not leverage feature buffers.

4.2.3 Summary Table

Table 4.2 summarizes the main characteristics of the *a posteriori* methods cited in this section and the three implementations of our framework. This table does not list the light transport effects supported, since all cited methods are generic (though some, like [DSHL10] give bad results if their assumption of noise free features is violated).

Table 4.2: Summary of the main characteristics of the *a posteriori* methods cited. The three implementations of our framework are identified with an "*". In the "Strategy" column, "AS" stands for Adaptive Sampling, and "AR" for Adaptive Reconstruction. In the "Reconstruction" column, "IS" stands for Image Space, and "SS" for Sampling Space (an additional "(F)" identifies methods that leverage information contained in feature buffers). The "Metric" column indicates the metric used to guide adaptive sampling, and is therefore empty for methods performing only adaptive reconstruction. Adaptive reconstruction does require a metric to set the aggressiveness of the denoising, and this metric is always the signal variance. In the "Reconstruction" column, "struct. aware denoising" refers to the use of filters that adapt their weights to the signal value, in order to preserve its structure, like the bilateral filter (see Section 5.5).

Method	Strategy	Metric	Reconstruction
[Mit87]	AS	Contrast	IS fixed pixel filter
[BM98]	AS	Perceptual	IS fixed pixel filter
[RPG99]	AS	Perceptual	IS fixed pixel filter
[FP04]	AS	Perceptual	IS fixed pixel filter
[RW94]	AR	-	IS parameter estimation
[McC99]	AR	-	IS(F) anisotropic diffusion
[XP05]	AR	-	IS struct. aware denoising
[DSHL10]	AR	-	IS(F) struct. aware denoising
[BEM11]	AR	-	IS local regression (F)
[SD12]	AR	-	IS(F) struct. aware denoising
[HJW+08]	AS+AR	Contrast	SS nearest neighbor
[ODR09]	AS+AR	Contrast	IS wavelet thresholding
[RKZ11]*	AS+AR	MSE	IS parameter estimation
[RKZ12]*	AS+AR	MSE	IS struct. aware denoising
[LWC12]	AS+AR	MSE	IS(F) struct. aware denoising and parameter estimation
[RKZ11]*	AS+AR	MSE	IS(F) struct. aware denoising

Chapter 5

Image space denoising techniques

In this chapter, we present the main image space denoising techniques that have been applied to the problem of reducing the variance of MCPT renderings. The core idea of image denoising is to reduce variance by estimating each pixel as a weighted average of its neighborhood,

$$\hat{u}(p) = \sum_{q \in N(p)} w(p,q)u(q), \tag{5.1}$$

where p is the pixel considered, $\hat{u}(p)$ the filtered color value at pixel p, $N(p)$ is a square neighborhood centered on p, and $u(q)$ the color of the neighbor pixel q. This formulation can be trivially extended to handle (R,G,B) triplets, but we will use the single-channel notation for simplicity. Equation 5.1 builds on the assumption of local similarity, which implies that nearby pixels are reasonably good predictors of the value of a pixel.

Estimating the value of a pixel as a weighted average of its neighborhood is a fundamentally biased process, where the color of each

neighbor q is used as an estimator for the color at pixel p. Consequently, the bias of the filtered value, $\hat{u}(p)$, is the weighted average of the bias at each neighbor q,

$$\begin{aligned}\text{Bias}\left[\hat{u}(p)\right] &= \sum_{q\in N(p)} w(p,q)\text{Bias}\left[u(q)\right] \\ &= \sum_{q\in N(p)} w(p,q)(\text{E}[u(q)] - \text{E}[u(p)]).\end{aligned}$$

The main appeal of image space denoising techniques is that they have a formidable variance reduction potential, at a very moderate computational cost. The variance of the filter output is

$$\begin{aligned}\text{Var}\left[\sum_{q\in N(p)} w(p,q)u(q)\right] &= \sum_{q\in N(p)} \text{Var}\left[w(p,q)u(q)\right] \\ &= \sum_{q\in N(p)} w(p,q)^2\,\text{Var}\left[u(q)\right].\end{aligned}$$

In a simple case with constant variance, $\text{Var}\left[X_i\right] = \sigma^2$, the sum of the squared filter weights directly define the variance reduction of the filter. For instance, for a 3×3 filter, we have $\sum_{i=1}^{9}(1/9)^2 = 1/9$, i.e. the variance will be reduced by a factor of 9. To get a similar variance reduction through sampling, one would have to increase the sampling rate by a factor of 9. For MCPT, this would mean increasing the rendering time by a factor of 9 as well, whereas the filtering cost would generally be much lower. In practice, rendering times are often in the order of minutes, or even hours, while filtering times are generally in the order of seconds. Figure 5.1 illustrates the two strategies (increasing the sampling rate versus increasing the filter support) for box filters of size 3×3, 9×9, and 27×27. Figure 5.1 illustrates one inconvenience of filtering, which is that it can only get rid of the low frequencies of the noise using a very large filter support. The residual low-frequency noise is quite visible in smooth areas, as the result using a 9×9 box filter illustrates. This will prove to be an important issue, especially for video sequences where uncorrelated low-frequency

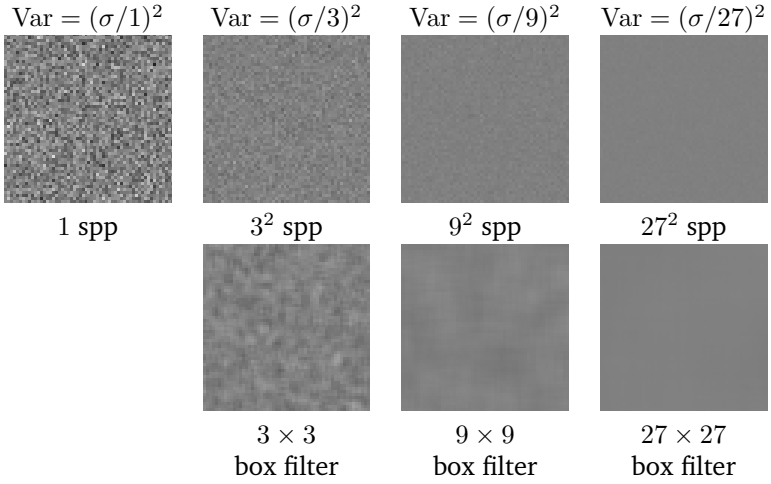


Figure 5.1: Sampling a uniform gray image ($f = 0.5$) with added Gaussian noise of standard deviation $\sigma = 0.1$. In the top row, we take increasingly many samples, while in the second row with filter the image obtained with 1 sample per pixel (spp) using increasingly large box filters. For each column, the variance of both outputs is the same.

noise leads to large scale flickering that is very noticeable. Naturally, the example given in Figure 5.1 is a simplified example, meant to illustrate the appeal of filtering. In practice, our task will be much more complicated, since filtering will be done in non uniform areas, where edges must be preserved.

A full survey of denoising techniques would be prohibitive, and we will therefore restrict ourselves to image space techniques that have been used in adaptive rendering methods. Section 5.1 covers parameter estimation, where the goal is to select the optimal parameters of the denoising filter. Section 5.2 covers transform domain filtering, where processing occurs after transforming the image into a sparse domain. Section 5.3 covers regression, where the image is reconstructed by fitting observed values to a model. Section 5.4 covers

anisotropic diffusion, where the image is denoised using a constrained diffusion process. Section 5.5 covers bilateral filtering, where weights are based on a pairwise affinity model on individual pixels, and its extension, the non-local means filter, which extends the affinity model to consider square patches of pixels. Lastly, Section 5.6 covers the BM3D algorithm of Dabov et al. [DFKE07], an hybrid technique that combines transform domain filtering with the spatial domain patch matching of the non-local means filter.

5.1 Parameter Estimation

Given a filter with one or more parameters, the goal of parameter estimation is to select the set of parameters yielding the best reconstruction. Usually, parameter estimation is done on a single parameter to simplify the analysis. The core question in parameter estimation is defining a metric evaluating the quality of the reconstruction.

A common approach is to select the parameter yielding the lowest MSE. Recall from Section 3.9 that the MSE of an estimator $\hat{O}$ is the sum of its variance and squared bias

$$\mathrm{MSE}\left[\hat{\theta}\right] = \mathrm{Var}\left[\hat{\theta}\right]^2 + \mathrm{Bias}\left[\hat{\theta}\right]^2.$$

A more aggressive filter will yield a lower variance, but also an increased bias. The key is in finding the best balance between the two. We illustrate the conflicting constraints of variance and bias in Figure 5.2, by filtering a simple 1D function corrupted with additive Gaussian noise. In the left image, we added a significant amount of noise, and in the right image, we added very little noise. We denoise the noisy data (blue plot) using two Gaussian filters. The first one has a small standard deviation (green plot, $\sigma = 2$), and the second one has a large standard deviation (red plot, $\sigma = 4$). In the right image, with little additive noise, both filters yields a smooth output, and the larger bias introduced by the larger filter is obvious. In the left image, with significant additive noise, the difference in variance reduction of both filters is visible on the wide lobe: the small filter output has

significant residual variance, whereas the large filter output is smooth.

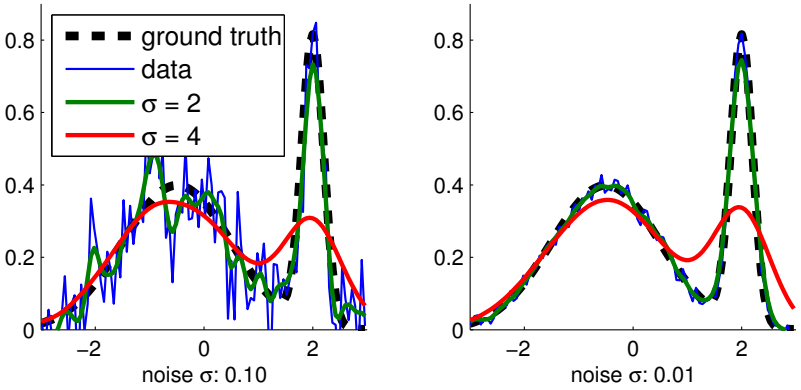


Figure 5.2: Filtering noisy data corrupted with additive Gaussian noise (blue plot) using Gaussian filters of varying standard deviation (green and red plots). The smaller filter (green plot) is less biased, but its output is still quite noisy on the left image where data has a high variance. The larger filter yields a smooth output in both cases, but is significantly biased. The wide lobe of the function is better resolved by the larger filter in the left image, and by the smaller filter in the right image, underlying the need to balance the variance reduction and the increased biased, in order to minimize the overall MSE.

Parameter estimation based on MSE minimization raises the question of estimating the MSE of the filtered output. This can be done in an analytical way, which is the approach we use in the first implementation of our framework (see Chapter 6, Section 6.2), or using a generic tool such as Stein's Unbiased Risk Estimate (SURE, see Chapter 3, Section 3.9.2 for more information).

Katkovnik proposed another approach for parameter selection, the intersection of confidence interval (ICI) algorithm [Kat99]. The core idea of ICI is to progressively adjust the parameter of the filter to make it more aggressive, while verifying that each new filtered output is

consistent with the previous ones. Consistency is defined according to each output confidence interval, based on the output value and its variance. This process is illustrated in Figure 5.3. In the ICI algorithm, the shifting center of each confidence interval models the growing bias, while the shrinking interval range models the diminishing potential variance gain.

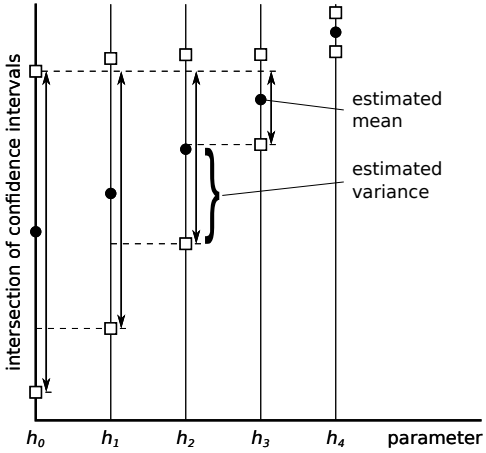


Figure 5.3: The Intersection of Confidence Intervals (ICI). We progressively increase the aggressiveness of a filter by adjusting its parameter h, while ensuring consistency of the each filtered output with all previous ones. Consistency is defined by checking for an existing intersection over all confidence intervals. The confidence interval is centered on the filtered value, while the interval range is proportional to the filtered value variance. The shifting interval center models the increased bias, while the shrinking interval models the diminishing variance reduction potential. The ICI algorithm selects the last parameter value with a consistent output (h_3 in this case).

Parameter estimation was used in multiple adaptive rendering algorithms [RW94, RKZ11, LWC12]. In each case, it was used to define the size of the filter neighborhood.

5.2 Transform Domain Denoising

In transform domain denoising, the signal is transformed from the spatial domain to an other domain where it has sparse support. Filtering occurs in this sparse domain, where signal can be more easily preserved, before transforming the signal back into the spatial domain. Transformation is performed by projecting the signal onto a set of basis functions, and the inner product of the signal with each basis functions defines the coefficients of the transform. A transform domain is said to have sparse support when most transform coefficients have little to no magnitude.

The most direct approach to transform domain denoising is coefficient thresholding, which we will illustrate using the wavelet transform. However, the principle of coefficient thresholding holds for all transforms. The wavelet transform encode both the frequencies of the signal and their locations, along specific orientations based on the wavelet basis used, and therefore allows for a local and anisotropic processing of images at a specific frequency subband. The frequency subband corresponds to the wavelet scale, where coarse scales encode low frequencies, and finer scales encode high frequencies. Coefficient thresholding consists in setting to zero coefficients assumed to be encoding noise, that is, coefficients with a magnitude smaller than the noise standard deviation. Thresholded coefficients $\hat{C}$ are computed as

$$\hat{C} = \begin{cases} C, & \text{if } |C| \geq \alpha\sigma_C \\ 0, & \text{otherwise} \end{cases}$$

where C are the wavelet coefficients, σ_C is the standard deviation of the noise in the coefficients, and α is a user defined parameter (typically between 1 and 3). Alternatively, it is possible to perform *soft* thresholding by subtracting the noise from the coefficients,

$$\hat{C} = sign(C) \cdot \max\left(0, C - \alpha\sigma_C\right).$$

Figure 5.4 illustrates how the well-known Cohen-Daubechies-Feauveau 9/7 (CDF 9/7) wavelets can be used to decompose a noisy image, and

denoise it using soft thresholding. Thresholding can be improved by choosing the α value leading to the lowest MSE, and Donoho and Johnstone [DJ95] proposed to do this using SURE-based minimization (see Chapter 3, Section 3.9.2 for a description of SURE). Wavelet

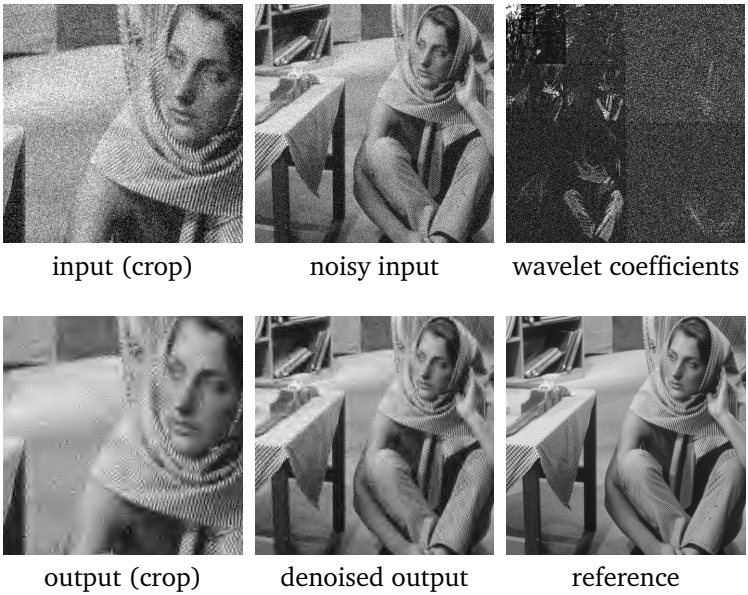


Figure 5.4: Denoising using wavelet thresholding. An image corrupted with Gaussian noise of standard deviation $\sigma = 0.1$ is decomposed using CDF 9/7 wavelets. The reconstruction after soft thresholding of the coefficients is much smoother and preserves the main features of the image. The crop output suffers from the well-known ringing artifacts of wavelet thresholding.

thresholding has been used extensively for denoising, and particularly in the current state of the art image space filter, BM3D, that will be further detailed in Section 5.6. In the context of MCPT, wavelet thresholding was used by Overbeck et al. [ODR09].

Another successful wavelet domain denoising approach was proposed by Portilla et al. [PSWS03]. Their algorithm builds on the observation that the amplitudes of nearby coefficients are strongly correlated, and that local clusters of coefficients can be modeled using Gaussian Scale Mixtures (GSM). The GSM model is then used to directly estimate the expected value of each noisy coefficient.

The main inconvenience of basic wavelet domain denoising methods is that they are prone to ringing artifacts along edges, as well as localized artifacts in smooth regions (both of which can be seen in Figure 5.4). These artifacts occur following the thresholding of fine scale (i.e. high frequency) coefficients, which are found along edges. The ringing nature of these artifacts is directly related to the shape of the wavelet bases. These artifacts highlight a fundamental issue with wavelets, which is that they can only efficiently handle zero-dimensional singularities. However, in image processing, zero-dimensional singularities (i.e. point singularities) are rare. The most common singularities in images are 1-dimensional edges (lines), which cannot be efficiently represented by conventional wavelets [CD99]. This observation has lead to other representation better suited for such singularities, such as wedgelets [Don99], ridgelets [CD99], and others.

In contrast to transform domain denoising, the three implementations of our filter (presented in Chapters 6, 7, and 8) operate directly in the spatial domain, and therefore do not suffer from ringing artifacts due to manipulation of wavelet coefficients. In addition, the implementations of our framework presented in Chapters 7 and 8 leverage a data-adaptive filter that directly encodes the edges of the image.

5.3 Regression

Regression analysis can be used to estimate the relation between one variable, and multiple other variables. Intuitively, the goal of a regression analysis is to predict the output y of a function $f(x)$. This is

done by first establishing a plausible regression model, that is then fitted onto the data in a least-squares sense. This idea is illustrated in Figure 5.5, where we fit a small data set using polynomials of increasing degrees. We perform both global and local regression. In global regression, all data points have the same weight. In local regression, data points near the evaluation point are given a larger weight when fitting the model. Using local regression, even a simple polynomial of degree 1 (a line) gives a smoothly varying reconstruction. As the degree of the polynomial increases, it starts to fit more precisely the input data. The "perfect" fit obtained using a polynomial of degree 6 is not necessarily desirable, as the data points could be noisy, in which case the model would actually be fitting the measurement noise. This is usually referred to as "over fitting". Note that, when using a polynomial of degree 6, the result of global and local regression is the same. Some non-parametric methods for regression analysis (where the fit model is derived from the data itself) exist, but they require larger data sets.

Regression analysis is closely related to machine learning, since it can be used to fit a model on training data, and then apply it to predict the result on new data points. In the context of denoising, regression analysis is used to smooth out a signal. For instance, in Figure 5.5, the curves fitted using polynomials of degrees 1 and 2 are smooth, but still follow the general trend of the data.

The guided image filter, introduced by He et al. [HST10], is based on local regression analysis. Their approach builds on a local linear model (that is, a polynomial of degree 1), combined with a guide image. Formally, this filter computes the output q as a linear transform of the guide image I in a window ω_k, centered on the pixel k,

$$q_i = a_k I_i + b_k, \forall i \in \omega_k,$$

where (a_k, b_k) are the coefficients on the linear model and constant in the window ω_k. The coefficients of the linear model, (a_k, b_k), are obtained using a least-squares fit to the input noisy data, p, with the

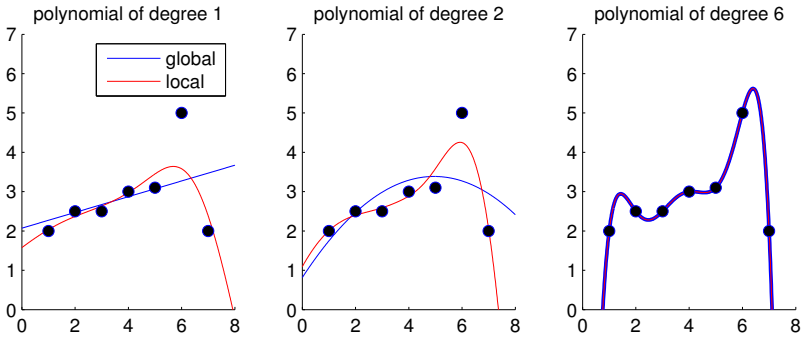


Figure 5.5: Global and local regression analysis of a small set of data points. In global regression, all data points are given the same weight when fitting the model. In local regression, data points closer to the evaluation point are given a larger weight when fitting the model. The set is fit using polynomials of increasing degrees, which allow for an increasingly closer fit of the data. Global regression highlights the constraints of each model, but even the simple line model (polynomial of degree 1) yields a smooth interpolation of the data points when using local regression. As the degree of the polynomial increases, the fit leads to very fast oscillations to accommodate the data points. This results in over-fitting of the data, which usually makes the model less useful when used to predict the output for new data points.

following cost function:

$$E(a_k, b_k) = \sum_{i \in \omega_k} ((a_k I_i + b_k - p_i)^2 + \epsilon a_k^2),$$

where ϵ is a regularization parameter used to penalize large a_k values. As ϵ increases, a_k will tend to zero, while b_k will tend to be the mean of all pixels in ω_k. Inversely, as ϵ goes to zero, a_k will tend to 1, while b_k will tend to 0. In practice, ϵ is chosen to be proportional to the variance in the noisy input p. The advantage of the linear model is that it naturally preserves the edges of the signal, since $\nabla q = a \nabla I$. The guide

image I could be the noisy image itself, but it doesn't have to be. One could use any other image as guide, and the guided image filter would then effectively filter the input image while preserving the structure of the guide. Using an alternate image as guide is particularly useful in the context of MCPT, since one can easily extract feature buffers (such as normal, depth, or texture buffers) that encode the edges of the scene. These feature buffers make for robust guide images, since they are generally much less noisy than the rendering itself. This approach was proposed by Bauszat et al.[BEM11] to efficiently filter the indirect illumination of MCPT renderings, using the normal, depth, and texture buffers as guide. Effectively, their approach estimates the filtered output as a linear combination of these three feature buffers. While their approach gives reasonable results in practice, it assumes that the rendered image can be obtained using a linear combination of the feature buffers, which is not the case for glossy and specular surfaces. In contrast, our framework makes use of a filter that assigns a different weight to each neighbor and therefore can accommodate arbitrary patterns caused by glossy or specular reflections.

5.4 Anistropic Diffusion

Isotropic diffusion solves the heat equation,

$$\frac{\delta u}{\delta t} = \alpha \Delta^2 u,$$

which relates how much a function u changes over time, $\frac{\delta u}{\delta t}$, to its spatial second derivative, $\Delta^2 u$. Since the second derivative measures the curvature of the signal, the smoothing due to diffusion is more pronounced in regions of high curvature. The parameter α models the conductance between nearby pixels, and effectively sets the rate of the diffusion, which is performed in an iterative process. It can be shown that this process is equivalent to Gaussian smoothing.

In anisotropic diffusion, proposed by Perona and Malik [PM90], the conductance parameter α is replaced by a spatially varying func-

tion $c(x, y)$,

$$\frac{\delta u}{\delta t} = \mathrm{div}(c(x, y)\nabla u) = \nabla c \cdot \nabla u + c(x, y)\Delta u,$$

where ∇u is the spatial gradient of u, and div is the divergence operator. The key is to define the conductance function, so that it prevents diffusion across edges, that is, the conductance should be small at edge locations. Perona and Malik proposed two conductance functions,

$$c(||\Delta u||) = e^{-(||\Delta u||/K)^2}$$

and

$$c(||\Delta u||) = \frac{1}{1 + \left(\frac{||\Delta u||}{K}\right)^2},$$

where K is a user parameter controlling the sensitivity to edges. The two proposed function assign a low conductance value when the spatial gradient is large, which is the case at edges. The user parameter K can be set either empirically, or, in the case of denoising, based on the level of noise in the image.

Anisotropic diffusion was used by McCool [McC99] to denoise Monte Carlo renderings, and improved the computation of the conductance map by leveraging auxiliary feature buffers (encoding the surface normals and depths) to more robustly detect edges. One inconvenience of anisotropic diffusion is that noise can only be distributed to connected neighbors. For instance, on a checkerboard surface, each tile of the checkerboard is separated from the other ones, which effectively limits the area over which smoothing can occur. In our framework, we can make use of powerful filters, such as the bilateral filter presented in Section 5.5, that can gather contributions from disconnected regions.

5.5 Bilateral Filter

The bilateral filter is a structure aware filter that was first introduced by Tomasi and Manduchi [TM98], and its key idea is to combine two kernels: a spatial kernel, that weights a neighbor contribution according to its spatial distance, and a range kernel, that weights a neighbor contribution according to its photometric similarity. The combination of the two kernels yields a generalized distance function that effectively measures the affinity between pixel pairs, and ensures that the filter will preserve edges.

We will present the original version of the bilateral filter in Section 5.5.1. In Section 5.5.2, we will present the non-local means filter proposed by Buades et al. [BCM05], a generalization of the bilateral filter with significantly increased robustness to noise. Lastly, in Section 5.5.3, we will present the concept of joint filtering, which was independently proposed by Eisemann and Durand [ED04] and Petschnigg et al. [PSA+04], and where the range kernel is evaluated on a separate guide image. In the context of MCPT, joint filtering allows to filter a rendering while preserving edges encoded in feature buffers (such as the normal, depth, or texture buffers).

5.5.1 Original Formulation

In its original formulation, the bilateral filter operates on pixel pairs. The affinity between two pixels is a function of both their spatial distance and their photometric distance, which are respectively measured using the spatial and range kernels. In principle, any kernel could be used for the spatial and range components, but in practice these are generally Gaussian kernels. Figure 5.6, reproduced from [KT09], illustrates how the bilateral filter can be used to filter a step function, and how it compares to a standard Gaussian filter. The addition of the range kernel effectively prevents the filter from crossing the edge. Using Gaussian kernels, the weights of the bilateral

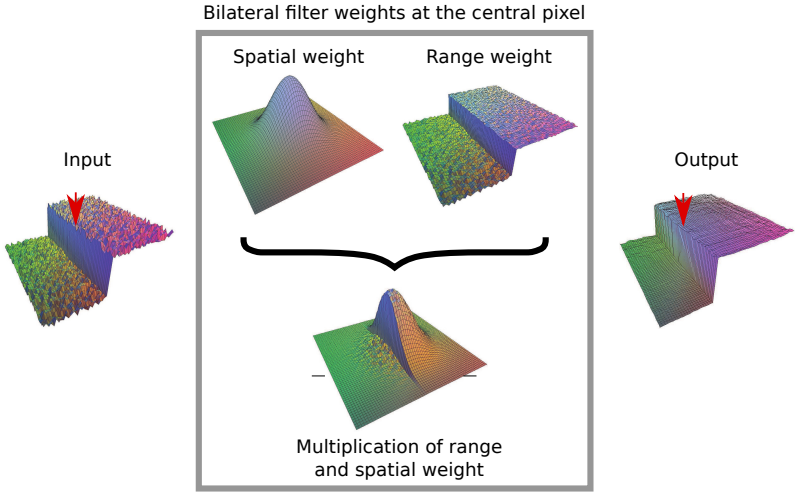


Figure 5.6: The bilateral filter is a structure aware filter that combines a spatial kernel and a range kernel. The spatial kernel gives larger weights to nearby pixels, while the range kernel gives larger weights to pixels with a similar value. The range weights encode the edge of the signal, which is therefore preserved in the filtered output. Figure reproduced from [KT09].

filter are obtained as

$$w(p,q) = \underbrace{\exp\left(-\frac{1}{2}\frac{||\mathbf{x}_p - \mathbf{x}_q||^2}{\sigma_s^2}\right)}_{\text{spatial kernel}} \cdot \underbrace{\exp\left(-\frac{1}{2}\frac{(u(p)-u(q))^2}{\sigma_r^2}\right)}_{\text{range kernel}}.$$

The bilateral filter has been used extensively in image processing. For instance, to do tone mapping [DD02], abstraction [WOG06], or style transfer [BPD06]. It was was adapted by Xu and Patanaik [XP05] to handle MCPT renderings. Xu and Patanaik observed that the bilateral filter is ill-suited to handle outliers, since those have, by definition, no affinity with their neighborhood and therefore cannot be filtered out. They propose to first filter the image using a Gaussian filter to reduce the magnitude of the outliers, and then compute the range kernel using the image with suppressed outliers as a reference. This improves the handling of outliers, at the expense of extra bias, since edges are slightly blurred when computing the range kernel. In our framework, instead of suppressing outliers, we take into account the local variance when computing the range kernel. The high variance of outliers ensures that our filtering is sufficiently aggressive to suppress them.

5.5.2 Non-Local Means Filtering

One issue when using the bilateral filter to denoise an image, is that the image noise affects the affinity measurement between pixel pairs, and therefore the weights of the filter. Because of this propagation of noise, the main characteristics of the image noise are preserved in the filtered output. While it is possible to minimize the residual noise artifacts by making the filter more aggressive, this comes at the expense of some more bias. The non-local means (NL-Means) filter, proposed by Buades et al., aims to address this issue.

The NL-Means filter builds on a key idea first explored by Efros and Leung in the context of texture synthesis [EL99]. This idea is deceptively simple and surprisingly effective: the weights of the NL-Means filter are determined by the distance between square image

patches, instead of individual pixel pairs. By averaging the distance computation over all pixel pairs of the patches, measurement noise is significantly reduced. For instance, with 7×7 patches, we average the distance computation over 49 pixel pairs, therefore reducing the variance of the distance computation by a factor 49. This significantly improves the robustness of the weight estimation and yields a much improved denoising performance. Thanks to its increased robustness, the NL-Means filter can consider pixels from a much larger region of the input image with no significant impact to the filtered output. The "non-local" in NL-Means, actually refers to this fact.

In Figure 5.7, we show the result of filtering the "barbara" image with the bilateral filter, and with the NL-Means filter using 7×7 patches. In practice, the NL-Means filtered output is significantly smoother, while still preserving the edges in the image.

input | bilateral filter | NL-Means filter with 7×7 patches

Figure 5.7: Increased robustness of the NL-Means filter. The bilateral filter weights encode some of the noise of the input, which is then visible in the filtered output. In contrast, the patch based affinity computation of the NL-Means filter eliminates most of the noise in the affinity function, leading to a much smoother output, while still preserving the edges of the image.

The NL-Means filter is at the core of the second implementation of our framework, presented in Chapter 7, and we give the details of the

filter in Section 7.1.1. We also used the NL-Means filter in the third implementation of our framework, presented in Chapter 8. This filter was also used by Moon et al. [MJL+13], to filter MCPT renderings using a virtual flash image. The virtual flash image captures many edges of the scene, including interactions with specular surfaces, and is used as a guide function to filter the noisy MCPT output. Since the virtual flash image itself can be noisy, the authors compute the weights using the NL-Means filter. The use and motivation for the NL-Means filter in our framework and Moon et al.'s framework are actually very similar.

5.5.3 Joint Filtering

Joint filtering, which is also called cross filtering, is a powerful idea that proposes to filter one image using a second one as a guide. In the context of rendering, common choices for guides are scene feature buffers, such as surface normals, texture, and depth information. The advantage of using such feature buffers as guide is that they are often noise free, or at least much less noisy than the rendered output, and therefore serve as reliable guides to detect edges.

Petschnigg et al. [PSA+04] introduced this idea in the context of bilateral filtering, calling it joint bilateral filtering. Here the range kernel is applied on the guide image, and the resulting bilateral weights are applied to the noisy input. They applied this idea to denoising photographs taken in low-light conditions, using another photograph taken with a flash as guide. Eisemann and Durand [ED04] concurently developped a very similar idea (i.e. denoising a photograph with a bilateral filter using a second photograph taken with a flash as guide), and coined the term cross-bilateral filtering.

All current state-of-the-art image space adaptive rendering methods build on the concept of joint bilateral filtering, and make use of feature buffers to guide denoising. Examples includes filters developped by Dammertz et al. [DSHL10], Sen and Darabi [SD12], and Li et al. [LWC12].

Joint bilateral filtering can be generalized trivially to joint NL-

Means filtering, and this is the basis of both Moon et al.'s method that employs a virtual flash image to filter MCPT renderings [MJL+13], as well as the latest implementation of our framework (see Chapter 8).

5.6 BM3D Filtering

The Block-Matching and 3D (BM3D) filtering algorithm by Dabov et al. [DFKE07] is the current state-of-the-art in image space filtering, and was used to great effect by Kalantari and Sen to filter MCPT renderings [KS13]. A key characteristic of the BM3D algorithm is its hybrid nature. It builds on the patch-based neighborhood search of the NL-Means filter ("block-matching" here), and combines it with wavelet thresholding. This effectively leverages the strengths of both techniques, yielding great results.

The core idea of BM3D is to group similar patches together, which are then stacked to create a relatively homogeneous 3D group. The stack is then transformed from the spatial domain into a wavelet domain, where its denoised using wavelet thresholding (using the noise standard deviation as threshold). After inverting the wavelet decomposition, we get the denoised patches in the spatial domain, that can be written back to the image. The key is that wavelet decomposition concentrates the signal energy in relatively few coefficients, and allows a better separation of signal and noise.

The wavelet decomposition is composed of two successive decompositions. The first is a 2D transform (which operates on the individual patches that constitute the stack), and the second is a 1D transform (that operates along the depth of the stack). A wide variety of decompositions could be used, but, in practice, the 2D transform used is generally the Discrete Cosine Transform (DCT), while the 1D transform along the stack depth is the Hadamard transform. The DCT, which is also used in many image and video compression schemes, decomposes the signal into a sum of cosines functions of different frequencies. The Hadamard transform is similar in spirit but operates on 1D signals.

In Figure 5.8, we compare the output of the BM3D algorithm, with the output of the NL-Means algorithm on the barbara image, and give the corresponding PSNR (Peak Signal to Noise Ratio).

input	BM3D	NL-Means
PSNR: 20.30	PSNR: 30.65	PSNR: 29.00

Figure 5.8: Impact of changing the patch size in the NL-Means filter. By setting the patch parameter $f = 0$, we emulate the behavior of the bilateral filter (i.e. affinity is measured between pixel pairs, not patches), and get a filtered output that encoded some of the input noise. In contrast, setting $f = 3$ yields a much smoother output, while still preserving the edges of the image.

When it comes to filtering MCPT renderings, BM3D suffers from two significant limitations that mitigate its qualities. The first limitation, is that there is no mechanism for leveraging auxiliary buffers (such as the normal, depth, or texture buffers) in the BM3D filter, which reduces its robustness in the presence of strong noise. The second limitation is that the BM3D filter is ill-suited for filtering spike noise (which is fairly common when rendering global illumination), since the transformation into the wavelet domain spreads the noise of each pixel to the whole 3D stack. Consequently, a single strong outlier could dominate the whole stack and lead to excessive blurring. In their method that filters MCPT renderings using BM3D, Kalantari and Sen [KS13] sidestep this second issue by operating on tone-mapped images, where the magnitude of outliers is significantly reduced, but

this comes at the expense of loss of energy. In contrast, our framework operates directly on the high dynamic range input, and considers the variance of each pixel individually, allowing to filter outliers aggressively without affecting the rest of their neighborhood.

Chapter 6

Adaptive rendering using greedy error minimization

This chapter presents our GEM algorithm, which is the first implementation of our adaptive rendering framework. The name GEM stands for Greedy Error Minimization, which our framework uses to drive adaptive sampling. This algorithm was initially developed in 2011, and the content of this chapter is reproduced from [RKZ11]. Chapters 7 and 8 will present improved implementations that rely on more advanced filtering techniques.

Monte Carlo techniques compute pixel colors by (quasi-)randomly sampling an integration domain that covers all light paths transporting light from a source to the camera. The integration domain may include effects such as depth of field, motion blur, and light paths with multiple interreflections. Unless one computes an excessive number of samples, this often leads to high pixel variance and the typical noise artifacts in Monte Carlo rendering. There are two main strategies to address this. The first is to distribute samples in an optimal fashion,

with respect to the problem at hand. The second is to smooth out noise by applying suitable filters. Both strategies can be applied in the high dimensional space of light paths or in the image plane. We focus on strategies that operate in the image plane.

We formulate the problem as follows: given a certain budget of Monte Carlo samples, obtain an image that minimizes the MSE by distributing samples in a suitable fashion in the image plane and by filtering the image with appropriate filters. We can interpret this as an optimization problem over the space of sample distributions and image filters. Our core idea is to make the problem tractable by restricting the space of filters to a discrete set of predetermined filters per pixel. Each pixel may have a different set of filters, but the set is predefined and not itself part of the optimization. We use a simple greedy strategy to obtain an approximate solution to the MSE minimization problem. Starting from an initial set of samples, we iterate over two steps. First, for each pixel we select the filter from its discrete set that minimizes the pixel MSE given the current samples. Second, given the currently chosen pixel filters, we distribute a new batch of samples that try to further reduce MSE as much as possible. This process is repeated until a termination criterion is fulfilled.

To minimize pixel MSE we express it as the sum of the squared bias, i.e., expected error, and variance. We define the set of filters at each pixel such that it provides a trade-off between reducing variance and increasing bias. Then we attempt to minimize pixel MSE by selecting the filter that offers an optimal compromise. The main challenge in practice is that we only have access to noisy data to estimate bias and variance. Therefore, an important component of our algorithm is a robust method to solve this filter selection problem.

We demonstrate and evaluate our framework using Gaussian filters at different scales as the smoothing filters. We describe simple but effective methods to select filter scales at each pixel and to distribute samples in each iteration, always attempting to minimize pixel MSE. We evaluate the performance of our approach and its robustness to noise by comparing it to images rendered using ground truth statistics, i.e., bias, variance, and MSE values. We show that our method

to minimize MSE based on noisy data comes reasonably close to the reference, and provides a significant improvement over previous adaptive sampling and reconstruction algorithms. Our method is consistent in that it converges to noise and bias free images as the number of samples increases. Bias is only guaranteed to vanish in the limit, however, as the number of samples goes to infinity. Our framework is orthogonal to MCPT on a pixel-by-pixel basis, and we implemented it on top of the PBRT renderer [PH10].

Figure 6.1 illustrates a typical result obtained using our algorithm, as well as the outputs of the intermediate steps. In particular, the top two rows of the insets show how the filtering and sampling steps are complementing each other. Indeed, regions reconstructed using small filters (dark pixels in the top row of the insets) correspond to densily sampled regions (bright pixels in the second row of the insets), and vice versa. The third row of the insets also shows how the reconstruction is progressively refined through the iterations.

In summary, we make the following contributions:

- We present a framework for adaptive sampling and reconstruction based on minimizing per-pixel MSE. In a greedy error minimization procedure, we iterate over two steps: selecting pixel filters from a set of smoothing filters to minimize pixel MSE, and distributing new samples in an attempt to maximally reduce MSE in each iteration.

- We describe an implementation of our framework with Gaussian smoothing filters at different scales. This includes robust methods to select filter scales and estimate MSE, and a strategy to distribute samples in each iteration.

- We evaluate our implementation by comparing it to results obtained using ground truth statistics. We show that in many cases, our approach comes reasonably close to ground truth.

To illustrate the efficiency of our proposed algorithm, we compare its results to those of Overbeck et al.'s Adaptive Wavelet Rendering (AWR) algorithm [ODR09], which was the state-of-the-art in image

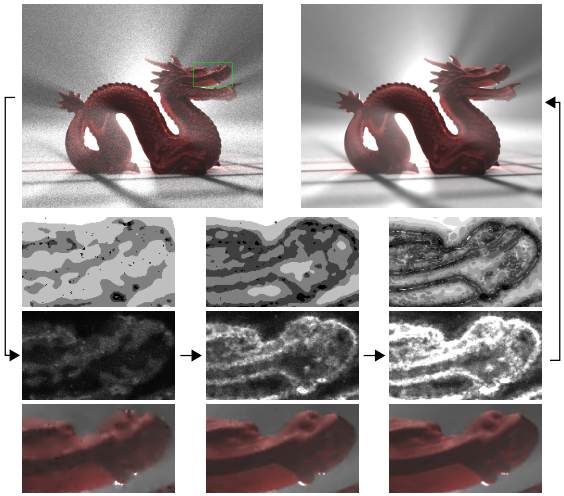


Figure 6.1: We minimize MSE in Monte Carlo rendering using adaptive sampling and reconstruction in image space. We iterate over two steps: given current samples, optimize over a set of filters at each pixel to minimize MSE; then, given a filter at each pixel, distribute more samples to further reduce MSE. Top-left: initialization with 4 samples per pixel. Insets, each column is one iteration (top to bottom): filter selection (smooth filters shaded white, sharp ones black), sample density map, reconstruction. Top-right: result at an average of 32 samples per pixel. This image features single scattering participating media, indirect illumination using photon mapping, depth of field, and area lighting.

space adaptive rendering at the time we published our own algorithm. The AWR algorithm has very similar design goals to our algorithm, which makes it a natural comparison point. Indeed, both algorithms operate in image space, couple adaptive sampling and reconstruction, target offline rendering, and aim for a general solution applicable to arbitrary scenes rendered using MCPT.

6.1 Algorithm Overview

Ideally, we would like to solve the following problem: given a certain budget of Monte Carlo samples, obtain an image that minimizes the MSE by distributing samples in a suitable fashion in the image plane and by filtering the image with appropriate filters. This problem is probably intractable in general, because the space of potential image filters is too large. A core idea is to make the problem more manageable by restricting the potential filters to a discrete set of predetermined filters per pixel. Each pixel may have a different set of filters, but the set is predefined and not itself part of the optimization.

Our framework uses a simple greedy strategy to solve the MSE minimization problem, as illustrated in Figure 6.2. Starting from an initial set of samples, we iterate over two steps. First, for each pixel we select the filter from its discrete set that minimizes the pixel MSE given the current samples. Second, given the currently chosen pixel filters, we distribute a new batch of samples that tries to further reduce MSE as much as possible under the current selection of filters. This process is repeated until a termination criterion is met, for example, a given sample budget is reached.

In the following we describe an implementation of this framework that uses the same set of filters for each pixel. In addition, the filters compute a linear function of their input, and they are related by a uniform scale. The smallest scale corresponds to the usual, unbiased pixel filter that is used in standard rendering. Selecting appropriate filters from this set allows us to minimize pixel MSE by making an optimal trade-off between bias and variance. Filters at smaller scales have little bias but much variance , and vice-versa for larger scales. We next describe the two steps of our approach in detail, i.e., filter selection (Section 6.2) and sample distribution (Section 6.3).

6.2 Filter Selection

In this section we describe how, at each pixel of a noisy image, we select a filter from our predefined set that attempts to minimize the pixel

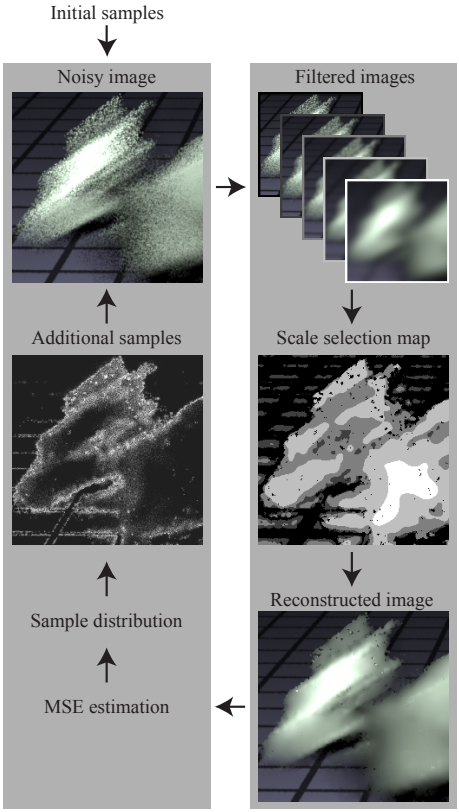


Figure 6.2: Overview of our framework. We iterate over two steps to minimize MSE: filter selection based on the noisy image on the right, and distribution of new samples to greedily reduce MSE on the left.

MSE, that is, the sum of the squared bias and the variance. In Section 6.2.1 we formulate an incremental MSE minimization strategy that avoids explicit bias estimation. We then describe in Section 6.2.2 how we implement this strategy under the assumption that the true

image is locally a quadratic function. In Section 6.2.3 we analyze the behavior of the resulting filter selector under noisy input, and finally we describe a post-processing approach to greatly reduce erroneous filter choices in Section 6.2.4.

6.2.1 Incremental MSE Minimization

In theory we could estimate bias and variance for each filter directly to minimize MSE. It is very challenging, however, to guess the true bias from noisy input. Hence, we avoid explicit bias estimation. Directly estimating the variance is less problematic, because it does not require knowledge of the true pixel value. Considering that the filters in our set are related by uniform scaling, we can order them in a fine to coarse, or sharp to smooth, manner according to their scales. A key observation is that, for most pixels, the filter bias increases and the variance decreases monotonically as we go from finer to coarser scales. Assuming monotonicity, we find the filter with minimum MSE simply by traversing the list of filters from fine to coarse. At each pixel, for each pair of consecutive fine and coarse filters f and c, we compute the change in MSE, $\Delta\mathrm{MSE}\left[f \to c\right]$, and stop when it is positive. Note that the difference between the MSE of the fine and coarse scales is

$$\begin{aligned}\Delta\mathrm{MSE}\left[f \to c\right] &= \mathrm{MSE}\left[c\right] - \mathrm{MSE}\left[f\right] \\ &= \underbrace{\mathrm{Bias}\left[c\right]^2 - \mathrm{Bias}\left[f\right]^2}_{\text{bias term}} + \underbrace{\mathrm{Var}\left[c\right] - \mathrm{Var}\left[f\right]}_{\text{variance term}},\end{aligned}$$

consisting of a *bias* and *variance* term. The crucial benefit of this approach is that we can well approximate the bias term without knowledge of the true bias $\mathrm{Bias}\left[c\right]$ and $\mathrm{Bias}\left[f\right]$, as we will show in Section 6.2.2.

We compare MSE minimization using exhaustive search over the filters to MSE minimization based on the assumption of monotonicity of bias and variance in Figure 6.3(a) and (b). We use a filter set consisting of five scales of Gaussian filters at dyadic intervals. For the sake of this comparison, we computed ground truth per-pixel bias and

(a) Exhaustive (b) Incremental (c) Our approx.

Figure 6.3: Results of scale selection using (a) minimization of the MSE using exhaustive search, (b) the incremental approach, (c) our incremental approach with bias computation using the quadratic approximation. We used five scales, where white indicates the coarsest and black the finest scale.

variance for each filter by empirically gathering ground truth statistics (of course, in practice ground truth statistics are not available; see Section 6.2.3). We obtain ground truth bias by rendering a reference image and, for each filter, computing the difference between the reference and filtered image. We obtain per-pixel variance by rendering a noisy image with a limited number of 16 samples per pixel many times. We then applied the filters to each noisy image and computed empirical pixel variance over all filtered images. In the figure we visualize the selected scales for both methods and observe that they agree for 99.7% of the pixels.

6.2.2 Quadratic Approximation

We now show how we can compute the bias term without knowledge of the true pixel value. Let us assume that the true image is a quadratic function within the support of the coarse and fine filters f and c at each pixel. In addition, we require the filters to have vanishing first central moments. We denote the scale of the coarse and fine filter by r_c and r_f. It is straightforward to show that in this case, there is a

simple relation between the bias of the filters based on their relative scales [Sil86],

$$\mathrm{Bias}\,[c] = \frac{r_c^2}{r_f^2}\mathrm{Bias}\,[f]\,.$$

Let us denote the true value of the image by ξ, and, by slight abuse of terminology, the filtered pixel value using the coarse filter by c and the value using the fine filter by f. Then $\mathrm{Bias}\,[c] = c - \xi$ and $\mathrm{Bias}\,[f] = f - \xi$. Using the above relation between $\mathrm{Bias}\,[c]$ and $\mathrm{Bias}\,[f]$ we get two equations in two unknowns, and we can eliminate ξ. After some more algebraic manipulation, we find that we can express $\mathrm{Bias}\,[c]^2 - \mathrm{Bias}\,[f]^2$ in terms of $c - f$ as

$$\mathrm{Bias}\,[c]^2 - \mathrm{Bias}\,[f]^2 \approx \frac{r_f^2 + r_c^2}{r_f^2 - r_c^2}(c - f)^2,$$

where the approximation is exact for quadratic image regions. Using this approximation, we get the following expression for the change in MSE,

$$\mathcal{S} \approx \underbrace{\frac{r_f^2 + r_c^2}{r_f^2 - r_c^2}(c - f)^2}_{\text{approximate bias term}} + \underbrace{\mathrm{Var}\,[c] - \mathrm{Var}\,[f]}_{\text{variance term}}. \tag{6.1}$$

We call this our *scale selector* $\mathcal{S}$ using an approximate bias term. If the scale selector is positive, we select the fine scale f; otherwise, we proceed to the next pair of coarser scales.

We compare MSE minimization using our scale selector $\mathcal{S}$ to the two previous methods in Figure 6.3(c). Similar as above, we use filtered reference images to evaluate the bias term and empirically established ground truth pixel variances. For 82.8% of pixels our method agrees with ground truth scale selection in (a), indicating that the quadratic approximation is valid for most pixels. Results for the other scenes of Figures 6.13 to 6.17 are given in Table 6.1.

Table 6.1: Percentage of pixels in agreement with scale selection obtained using exhaustive search, for both the incremental approach (Inc.) and our approximation (Our) for scenes of Figures 6.13 to 6.17.

Scene	Inc.	Our	Scene	Inc.	Our
killeroos	97.7%	92.8%	toasters	98.7%	91.9%
plants-dusk	95.5%	86.0%	yeahright	98.2%	87.8%

6.2.3 Estimation from Noisy Data

Of course in practice, we do not have access to ground truth data for the filtered pixel values, c and f, and their variances, $\mathrm{Var}\,[c]$ and $\mathrm{Var}\,[f]$. Instead, we need to estimate them from the noisy data available, i.e., the Monte Carlo samples that we acquired so far in the iterative procedure. We denote these samples by $s_i, i = \{1 \dots k\}$. We now express the filtered pixel values and their variances directly using the Monte Carlo samples. Again, our equations are for an individual pixel.

A filtered pixel value for, e.g., the fine filter is simply the weighted average

$$f = \sum_{i \in \{1 \dots k\}} w_i^f s_i,$$

where w_i^f are the filter weights for each sample s_i. The expression for the coarse filter is analogous. The pixel variance is

$$\mathrm{Var}\,[f] = \sum_{i \in \{1 \dots k\}} (w_i^f)^2 \mathrm{Var}\,[s_i]\,, \tag{6.2}$$

where we use the squared filter weights, because the Monte Carlo samples are supposed to be uncorrelated. Again, the expression for $\mathrm{Var}\,[c]$ is analogous. Unfortunately, $\mathrm{Var}\,[f]$ and $\mathrm{Var}\,[c]$ rely on the variances of the Monte Carlo samples $\mathrm{Var}\,[s_i]$, which are not known. Therefore, for each s_i we use the empirical variance over all samples that are

in the same pixel (i.e., square pixel region) as s_i. Let us denote this subset of samples by P. Then

$$\mathrm{Var}\left[s_i\right] \approx \frac{1}{|P|-1} \sum_{j \in \{P\}} (s_j - \bar{s})^2, \tag{6.3}$$

where $\bar{s}$ is the mean of the samples in P, and $|P|$ is the number of samples in the pixel.

We have now expressed our scale selector $\mathcal{S}$ in Equation 6.1 directly using the Monte Carlo samples. Interpreting the Monte Carlo samples as random variables, $\mathcal{S}$ is itself a random variable. This opens up the possibility to analyze the behavior of $\mathcal{S}$ under given assumptions about the random samples. Unfortunately, even for normally distributed s_i, its density is highly complex and not easily amenable to analytical investigation.[1]

Empirical Analysis. We analyze the probability density of the scale selector $\mathcal{S}$ using an empirical experiment. We use independent and identically distributed samples s_i that sample a constant 1D function with value zero and additive noise, for simplicity. The 1D function consists of one million "pixels". We use Gaussian filters for f and c with a fixed relative scale of two. The free input parameters of our experiment are:

- The noise variance of the random samples s_i.
- A radius r for the fine filter, which determines the weights w_i^c and w_i^f.
- The number of samples per pixel $|P|$ used to estimate empirical sample variance $\mathrm{Var}\left[s_i\right]$.

[1] For normally distributed Monte Carlo samples s_i, their empirical variance follows a chi-square distribution. The variance of the filters $\mathrm{Var}\left[c\right]$ and $\mathrm{Var}\left[f\right]$ is then a weighted sum of random variables with chi-square distribution. The difference $\mathrm{Var}\left[c\right] - \mathrm{Var}\left[f\right]$ would be a convolution of the densities of $\mathrm{Var}\left[c\right]$ and $\mathrm{Var}\left[f\right]$ if these were independent, but since they share the same Monte Carlo samples, they are not independent and the distribution of their difference is more complicated, etc.

For this experiment, the ideal decision of the scale selector $\mathcal{S}$ is to always select the coarse scale (i.e., $\mathcal{S}$ is negative), since the true bias for all filters is zero and going to a coarser scale reduces variance. Because of the noisy s_i, however, the scale selector will occasionally make a wrong decision. We call the probability that the selector is positive and makes a wrong decision the *error rate* of the selector. Our goal now is twofold: First, to understand how the error rate is related to the parameters of the experiment; and second, to derive a single user specified parameter that directly controls the error rate, independent of the input parameters.

Before discussing the results of the experiment, we point out that the bias and variance terms in the scale selector are independent random variables, because a weighted sum of samples (as in the bias term) is independent from the empirical variance of the samples (as in the variance term). Therefore, the probability density of the scale selector is a convolution of the densities of the bias and the variance term, and we can study these separately. In Figure 6.4 we plot the histograms of the bias and variance terms and the histogram for the scale selector. The histograms are collected from the one million 1D "pixels" in our experiment. The error rate is simply the area under the histogram of the scale selector over the positive part of the horizontal axis, indicated by the area shaded in gray. We now summarize the observations we made in our experiment by varying the input parameters.

Sample Variance. The error rate remains constant under different variances of the random samples s_i. This is because the bias and variance terms are both proportional to the variance of the samples. Consequently, the scale selector, which is the sum of the two, is scaled as well. But scaling a random variable does not change the probability that it is larger (or smaller) than zero, hence the error rate remains constant.

Filter Radius. The error rate depends weakly on the filter radius r. Larger scales lead to slightly lower error rates. We observed that

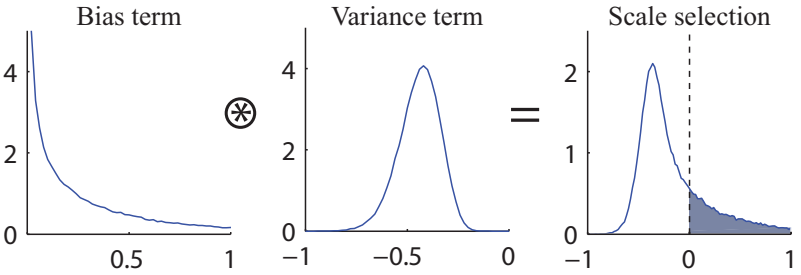


Figure 6.4: Histograms of bias, variance, and scale selector $\mathcal{S}$, which is the convolution of the former two. The horizontal axes indicate the values of the random variables, and the vertical axes the relative frequencies of the corresponding values. The parameters are: normally distributed samples with variance 10, number of samples per pixel $|P| = 32$, and a filter scale factor $r = 1$.

the bias term is scaled inversely proportional to r. The variance term, however, is not scaled exactly the same. Instead, the density of the variance term becomes slightly sharper with larger radii as illustrated in Figure 6.5(a). Because the error rate is based on the convolution of the densities of the bias and variance terms, it will be slightly reduced for larger filters due to the sharpening of the variance term.

Number of Samples per Pixel. The error rate remains largely constant under different numbers of samples per pixel $|P|$, except for low sample counts, where the error rate increases. We observed that the bias term is again scaled inverse proportionally to $|P|$. The variance term is scaled similarly, but for low sample counts the shape of its density is significantly different as visualized in Figure 6.5(b). We have observed that this mismatch leads to significantly higher error rates for low sample counts.

Intuitive Parameterization. Based on the above observations, we introduce an intuitive user specified parameter γ that allows the user

to directly indicate the desired error rate for constant inputs. This can be achieved easily by weighting (i.e., scaling) the bias and variance terms relative to each other. We apply a weight consisting of two factors ρ and $z(\gamma)$ to the bias term. The factor ρ compensates for low sample counts, while $z(\gamma)$ controls the error rate. Both are empirically determined. We found that a factor $\rho = (1 - 1/|P|)$ works well in practice to compensate for the effect of low sample counts. Using experimentation, we determined the factor $z(\gamma)$ such that the user parameter γ approximately achieves the desired error rate. We manually found an appropriate mapping to be $z(\gamma) = -\log(1 - (1.9\gamma)^{(1/\sqrt{2})})$ valid in the range $\gamma \in 0 < \gamma < 0.4$. Error rates above 0.4 are not interesting in practice. An alternative would be to tabulate the relation between γ and the error rate to provide an intuitive user parameter.

6.2.4 Post-Processing the Filter Selection

Controlling the error rate is useful to adjust the trade-off between bias and variance, but any non-zero error rate will produce a given percentage of wrong filter selection decisions that will be noticeable as spikes in the reconstructed image. Therefore, we post-process the filter selections to remove these outliers. We illustrate the process in Figure 6.6, where we filter a 1D signal consisting of two boxes with uniform noise, evaluated at 250 1D pixels. Each pixel received 32 samples that we used to estimate empirical pixel variance. Our filter set consists of Gaussian filters at eight scales related by scaling factors of $\sqrt{2}$. Figure 6.6(a) depicts the noisy input signal and the 8 filtered inputs. We show the reconstructed signal in Figure 6.6(b).

We represent the results of our scale selectors as binary *stopping maps* for each pair of scales at each pixel. This is shown as one row per filter pair in Figure 6.6(c). A value of 1 (shaded gray) means that we should stop and use the fine scale, and 0 (black) indicates that going to the coarse scale is estimated to reduce MSE. An important observation is that outliers appear as isolated clusters of zeros or ones, where the size of the clusters is related to the size of the pair of filter scales. Hence, we can remove outliers by additionally filtering

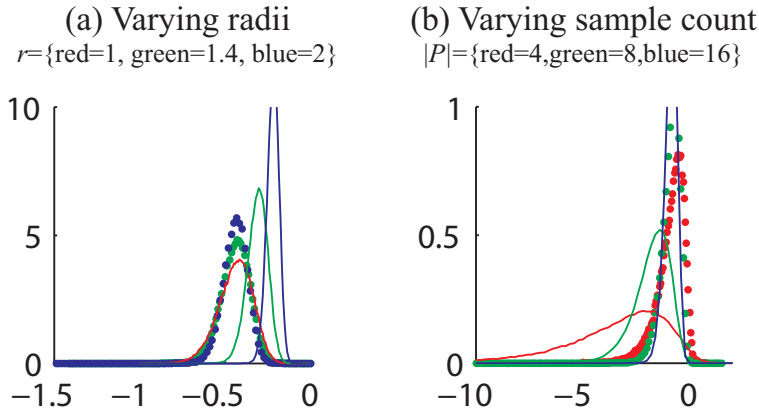
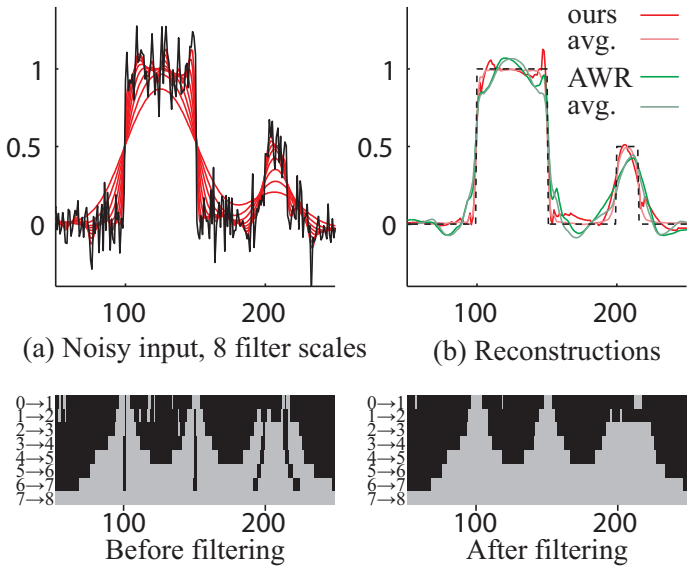


Figure 6.5: Histograms for the variance term under varying (a) radii r and (b) sample count $|P|$. In (a) we illustrate that the variance distribution is not inversely proportional to the radii. We scale the distributions for radii $r = 1.4$ and $r = 2$ with factors 1.4 and 2 to match the distribution of $r = 1$. We depict the result with dotted lines, showing that the variance distribution becomes slightly sharper for larger filters. In (b) we provide a similar visualization for different sample counts $|P|$. We scale the histograms for $|P| = 4$ and $|P| = 8$ with factors $1/4$ and $1/2$ to match the histogram for $|P| = 16$. The results (dotted lines) show that the scaled distributions are less sharp than the true distribution.

each map (row), where we choose the size of the outlier removal filter according to the size of the pair of scales. In practice we obtained good results with Gaussian filters of the same size as the coarse scale. Then we round the filtered map back to binary. In practice, we ignore the center pixel of the outlier filter, because this dominates the result too much for small outlier removal filters. A more thorough investigation to determine optimal outlier removal filters is an interesting topic for future research. The final filter selection at each pixel is the finest scale with value 1 (gray in the figure). This process works well in



(a) Noisy input, 8 filter scales (b) Reconstructions

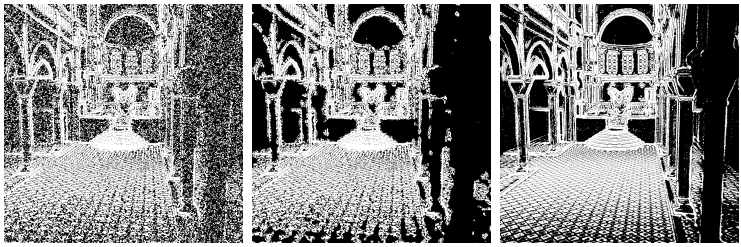
(c) Stopping maps for each pair of scales, fine to coarse from 0-7

Figure 6.6: Illustration of outlier removal during post-processing: (a) noisy input with 8 filter scales, (b) reconstructions (including average over 200 runs) of our approach and wavelet soft thresholding, (c) binary stopping maps for each pair of scales.

practice because the scale selector guarantees a low outlier rate.

Figure 6.6 also includes a comparison to soft wavelet thresholding using Daubechies wavelets as used by Overbeck et al. [ODR09]. For both methods we show the reconstruction of the noisy signal on the left, and the average reconstruction by generating the noisy signal 200 times. The average per-pixel MSE for wavelet thresholding is 7.6×10^{-3}, and for our method it is 2.5×10^{-3}. The average reconstruction from Wavelet thresholding also contains ringing artifacts, which stem from aliasing in the wavelet decomposition.

Figure 6.7 visualizes postprocessing using a 2D example, where



Before filtering After filtering Reference

Figure 6.7: Filtering a binary stopping map from scale 2 to 3 for the "sibenik" scene. Left: before outlier removal; middle: after post-processing; right: reference from ground truth statistics.

we show the stopping map from scale 2 to 3. The input data uses 32 noisy samples per pixel. We remove most outliers without losing any features from the unfiltered map. In Figures 6.8 and 6.9 we compare the performance of our complete filter selection procedure to selection maps obtained by minimizing MSE according to empirical ground truth statistics collected from a large number of samples, similar as in Figure 6.3. We use five filters at dyadic scales. The coarsest scale is visualized in white, the finest one in black. Our maps were computed using a uniform sample distribution of 32 samples per pixel. We lose some detail compared to ground truth, but reliably retain the main features. All maps were obtained by setting the error rate parameter γ to 0.2 and using outlier removal.

6.3 Sample Distribution

The goal of the sample distribution is to place new samples in the image plane, such that the *relative* MSE given the current per-pixel filter selection is reduced as much as possible. The idea is to select the m pixels, whose relative MSE can be improved the most by distributing n new samples over the support of their selected filters.

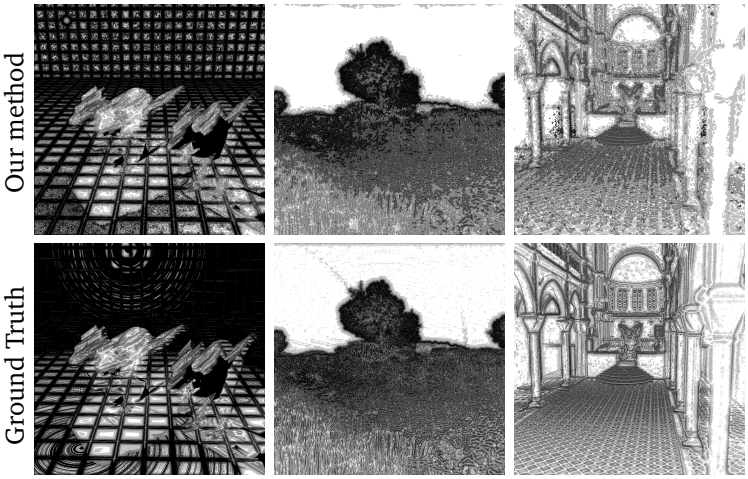


Figure 6.8: Selection maps for five scales using our method (top) and ground truth statistics (bottom). Our method selects filters using statistics from 32 samples per pixel, while ground truth statistics are collected from a large number of samples. We show final renderings in Figures 6.13 to 6.15.

We estimate the MSE of a selected filter based on the same approach as in Section 6.2. We simply accumulate the estimated MSE differences until we reach the selected filter. The MSE of the finest scale is estimated as its variance, since the finest scale is considered unbiased as it uses the pixel filter requested by the user. We compute relative MSE by dividing the estimated MSE by the squared value of the selected scale plus a small constant $\epsilon = 0.001$ to prevent overemphasizing very dark image areas. Because (relative) MSE is inversely proportional to the number of samples contributing to a filter, adding n samples over the support of a filter that received a total of n_s samples from previous iterations reduces (relative) MSE by a factor $n_s/(n+n_s)$. Therefore, after some algebraic manipulation we find that the reduction in relative MSE after adding n samples is

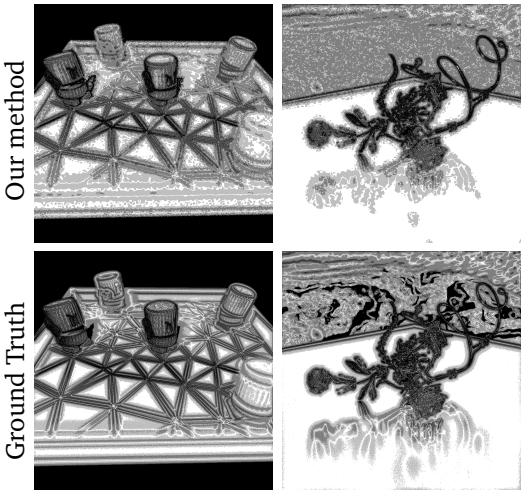


Figure 6.9: Selection maps for five scales using our method (top) and ground truth statistics (bottom). Our method selects filters using statistics from 32 samples per pixel, while ground truth statistics are collected from a large number of samples. We show final renderings in Figures 6.16 to 6.17.

$(\text{relative MSE}) \cdot n/(n + n_s)$.

We maintain a priority queue of pixels according to their potential error reduction. In each iteration, we retrieve m pixels from the queue and distribute samples over their filters. The assumption behind our computation of potential MSE improvement is that the filters of these pixels do not overlap, but we do not enforce this. We randomly select n pixels in the support of each filter, where we use importance sampling according to the filter weights. We then draw one (or more, if the pixel was chosen several times) additional sample for each of the selected pixels. We maintain the total number of samples per pixel to be able to compute the potential MSE improvement as required above.

Figure 6.10 visualizes the sample distribution generated by our algorithm using an error rate of $\gamma = 0.2$ and with an average of 32 samples per pixel. We compare our sample map to a "ground truth map", which is obtained using the same algorithm but with ground truth statistics for MSE estimation, as for the comparison in Figures 6.3, 6.8 and 6.9. We also include the distribution obtained from AWR in the comparison. We observe that our approach exhibits more adaptivity. The AWR algorithm also produces density peaks aligned with the subsampled wavelet grids. In contrast, our method operates at full resolution for all filter scales, which leads to a smoother sample distribution.

6.4 Implementation

We now detail some important aspects of our implementation: computation of filter scales and their bias and variances, image reconstruction from a sub-pixel grid to accommodate non-uniform sample distributions, additional details concerning final reconstruction, and a brief description how we integrate our method in PBRT.

Computation of Scales and Their Statistics. Computing filter scales and their variances directly using the Monte Carlo samples as described in Section 6.2.3 would be expensive in terms of computation and storage. Instead, we store only the finest filter scale and the mean of the empirical sample variances in each pixel and update them in each iteration. Updating the finest filter scale is straightforward. Since variance decreases linearly with the number of samples, using Equation 6.3 the mean variance of pixel p is

$$\mathrm{Var}\,[p] \approx \frac{1}{|P|}\frac{1}{|P|-1}\sum_{j\in\{P\}}(s_i-\bar{s})^2.$$

We update pixel mean variances from one iteration step to the next by maintaining the necessary terms separately.

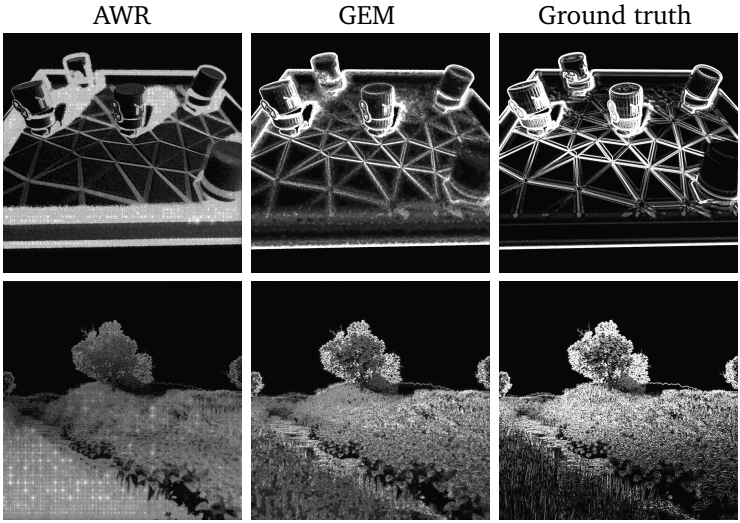


Figure 6.10: Sample densities with an average of 32 samples per pixel obtained with the AWR algorithm, our scale selection method, and scale selection using ground truth statistics. The AWR method samples smooth regions more densely. Our algorithm (GEM) better detects smooth regions and distributes more samples in the high-frequency regions, approximating the ground truth distribution more closely.

In each iteration we compute the coarser scales and their variances by further filtering the finest scale and the mean pixel variances. For the variances at a given scale we filter the initial pixel mean variances $\mathrm{Var}\,[p]$ using the *squared* version of that scale filter. This method is valid if the initial pixel mean variances are uncorrelated [ODR09]. Since we estimate these using samples landing within each individual square pixel, i.e., the estimates do not share any samples, there is no correlation as required.

In our implementation we use Gaussian filters and their squares. Since both are separable, filtering is efficient. We do not perform any

local updates but simply filter the complete finest scale and the pixel mean variances after each iteration. As an important detail, our implementation also allows us to use an arbitrary pixel filter employed by the renderer as our finest scale, while only the coarser scales are Gaussian filtered. We compute the bias term for pairs of coarser scales as described in Section 6.2.2. To compute the bias term for the transition from the pixel filter to the first Gaussian filtered scale, note that the pixel filter is unbiased by definition. Therefore, the bias term here is simply the squared difference between the pixel filter and the first Gaussian filtered scale. In practice, we use the pixel filter employed by the renderer as the finest scale, in addition to four Gaussian filters at dyadic scales.

Filtering Non-Uniform Sample Distributions. In our sample density maps, we have sharp and significant changes across image edges. Therefore, adjacent pixels may receive significantly different numbers of samples, which could severely bias the reconstruction if not handled correctly. Following the work of Mitchell [Mit87], we store samples on a subpixel grid. Each individual subpixel value is obtained using a subpixel box filter, which is valid provided the sample distribution is uniform over the subpixel. We use a subpixel grid resolution of 4×4. At low sample counts, the subpixel grid has many holes that we must fill. We use a simple two-step pull-push strategy, where we fill empty sub-pixels with the mean value of the samples in the whole pixel. To minimize the number of holes in the subpixel grid, we distribute images samples using scrambled low-discrepancy sequences [KK02b]. Other dimensions are still sampled using pure uniform random distributions. We then apply our filters on the subpixel resolution.

Final Reconstruction. To optimize final image quality we use a slightly modified filter selection procedure in the last iteration of our procedure (Figure 6.2). First, we use eight Gaussian filters related by a factor of $\sqrt{2}$, as opposed to four filters related by a factor of 2 during the previous iterations. The standard deviation of the finest scale is 1.0 during the adaptive phase and $\sqrt{2}$ during the final reconstruc-

tion. The increased number of scales reduces seams that may appear at transitions between scales in the final reconstruction. We found that using more scales during the adaptive process does not reduce error, but comes at a performance penalty. Second, we use a larger Gaussian filter for filtering the binary stopping map during the final reconstruction. While we use the coarser scale Gaussian filter during the adaptive phase, we double this size for the final reconstruction. Using smaller filters during the adaptive process better samples clusters of outliers, while using larger filters during the final reconstruction ensures we smooth out most of the remaining outliers. Third, when filtering the binary stopping maps we use the filtered binary value only if it leads to selecting the coarser scale. By disregarding filtered binary values that switch to stopping at the finer scale we suppress spike noise more effectively.

Integration with PBRT. We have implemented our algorithm using PBRT to demonstrate its compatibility with standard Monte Carlo ray tracers. Our implementation consists of a scale selector and an adaptive sampler. The adaptive sampler implements the existing PBRT *Sampler* interface. It has two states: "initialization" and "adaptive". Rendering starts with the "initialization" phase, where each pixel is sampled uniformly using 4 samples per pixel. After initial sampling, the scale selector first computes the filtered scales and the corresponding variances as described above. It then uses the scale selector from Section 6.2.3 and the post-processing algorithm from Section 6.2.4 to determine the scale for each pixel that minimizes the MSE. During the "adaptive" phase of the sampler, all pixels are first ranked according to their estimated improvement in relative MSE, which is established as described in Section 6.3. We then select the m pixels with the highest estimated improvement. In our implementation, the user specifies a desired average number of samples per pixel n that should be distributed over the iterations. We performed 8 iterations in all our results. Hence, for an image composed of M pixels, we distribute $N = M(n-4)/8$ samples per iteration, over $m = N/n$ pixels, and each selected pixel receives n samples. The n samples are

distributed over the selected filter scale as described in Section 6.3.

6.5 Results

We generated all results on a dual 4-cores XEON system at 2.50GHz, with 8GB of RAM, using 8 threads. We implemented both our approach and the AWR algorithm on top of PBRT [PH10]. For our method, we set the error rate parameter γ to 0.1 for all scenes. For the AWR algorithm, we use the Daubechies 9/7 wavelets, a smoothing constant $c_s = 1$, and renormalization factor of 1.05 as suggested in the original paper. Due to the limited amount of samples drawn per iteration with the AWR algorithm (from 64 to 2048), our implementation scales poorly to 8 cores, which dramatically decreases its performance.

We present results from five test scenes using a wide range of effects, geometries and materials. The "killeroos" scene showcases the impact of motion blur. The "plants-dusk" scene uses environment lighting with a very complex geometry. The "yeahright" scene shows glossy materials, a model with fine geometric details, environment lighting, and one-bounce indirect illumination. The "plants-dusk", "toaster" and "sibenik" scenes were used in the original AWR paper [ODR09]. We modified the "sibenik" scene to have an environment light (seen by refraction through the windows) and a single area light over the gargoyle.

We compare the following four methods:

- NAIVE: Uniform sample distribution and filter using finest kernel. This is the default PBRT behavior.
- AWR: Adaptive sample distribution and reconstruction using wavelet coefficient shrinkage [ODR09].
- GEM: Our proposed greedy error minimization approach with adaptive sampling and filter selection.
- GEM-GRD: Our proposed greedy error minimization approach with adaptive sampling and filter selection, but with ground

truth bias and variance values. This is to illustrate the best results we could theoretically achieve with our technique.

In Figure 6.11 we illustrate the impact of γ, the single user parameter of our method, on the "sibenik" scene. Intuitively, γ controls the error rate for filter selection in uniform areas. Lower γ values produces smoother results in uniform areas, but they tend to blur across edges. Larger γ values preserve edges better, but exhibit artifacts due to outliers in uniform areas. The experiment shows that the MSE remains relatively constant for a wide range of γ values from about 0.20 to 0.30, meaning that scale selection and outlier removal operate effectively. For lower γ values, scale selection tends to pick filters that are too smooth. For higher values, the error rate in uniform areas becomes too large, such that we are not able to remove outliers robustly any more.

In Figure 6.12 we report on the convergence of the four methods measured in terms of relative MSE to a reference image produced with PBRT. We compute relative pixel MSE as $(img-ref)^2/(ref^2+\epsilon)$, and we report the average error over the images. We set $\epsilon = 0.01$ to prevent over-weighting of errors in very dark regions. Our algorithm explicitly attempts to minimize this error, which AWR does not. We acknowledge that relative mean squared error is not a perfect measure for image quality, but we believe it is still a useful indicator for the convergence rate of a method.

For all scenes, our method consistently improves upon both the NAIVE and AWR approaches. The AWR method usually improves the MSE at lower samples counts, however converges slowly to the right solution. Several factors could cause this behavior. First, the functional variance estimate in AWR only considers the maximum and minimum sample values within a pixel. This essentially maximizes the impact of outliers, both bright and dark. Another source of error are ringing effects in dark regions, emphasized by the relative error metric. Finally, using the standard parameters for AWR tends to overblur images, leading to smooth results, but not necessarily low numerical error. This is apparent as AWR has low error initially for scenes with large uniform, but noisy areas ("killeroos", "sibenik"). In contrast, the

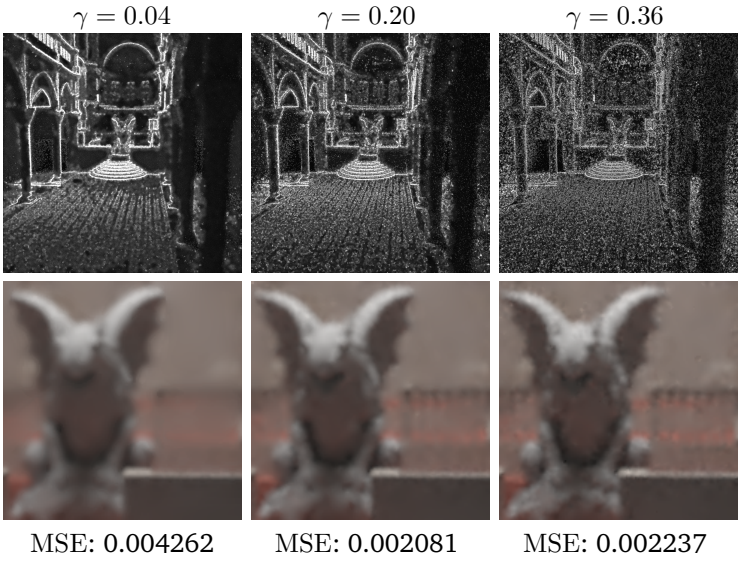


Figure 6.11: Top: sample densities obtained for the "sibenik" scene by varying γ. Low values yield a better reconstruction of smooth regions, which leaves more samples to resolve edges. High values yields more uniform sample densities, since outliers are draining more samples.

error remains high for the "plants-dusk" scene, which contains a lot of high frequency details.

Figure 6.12 evaluates the performance of our and AWR's adaptive filtering techniques used with standard uniform sampling. In the "killeroos" and "plants-dusk" scenes, uniform sampling performs significantly worse than adaptive sampling. Adaptive sampling only slightly improves the MSE of the "sibenik" scene, despite the apparent adaptivity of our sample distribution (see Figure 6.11). Even with reference statistics we observe a similar behavior. This may be because most of the error is due to variance in uniform regions, which is filtered effectively without adaptive sampling.

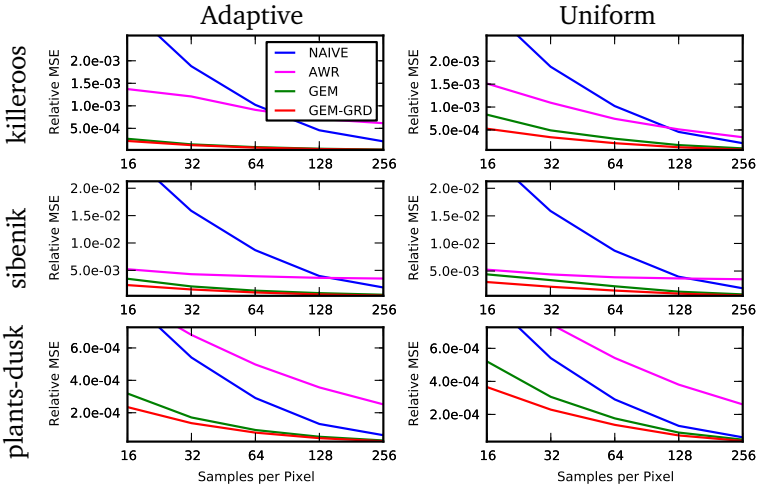


Figure 6.12: Convergence plots over average number of samples per pixel measured in average per-pixel MSE. Left: adaptively distributed samples; right: uniformly distributed samples with adaptive reconstruction only. The NAIVE method is shown as a reference. AWR is the algorithm proposed by Overbeck et al. [ODR09], GEM is our proposed approach, and GEM-GRD is our approach using ground truth statistics. For scenes with a combination of noisy areas and high frequency details, such as the "killeroos" and the "plants-dusk" scenes, adaptive sampling provides a significant improvement. For the "sibenik" scene, where noise is equally distributed across the image and there are large uniform areas, adaptive sampling yields only a slight improvement, even with the reference statistics. AWR behaves similarly, but has consistently lower convergence rates.

In Figures 6.13 to 6.17 we present results obtained with the naive approach, the AWR algorithm, our method and the ground truth adaptive result obtained by minimizing the ground truth MSE. Results obtained with the AWR algorithm frequently show ringing artifacts

caused by aliasing in the Daubechies wavelet decomposition. Ringing is most evident at the edge between the animal ear and the background in the "killeroos" scene (Figure 6.13). AWR tends to oversmooth images with many details, which is best visible in the "plants-dusk" scene (Figure 6.14). The algorithm performs extremely well in uniform regions, which are reconstructed virtually noise free.

Our method gives high quality results both visually and numerically, offering a good compromise between noise and sharpness. We filter more conservatively across edges retaining most high frequency information, but still manage to smooth uniform regions effectively. The "toaster" scene is a good example: we maintain the sharp edges of the model, but have slightly noisy smooth shadow transition on the ground. For all five scenes, our method gives results which are visually close to the solution obtained using ground truth statistics.

We evaluated the performance overhead of our method with the Google CPU Profiler, since this tool supports multi-threaded applications. For our simplest scene ("toasters"), we spend 4.36% of rendering time in our adaptive sampler and filter selection algorithms. Since our algorithm is independent of scene complexity, this is indicative of a worst case scenario. If measured in terms of samples rendered per second, there is often a larger discrepancy between our adaptive rendering performance and the uniform one. We believe the reason for this is that adaptive sampling tends to generate more "difficult" samples in average than uniform sampling. For instance, for the "killeroos" scene, our adaptive sampler targets the animals ($\sim 33K$ primitives each) much more than the ground and back wall, which yields a correspondingly larger cost per sample.

Discussion and Limitations. Our algorithm relies heavily on the estimated variance to guide the adaptive sampling scheme. In tests using two samples per pixel in the initialization phase, we obtained very similar results to those reported in Figure 12 (an improvement of up to 3% of the MSE, depending on the scene), illustrating our algorithm robustness to noise in the initial variance estimate. Regardless, our algorithm has difficulties handling regions where light paths are

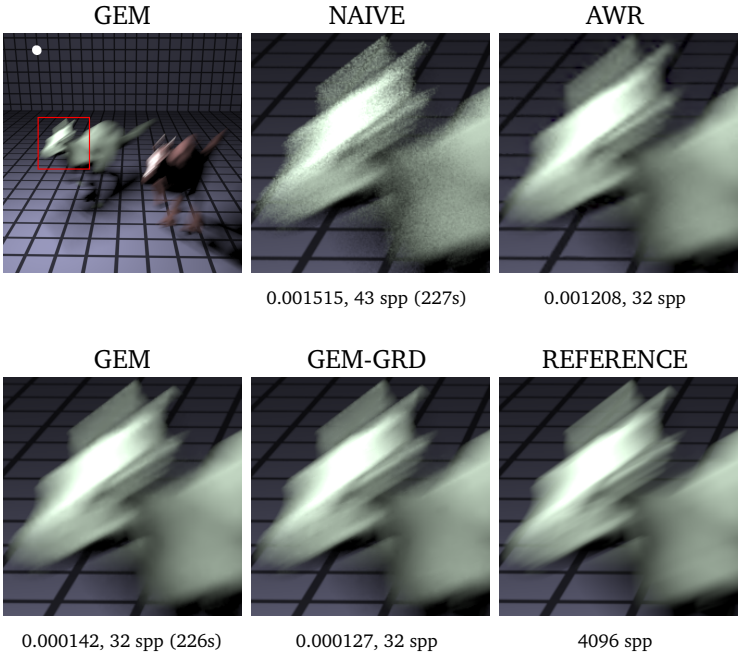


Figure 6.13: Result for the "killeroos" scene at 1024×1024 pixels. We indicate MSE and rendering times in seconds. Timing data for the AWR is omitted since our implementation does not scale to multiple CPUs. We adjusted the number of samples in the "NAIVE" method to match the rendering times of our "GEM" method, providing an equal time comparison.

unlikely to be found by brute force sampling. There, it tends to systematically underestimate the variance, which leads to undersampling and filtering artifacts. The dark regions of the "sibenik" scene, where more than 95% of the samples carry a null radiance, present such a case. Our method also cannot reconcile the need to filter overlapping elements in the same pixel using different kernels. The "killeroos"

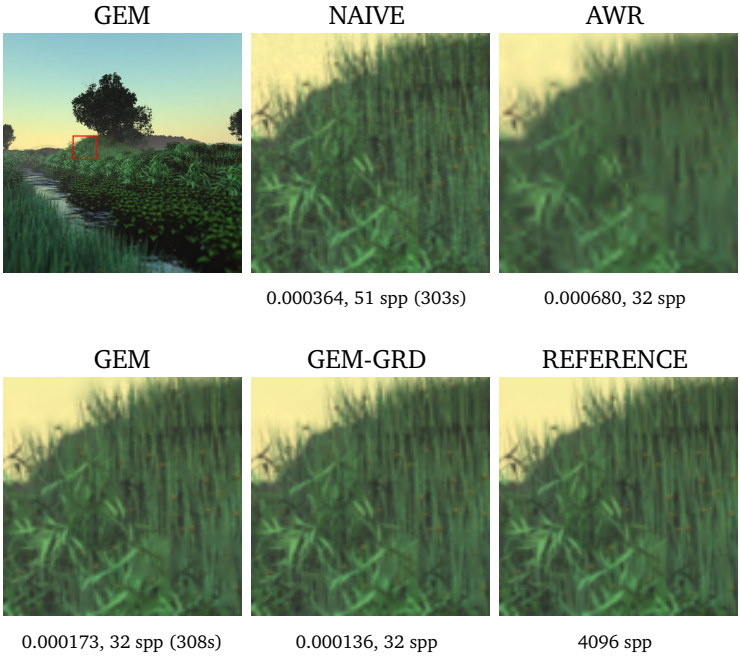


Figure 6.14: Result for the "plants-dusk" scene at 1024×1024 pixels. We indicate MSE and rendering times in seconds. Timing data for the AWR is omitted since our implementation does not scale to multiple CPUs. We adjusted the number of samples in the "NAIVE" method to match the rendering times of our "GEM" method, providing an equal time comparison.

scene features sharp lines which illustrate this fact (Figure 6.13). One can still faintly see the lines through the blurred animal head in the 4096 spp rendering, while they are mostly blurred out in our result. Also, the soft shadow cast by the animal is filtered heavily, except around the sharp lines, resulting in an increased noise level in our reconstruction. In other words, given contradictory filtering require-

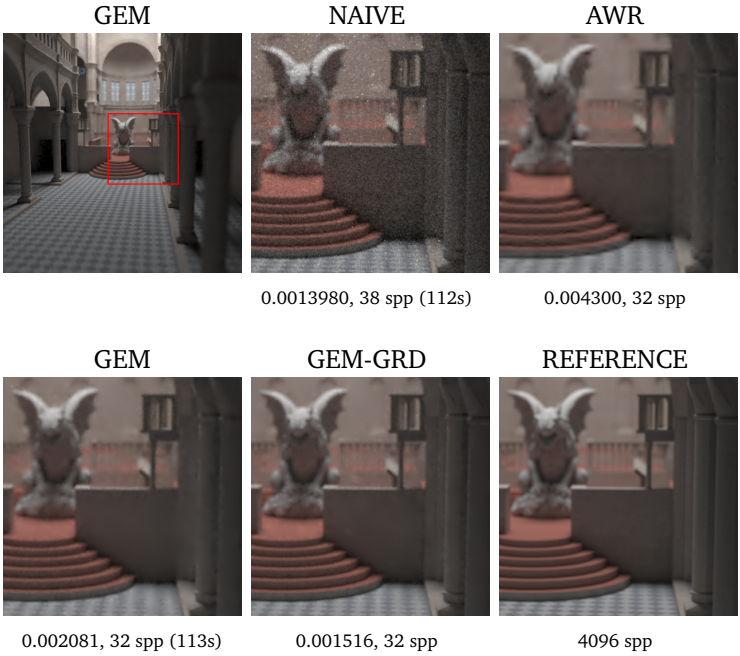


Figure 6.15: Result for the "sibenik" scene at 1024×1024 pixels. We indicate MSE and rendering times in seconds. Timing data for the AWR is omitted since our implementation does not scale to multiple CPUs. We adjusted the number of samples in the "NAIVE" method to match the rendering times of our "GEM" method, providing an equal time comparison.

ments, our algorithm preserves the most prominent feature. Nonetheless, the adaptive process of our method assigns more samples to these difficult regions, mitigating the problem.

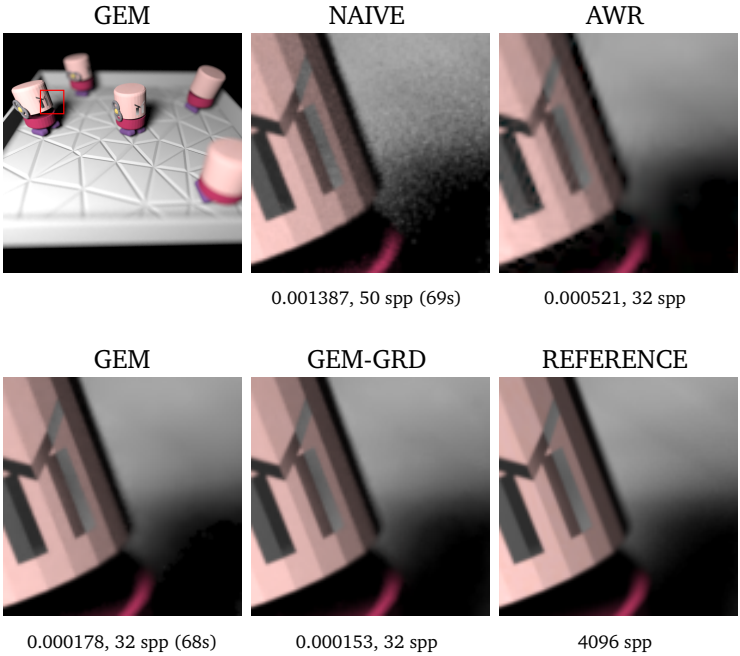


Figure 6.16: Result for the "toasters" scene at 1024×1024 pixels. We indicate MSE and rendering times in seconds. Timing data for the AWR is omitted since our implementation does not scale to multiple CPUs. We adjusted the number of samples in the "NAIVE" method to match the rendering times of our "GEM" method, providing an equal time comparison.

6.6 Conclusion

In this chapter, we described a versatile adaptive sampling and reconstruction algorithm that greedily minimizes MSE in Monte Carlo rendering. The method provides significant improvement in terms of numerical error and image quality over previous work. A key compo-

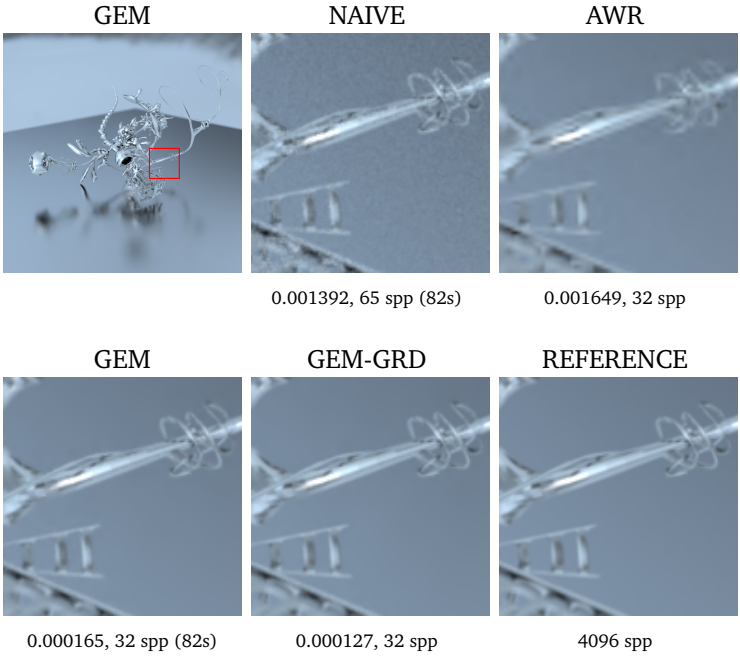


Figure 6.17: Result for the "yeahright" scene at 1024×1024 pixels. We indicate MSE and rendering times in seconds. Timing data for the AWR is omitted since our implementation does not scale to multiple CPUs. We adjusted the number of samples in the "NAIVE" method to match the rendering times of our "GEM" method, providing an equal time comparison.

nent is a robust filter selection procedure that minimizes pixel MSE over a set of discrete filters. A main limitation of our approach is that, similar as AWR, our variance estimation assumes that the Monte Carlo renderer generates random samples. As a consequence, we will overestimate variances for low discrepancy sequences or stratified samples. One could heuristically reweight our variance term to account for this,

but a more thorough solution would be desirable. A weakness of our approach is that we cannot filter noise close to edges because we only use isotropic filters.

When this work was initially published, we noted that an interesting avenue for improvement would be to extend our approach to more general filters, such as cross-bilateral kernels. This has been done by Li et al. in their paper titled SURE-based Optimization for Adaptive Sampling and Reconstruction [LWC12], where they use a filterbank of cross-bilateral filters with varying support.

In the next chapter, we'll present an improved implementation of our adaptive framework that addresses the two main shortcomings of the GEM algorithm, that is, the restriction to pure random samples and to isotropic Gaussian filters.

Chapter 7

Adaptive rendering using non-local means filtering

This chapter presents our NLM algorithm, which is the second implementation of our adaptive framework. The name NLM refers to the fact that this implementation makes use of the NL-Means filter [BCM05] to denoise renderings. It builds upon the same image space iterative approach as our previous GEM algorithm (see Chapter 6), but addresses its two main shortcomings: the use of a filterbank of isotropic Gaussian filters, which prevented efficient filtering along edges, and the restriction to pure random sampling, whereas modern renderers use low-discrepancy sampling techniques that can sometimes substantially reduce the variance. Our NLM algorithm was initially developed in 2012, and the content of this chapter is reproduced from [RKZ12].

The algorithm we develop in this chapter builds on the observation that previous image space adaptive rendering methods, such as our GEM algorithm (see Chapter 6) and Overbeck et al.'s AWR algorithm [ODR09], are based on image denoising techniques that are not competitive with the state of the art in image processing. For example, the AWR algorithm uses straightforward wavelet shrinkage,

and our GEM algorithm is based on adaptive bandwidth selection with isotropic Gaussian filters. Both techniques leave ample room for improvement, as we show in this chapter.

The main contribution of our approach is to extend an image denoising filter that is competitive with the state of the art in image processing and employ it in an adaptive rendering framework. We show that our approach is more robust under severe noise than previous techniques, significantly reducing numerical error and visual artifacts. In addition, our approach is more effective at removing noise even in complex image regions while minimizing the smoothing of image features. Finally, our technique is compatible with efficient low discrepancy sampling while previous techniques assumed random sampling, which affords additional improvements in output quality. Our technique shares the advantages of other image based approaches. It can deal with arbitrary light transport and lens effects, and it only requires the Monte Carlo samples as its input such that it is straightforward to implement it on top of existing renderers.

Our framework builds on an iterative strategy consisting of three components in each iteration step. First, we distribute a given budget of Monte Carlo samples over the image. We sample the image uniformly in the initial iteration step, while consecutive iterations employ adaptive sampling. Second, we filter the image to reduce noise. Finally, we estimate the error remaining in the filtered image to drive adaptive sampling in the next iteration step. The effectiveness of this scheme largely hinges on the denoising filter. A core idea in our approach is to extend and adapt the non-local means filter [BCM05], which is competitive with the state of the art in image denoising, to our adaptive rendering framework. A challenge in applying such a filter for adaptive rendering is the need to obtain an error estimate to drive adaptive sampling. We address this issue using a dual-buffer strategy: we simply split the samples into two sets that we render and filter in two separate image buffers. The difference between the filtered buffers serves as an effective error estimate. Hence we demonstrate that it is possible to employ sophisticated denoising filters for adaptive rendering, and our results show significant improvements

over previous work. In summary we make the following contributions:

- An adaptive rendering framework based on non-local means filtering. We demonstrate significant improvements in numerical error and visual quality compared to previous work.
- A dual-buffer strategy for non-local means filtering. This allows us to obtain error estimates to drive adaptive sampling, and to extend our method to low-discrepancy sampling.
- Extensions of the non-local means filter to adapt it to denoise images produced by Monte Carlo rendering.

7.1 Algorithm Overview

The objective of our algorithm is to optimize rendering quality given a user specified sampling budget. Figure 7.1 gives a visual overview of our method. We build on image space adaptive sampling and filtering using an iterative scheme, where each iteration is composed of three steps: (1) sampling, (2) filtering, and (3) estimating the residual error. In each subsequent iteration we use the error estimate from the previous one to drive adaptive sampling, where we distribute a predetermined fraction of the sample budget each time. We bootstrap the algorithm in the first iteration by sampling the image uniformly. A key component of our approach is the denoising filter in step two. We use a variant of the NL-Means filter tailored to our adaptive rendering framework. During the iteration we keep track of per-pixel statistics including sample count and empirical variance, in addition to the usual pixel value. We use these statistics to adapt the NL-Means filter to local image characteristics. In addition, we operate on two image buffers (indicated in red and green in Figure 7.1), instead of one, each receiving half of the samples. This is a key idea of our method that allows us to improve filtering quality and obtain simple but effective error estimates to drive adaptive sampling.

We next describe the main components of our algorithm. We review the original NL-Means formulation and introduce our extensions

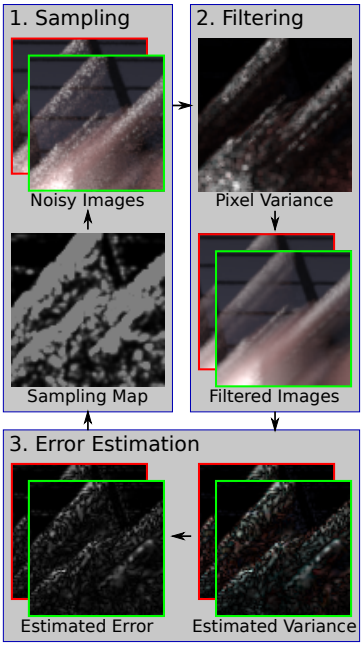


Figure 7.1: Overview of our dual-buffer framework. We indicate the two buffers with red and green borders. We iterate over three steps: sampling, filtering, and error estimation. In the filtering step, we first estimate the pixel variances using the noisy buffers. Then we denoise the buffers with our variant of NL-Means filtering. In the error estimation step, we first estimate the residual variance in the filtered buffers, and then derive the potential error reduction afforded by placing an additional sample per pixel. In the sampling step, we distribute a set of samples according to the estimated errors in each buffer. We iterate until the sample budget is exhausted.

in Section 7.1.1. Section 7.1.2 covers our error estimation technique, and Section 7.1.3 describes our adaptive sampling scheme.

7.1.1 The NL-Means Filter

The NL-Means filter [BCM05] is a non-linear, edge-preserving filter that computes each output pixel as a weighted sum of input pixels. The set of input pixels contributing to one output pixel may originate from a large region in the input image, hence the term *non-local*. A key feature of the NL-Means filter is that the weights are determined by the distance between small image patches, as illustrated in Figure 7.2. The NL-Means filter is a generalization of the bilateral filter [TM98], which considers distances between pairs of pixel values, instead of small patches, to compute filter weights. This extension leads to much improved denoising performance, and NL-Means and its variants are among the most successful denoising algorithms.

We propose several key extensions to the original filter formulation that allow us to use it effectively in an adaptive rendering framework:

- dual-buffered filtering (Section 7.1.1),
- support for non-uniform variance (Section 7.1.1),
- symmetric distance computation to better handle gradients (Section 7.1.1).

We start by giving a definition of the original NL-Means filter, and then detail our extensions and their impact on the filtering process.

The NL-Means filter computes the filtered value $\hat{u}(p)$ of a pixel p in a color image $u = (u_1, u_2, u_3)$ as a weighted average of pixels in the square neighborhood of size $2r + 1 \times 2r + 1$ centered on p, as illustrated in Figure 7.2,

$$\hat{u}_i(p) = \frac{1}{C(p)} \sum_{q \in N(p)} u_i(q) w(p, q), \tag{7.1}$$

where $N(p)$ is the square neighborhood centered on p, $w(p, q)$ is the weight of the contribution of q to p, i is the index of the color channel, and $C(p)$ is a normalization factor,

$$C(p) = \sum_{q \in N(p)} w(p, q).$$

The weight $w(p,q)$ of a neighbor q is based on the distance between a pair of small patches of size $2f+1 \times 2f+1$ centered at p and q. The patch distance $d^2(P(p),P(q))$ is the average of the per-pixel and per color channel squared distances $d_i^2(p,q)$ over the patches,

$$d_i^2(p,q) = (u_i(p) - u_i(q))^2, \tag{7.2}$$

$$d^2(P(p),P(q)) = \frac{1}{3(2f+1)^2} \sum_{i=1}^{3} \sum_{n \in P(0)} d_i^2(p+n, q+n), \tag{7.3}$$

where $P(p)$ and $P(q)$ are the patches centered on p and q, $P(0)$ represents the offsets to each pixel within a patch.

An important observation is that, because the input signal is noisy, the measured squared distances are biased. Therefore, the original NL-Means filter [BCM05] subtracts the variance of the measured squared distances to cancel out the noise contribution from the patch distance. Assuming uniform pixel noise with variance σ^2 and uncorrelated pixels p and q, the modified patch distance is

$$max(0, d^2(P(p),P(q)) - 2\sigma^2).$$

The weight $w(p,q)$ of the contribution of pixel q to p is then obtained using an exponential kernel,

$$w(p,q) = \exp^{-\frac{max(0,d^2(P(p),P(q))-2\sigma^2)}{k^2 2\sigma^2}},$$

where k is a user specified damping factor that controls the strength of the filter. A lower k value yields a more conservative filter.

We also use the patchwise extension that was proposed by Buades et al. [BCM05], which produces slightly smoother outputs. Instead of weighting only the pixel p at the center of the patch with the weight $w(p,q)$, we weight all pixels in the patch centered at p with $w(p,q)$. Each pair of pixels occurs in $2f+1 \times 2f+1$ patches, each time with a distinct weight $w(p+n,q+n)$, where n is the offset of p and q in the patch. In the patchwise implementation the final weight $W(p,q)$ for a pair of pixels is simply the average over all weights that involve

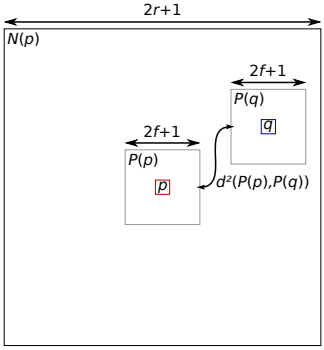


Figure 7.2: NL-Means computes the filtered value $\hat{u}(p)$ of pixel p as a weighted average of all pixels q in a square neighborhood of size $2r+1 \times 2r+1$. The weight between p and q is based on the squared distance $d^2(P(p), P(q))$ between the pair of patches $P(p)$ and $P(q)$ of size $2f+1 \times 2f+1$ centered on p and q.

these two pixels,

$$W(p,q) = \frac{1}{(2f+1)^2} \sum_{n \in P(0)} w(p+n, q+n).$$

Dual-Buffer Filtering

Dual-buffered filtering is our key modification to NL-Means that enables most of our other extensions and allows us to employ it in an adaptive rendering framework. We observe that in the original NL-Means approach the filter weights and the input signal are correlated. This makes it challenging to analyze the error introduced by the filter. In addition, since the filter weights not only adapt to the signal but also to the noise, some noise tends to be preserved in the output. We address both issues with our dual-buffered implementation. We maintain two image buffers, A and B, that receive half the samples in each iteration of our framework (Figure 7.1). We also store pixel

statistics including the number of samples and the empirical sample variance separately in each buffer. We then use the two buffers to cross filter each other, that is, we use the filter weights computed from buffer A to filter buffer B, and vice versa. The final output is the average of the two filtered buffers. Our approach eliminates the correlation between the filter weights and the noise, and we get much improved filtering as shown in Figure 7.3. In addition, the per-pixel differences between the buffers serve as a basis for our variance and error estimation components that we describe next.

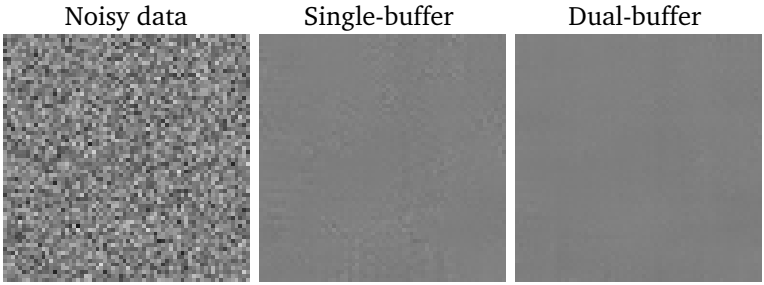


Figure 7.3: Filtering a noisy uniform input (left), using the single-buffer approach (center), or our dual-buffer approach (right). Using a single buffer tends to preserve structures from the noisy input because of the correlation between filter weights and input. The filter parameters are $r = 10$, $f = 3$ pixels, and $k = 0.45$. We list the mean squared error (MSE) under each image.

Non-Uniform Variance

While the original NL-Means formulation assumes uniform noise over input images, Monte Carlo rendering generally leads to highly non-uniform noise patterns. The type of noise and its magnitude depend on the scene geometry, object materials, and light transport and lens effects. Therefore, let us denote per-pixel variance of color channel i at

pixel p by $\mathrm{Var}_i[p]$, replacing the uniform variance σ from Section 7.1.1. We modify the previous per-pixel squared distance computation by performing variance cancellation and normalization on a per-pixel basis,

$$d_i^2(p,q) = \frac{(u_i(p) - u_i(q))^2 - \alpha(\mathrm{Var}_i[p] + \mathrm{Var}_i[q,p])}{\epsilon + k^2(\mathrm{Var}_i[p] + \mathrm{Var}_i[q])}, \tag{7.4}$$

where α controls the strength of variance cancellation. In addition, we define $\mathrm{Var}_i[q,p] = min(\mathrm{Var}_i[q], \mathrm{Var}_i[p])$ to clamp the variance at position q to the variance at p. This ensures that the potentially large variance of brighter neighbors does not cancel out the measured squared difference, and it prevents bright regions from blurring into dark ones. Note that with $\mathrm{Var}_i[p] = \mathrm{Var}_i[q] = \sigma^2$ we fall back to the original definition. We set $\epsilon = 10^{-10}$ to prevent divisions by 0. The value of ϵ must be very small in order to preserve features in dark areas. Finally, the patch distance is computed as in Equation 7.3, and the weight $w(p,q)$ of the contribution of pixel q to p is now simply

$$w(p,q) = \exp^{-max(0, d^2(P(p),P(q)))}. \tag{7.5}$$

Additionally, we set to zero weights $W(p,q)$ below a threshold of 0.05, to help preserve fine features in noisy areas where non-feature neighbors can greatly outnumber feature neighbors, dominating the result even with very low weights.

Variance Estimation. Estimating the pixel variance $\mathrm{Var}\,[p]$ is a key component of our algorithm (we omit the index i of the color channel for better readability from now on). Assuming that samples are drawn from a random distribution, we could estimate the pixel variance using the empirical variance of the samples contributing to the pixel, which we denote by $\Sigma[p]$, as in our GEM algorithm (see Section 6.4). Random sampling, however, may lead to significantly higher variance than more sophisticated sampling strategies such as low-discrepancy sequences [KK02b]. Therefore, we develop a simple approach to support low-discrepancy sampling in our framework, taking

advantage of our two buffers. The same idea was developed by Sijbers et al. [SDDVA+98] to estimate the variance in magnetic resonance images, but assuming uniform variance across the image.

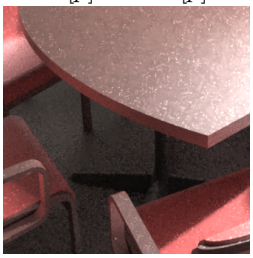
We obtain an initial estimate $\Delta[p]$ of the pixel variance $\mathrm{Var}\,[p]$ using the squared difference between the two buffers, $\Delta[p] = (A(p) - B(p))^2/2$. This is an unbiased estimate, but it is rather noisy because it is based on just two samples, that is, the two buffers. We found that we can improve our results, as illustrated in Figure 7.4, by applying an additional smoothing step to the initial variance estimates $\Delta[p]$. The intuition behind our smoothing approach is that the empirical sample variances $\Sigma[p]$ have a structure similar to the actual pixel variances, but $\Sigma[p]$ may be strongly biased. Therefore, we smooth the initial estimates $\Delta[p]$ by cross filtering them with the empirical sample variances $\Sigma[p]$, that is, we compute NL-Means filter weights using $\Sigma[p]$ and apply them to filter $\Delta[p]$. We illustrate the process in Figure 7.5.

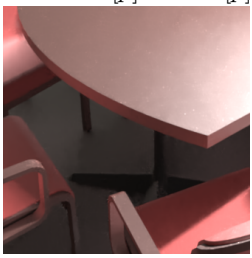
To obtain the filter weights we also need the variance of $\Sigma[p]$, which we compute as the difference between $\Sigma[p]$ from buffers A and B. We compute pixel distances as in Equation 7.4, but using Σ and its variance estimate. Because the variance estimate for Σ is noisy, we set $\alpha = 4$ in Equation 7.4 to discard all differences lower than two standard deviations. We use a small neighborhood with $r = 1$, a patch size of $f = 3$, and $k = 0.45$. The small neighborhood ensures that the variance peaks, which are aligned with the outliers pixel values, are preserved. Since the filtering tends to increase the variance estimate of pixels neighboring noisy regions, we also clamp the filtered value of $\Delta[p]$ so that it does not exceed $\Sigma[p]$.

Symmetric Distance

Our symmetric distance computation is designed to improve the filtering of smooth gradients. Because the original NL-Means formulation tends to constrain the filter to neighbors orthogonal to the gradient direction, as we illustrate in Figure 7.6, it often results in distracting artifacts in otherwise smooth gradients. Our extension builds on the observation that, in a smooth gradient, all pixel differences are radi-

$\Delta[p]$ as $\mathrm{Var}\,[p]$ Filtered $\Delta[p]$ as $\mathrm{Var}\,[p]$





MSE: 9.742E-3 MSE: 1.809E-3

Figure 7.4: Filtering with low-discrepancy samples. In the left image, we directly used the squared difference $\Delta[p]$ between our two buffers as an estimate of the pixel variance $\mathrm{Var}\,[p]$ in the NL-Means filter. In the right image we cross-filtered $\Delta[p]$ using the sample variance $\Sigma[p]$ to obtain a smoother estimate for $\mathrm{Var}\,[p]$. The unfiltered buffer variance $\Delta[p]$ is too noisy to reliably estimate pixel distances, whereas our filtered variance yields a smooth output.

ally symmetric with respect to the center pixel. The error introduced by the contribution of a given neighbor is canceled by the contribution of the radially symmetric neighbor. We exploit radial symmetry in the signal by defining a modified distance to a pair of symmetric neighbors q_1 and q_2,

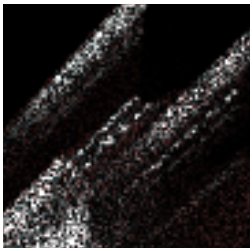
$$d_i^2(p,\bar{q}) = \frac{(u_i(p) - u_i(\bar{q}))^2 - (\mathrm{Var}_i[p] + \mathrm{Var}_i[p,\bar{q}])}{\mathrm{Var}_i[p] + \mathrm{Var}_i[\bar{q}]}, \tag{7.6}$$

where

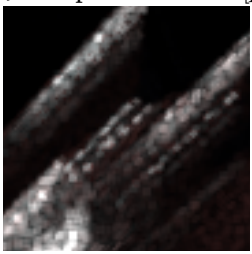
$$\begin{aligned} u_i(\bar{q}) &= (u_i(q_1) + u_i(q_2))/2, \\ \mathrm{Var}_i[\bar{q}] &= (\mathrm{Var}_i[q_1] + \mathrm{Var}_i[q_2])/4, \\ \mathrm{Var}_i[p,\bar{q}] &= (\mathrm{Var}_i[p,q_1] + \mathrm{Var}_i[p,q_2])/4. \end{aligned}$$

Note that $\mathrm{Var}_i[\bar{q}]$ is half the mean variance of pixels q_1 and q_2, since the effective sampling rate doubles when we compute $u_i(\bar{q})$. Using

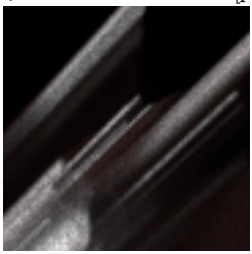
(a) Sample variance $\Sigma[p]$



(b) Buffer variance $\Delta[p]$



(c) Filtered $\Delta[p]$, used as $\mathrm{Var}\,[p]$



(d) Reference

Figure 7.5: Estimating $\mathrm{Var}\,[p]$. We use the sample variance $\Sigma[p]$ (a) to cross-filter the inter-buffer variance $\Delta[p]$ (b), an unbiased but very noisy estimate of the true variance. The resulting filtered variance (c) is our estimate for $\mathrm{Var}\,[p]$. It is a reasonable approximation of the reference variance (d) as computed over 100 images.

these squared symmetric distances we compute the corresponding symmetric weight $w(p, \bar{q})$ as before.

In practice we always compute the asymmetric distances to both q_1 and q_2 as in Equation 7.4 and the corresponding weights $w(p, q_1)$ and $w(p, q_2)$, in addition to the symmetric weight $w(p, \bar{q})$. Only if the symmetric weight is larger the sum of both asymmetric weights, we set both $w(p, q_1)$ and $w(p, q_2)$ to $w(p, \bar{q})$. Otherwise we keep the asymmetric weights. This prevents the use of the symmetric weights if we do not have enough confidence in the symmetry of the data.

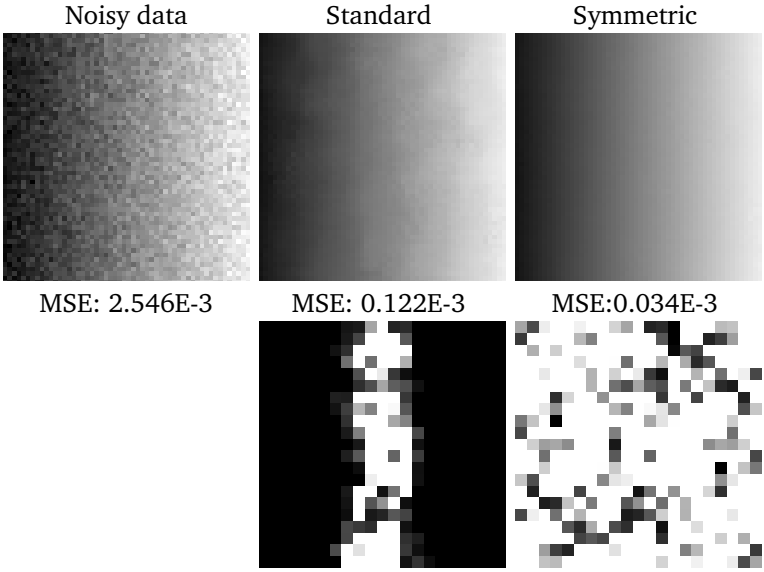


Figure 7.6: A noisy ramp (top left), filtered using the standard NL-Means filter (middle column) and our extended filter including symmetric distances (right column). The corresponding filters for the center pixel are shown in the bottom row. The standard filter is constrained to neighbors orthogonal to the gradient direction, while our extended variant has a much larger effective support. The filter parameters are $r = 10$, $f = 3$, and $k = 0.45$.

7.1.2 Error Estimation

The error in the filtered image consists of residual variance and bias introduced by the filtering. An analytical error analysis of the NL-Means filtered output, however, is rather complicated because the filter weights are both noisy and correlated with one another. Instead, we directly estimate the error as the squared relative difference between the two filtered buffers $\hat{A}$ and $\hat{B}$. We use the relative difference

to prevent overweighting differences in bright image regions. We estimate the error E separately for each buffer (for simplicity we omit the buffer index from the notation). The error of the filtered buffer $\hat{A}$ is

$$E = \frac{(\hat{A} - \hat{B})^2}{\epsilon + \hat{A}^2},$$

where $\epsilon = 10^{-3}$ is an offset to prevent divisions by 0. Note that our estimate accounts for variance but not bias in the error. We do not attempt to estimate bias because the NL-Means filter tries to avoid bias by design. In Figure 7.7 we compare our estimated error to the ground truth error, that is, the relative squared distance to the reference image. Our estimate captures the main features of the ground truth error, but may miss some small image details that are smoothed out in both buffers. These details can only be resolved with a larger number of samples per pixel.

Because we use the error estimation in the next sampling step to guide the distribution of samples, we additionally weight it according to the potential error reduction obtained by adding a single sample in each pixel. Given that the pixel variance is inversely proportional to the number of samples, adding one sample to a pixel p will decrease the variance by a factor of $1/(1 + n_p)$, where n_p is the number of samples already contributing to the filtered value of p. The new sample will also contribute to the pixels in the neighborhood of p, whose filter weights include p. Consequently, the error weight of an additional sample in p is

$$W(p) = \frac{\sum_{N(p)} w(p, q)}{1 + n_p},$$

and the weighted error of p is simply $W(p)E(p)$.

7.1.3 Sampling

We allocate an equal fraction of the total sample budget to each iteration. We split the number of samples in each iteration evenly over the two image buffers and distribute samples according to the same desired per-pixel sample density in each buffer. We represent the desired

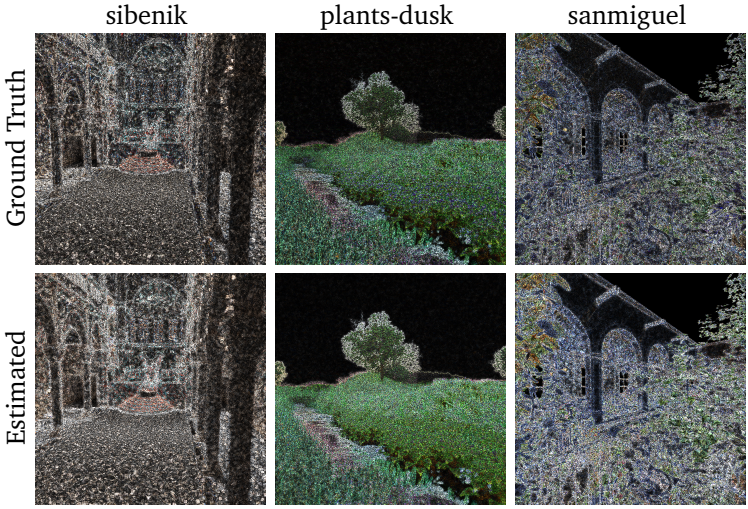


Figure 7.7: Comparison of our error estimation using the relative squared difference between the two filtered buffers and the relative squared difference to the reference image, using uniform distributions of 32 samples per pixel.

sample density using a sampling map that specifies for each pixel the number of additional samples to draw.

For the initial iteration we use a uniform sampling density. For the subsequent iterations we obtain the sampling map as follows. We sum the weighted error maps of both buffers, and smooth it using a small Gaussian kernel with $\sigma = 0.8$. This smoothing step allows pixels that missed a rare event (for instance a fast moving object), to still be sampled if their neighbors recorded the event, effectively filling holes in our sampling map. We then normalize the sampling map to sum up to $N/2$, where N is the sample budget per iteration, which gives us the number of samples to be drawn per pixel. Finally, we clamp the per-pixel sample count to a predefined value to prevent pixels with very large errors from draining too many samples. For each pixel,

we use the fractional part of the sample count as the probability of drawing an additional sample. For instance, if we need to draw 3.2 samples in a pixel, we have a 80% chance of drawing 3 samples, and a 20% chance of drawing 4 samples. The rounding error is propagated from pixel to pixel, to ensure that the overall average is maintained.

We show the sample density maps obtained with our algorithm for a set of test scenes in Figure 7.8. This figure also shows ground truth sample density maps obtained using ground truth error maps, that is, the relative squared difference to a reference image. The density maps are similar in both cases, but the MSE is lower when driving the sampling using the ground truth error. We presume this is because the ground truth error captures the bias introduced by the filtering, while our estimation does not.

7.2 Implementation

We integrated our algorithm in the PBRT [PH10] framework. We implemented a dual-buffered film interface, along with a new sampler and denoiser. The renderer of PBRT was also modified to perform sampling over multiple iterations. We accelerate NL-Means filtering by exploiting the fact that all operations on pixel patches amount to averaging per-pixel values over a patch. This averaging could be performed in constant time using summed area tables [LWC+08], or in linear time using separable box filters. We use a box filter for numerical stability. Consequently, the cost of our implementation is mostly driven by the filter window size, while the patch size has a negligible impact. This brings the cost of the filter in line with that of a bilateral filter. Given the embarrassingly parallel nature of the filter, we ported it to CUDA to further improve performance. We suffer some overhead due to data transfer between the CPU and the GPU at each iteration, but this could be eliminated by using the NVIDIA OptiX [PBD+10] framework instead of PBRT to do the raytracing.

Our weighted error estimation, which we use to drive the adaptive sampling, assumes that the bias introduced by our filter is negligible.

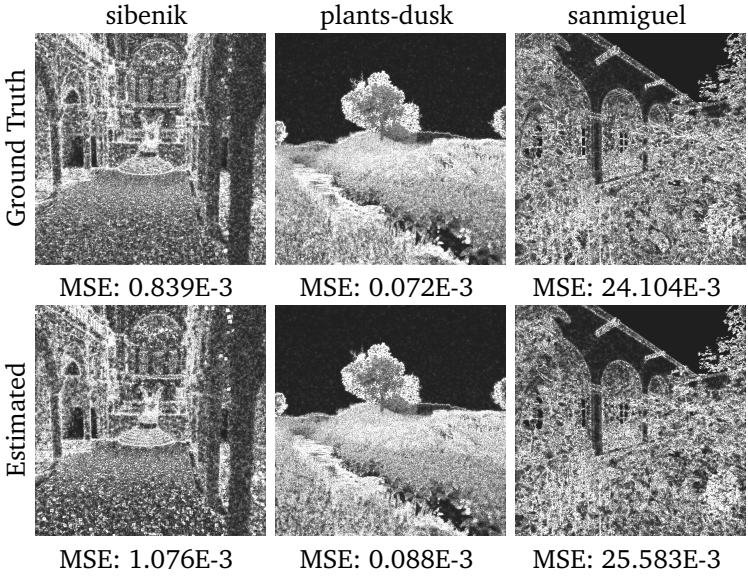


Figure 7.8: Sampling density maps with 32 samples per pixel on average. The top row shows maps obtained by driving the sampling with the squared difference to the reference image. The bottom row shows maps obtained with our estimated error. We list MSE values of the corresponding renderings under each image.

To minimize the amount of bias, we use a very conservative filter to drive sampling, and a more aggressive one for final reconstruction. During the iteration we use parameters $r = 7$ and $\alpha = 0.5$ in Equation 7.4. For final reconstruction we use $r = 10$ and $\alpha = 1$. In both cases we use $f = 3$ and $k = 0.45$. To ensure that our reconstruction of pixel values accounts for the sharply varying sample densities across image edges, we use a subpixel grid and simple push-pull filtering to fill subpixel holes, similarly to the implementation of our GEM algorithm (see Section 6.4).

7.3 Results

We evaluate our algorithm (NLM) on a set of five test scenes and compare our results to our previous GEM algorithm (see Chapter 6), and unfiltered uniform low-discrepancy sampling (LD). Our test machine is a dual 4-core XEON system at 2.50GHz, with 8GB of RAM, and a GeForce GTX580 GPU. All three methods are implemented on top of PBRT, using 8 threads. For the GEM algorithm we use the default suggested settings. For our algorithm we use four iterations, one with uniform sampling followed by three with adaptive sampling, and the parameters mentioned in Section 7.2. Each iteration is assigned one fourth of the total sample budget. We report the relative mean squared error (MSE), but also provide the perceptually based structural similarity (SSIM) [WBSS04a] with respect to the reference in Table 7.1. The MSE is computed directly on the high-dynamic range data, while the SSIM is computed on tone-mapped images. Our tone mapping is a simple gamma correction, using $\gamma = 2.2$, followed by clamping to the range $[0, 1]$.

We provide convergence plots for standard low-discrepancy sampling with no filtering (LD), the GEM algorithm, and our own method (NLM) in Figure 7.9. For our method we provide convergence plots using both adaptive sampling and uniform sampling. Our method consistently yields the lowest MSE value. The gain of adaptive sampling depends on scene complexity. In scenes such as the "conference" scene, where noise can be filtered out effectively over large regions, the adaptive sampling offers a marked improvement. The "killeroos" scene is similar, where noise is restricted to limited regions that the adaptive sampler targets extensively. On complex scenes such as the "sanmiguel" scene, too many features need to be sampled, negating any potential gain of the adaptive sampling.

In Figure 7.10, we show the impact of the patch-based distance computation of the NL-Means filter. To isolate the impact on the filtering, we used uniform sampling, instead of adaptive sampling. By setting the patch size parameter to $f = 0$, we get the same behavior as the bilateral filter, minus the spatial component of its distance

Table 7.1: Perceptual quality, measured using the SSIM metric, for images of Figures 7.13 to 7.17. Values range from 0 to 1000, where 1000 indicates a perfect match with the reference. The SSIM metric is based on tone-mapped images using gamma correction ($\gamma = 2.2$) and clamping to the range $[0, 1]$. The LD column uses uniform low-discrepancy sampling with no filtering. The GEM column uses the method described in Chapter 6 with adaptive random sampling. The NLM column uses our method with adaptive low-discrepancy sampling.

Scene	LD		GEM		NLM	
	Full	Inset	Full	Inset	Full	Inset
killines	961	853	993	956	997	987
plants-dusk	971	962	990	912	996	975
sibenik	650	650	935	912	969	956
conference	523	416	914	877	970	963
sanmiguel20	755	532	844	793	891	863

computation. The resulting image has significantly more residual variance and bias because of the unreliable distance estimates than when setting $f = 3$, which is the value we use for all other results.

To illustrate the need for the per-pixel variance estimate used in our formulation of the NL-Means filter, we show in Figure 7.11 the result of filtering the "sibenik" scene (uniformly sampled using 32 samples per pixels) with a uniform variance estimate for all pixels in the image. In this case, we used the mean pixel variance (computed over the entire image) to guide the filtering. As expected, some regions are then correctly filtered, while others are over- or under-filtered. Some other value may yield a better result, but it would have to be hand-tuned and there would always remain an implicit trade-off between over- and under-filtered regions.

Renderings of our five test scenes are shown in Figures 7.13 to 7.17. We performed equal rendering time comparisons to account for the

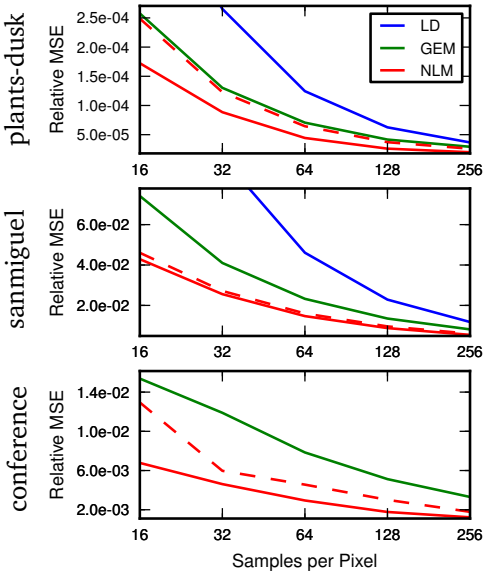


Figure 7.9: Convergence plots for the scenes of Figures 7.15 to 7.17. For our method, we show the convergence using adaptive sampling (solid line), and uniform sampling (dashed line). In all scenes, our method consistently improves upon the GEM algorithm. The gain of adaptive sampling with our method is constrained by the scene complexity. For the "conference" scene, the MSE of the LD method is too high to appear on the plot.

overhead of each method. We rendered all scenes at a resolution of 1024×1024 pixels. The "killeroos" scene illustrates the efficiency of our method for relatively low-dimensional cases including motion blur and area lighting. Our use of low-discrepancy samples coupled with the anisotropy of our filter provides a significant improvement over the GEM algorithm. The "sibenik" scene illustrates the case of a moderately noisy scene. It is path-traced and features area lighting, depth of field, and single-bounce indirect illumination. The GEM algo-

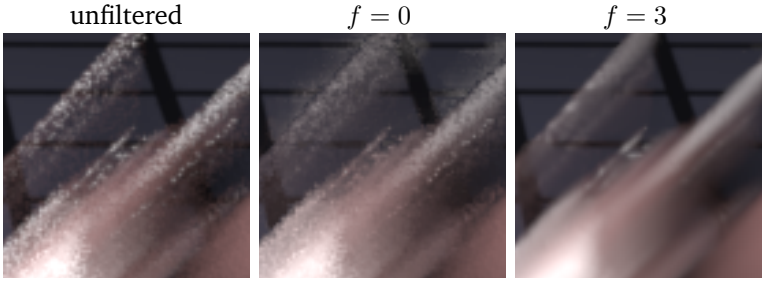


Figure 7.10: Filtering of a uniformly sampled image with 16 samples per pixel, using two patch sizes, $f = 0$ and $f = 3$. By setting $f = 0$, we obtain the same behavior as the bilateral filter, minus the spatial component of its distance computation. In both cases, we set $k = 0.45$ and $r = 10$. The smaller patch size is not sufficient to compute robust pixel distances, leading to a significantly degraded image.

rithm gives very good results in smooth regions of the scene, but has difficulties handling noise along the many sharp edges. Our method gives similar results in smooth regions, but is much more effective at resolving sharp edges. The "plants-dusk" scene features environment lighting and depth of field. In the focused grass region shown in the close-up, the GEM algorithm produces an overly blurry image and increases the MSE compared to uniform low-discrepancy sampling, illustrating the limitation of the isotropic filters used in GEM. Our method, on the other hand, preserves the small anisotropic features while further removing some of the residual noise. The "conference" scene highlights the robustness of our method to severe noise. To this end we rendered the scene using a path-tracer, even though it would be appropriate in practice to use a more sophisticated rendering algorithm that produces less noise in the first place. The scene features single-bounce indirect illumination and moderately glossy materials. The GEM method cannot reliably select the appropriate bandwidth in the presence of such high levels of noise, which leads to jarring fil-

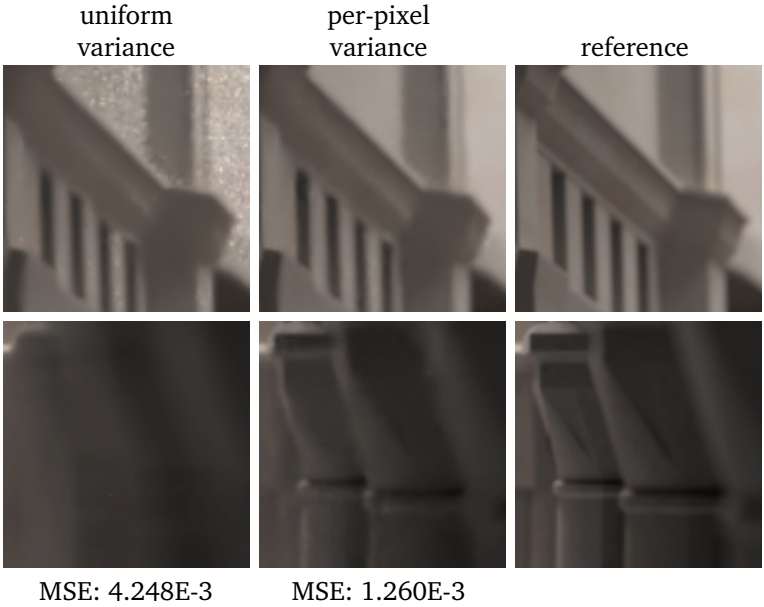


Figure 7.11: Filtering of the "sibenik" scene (uniformly sampled using 32 samples per pixels) with the standard NL-Means formulation which assumes a uniform variance across the image (left column) and our own formulation which uses per-pixel variance estimates (middle column). The reference rendering is given in the right column. While filtering using a uniform variance estimate (the mean pixel variance over the image in this case) works for some regions, many end up over- or under-filtered.

tering artifacts, while our method can still produce a pleasing image with few obvious artifacts. Also, the GEM algorithm tends to suppress noise spikes, producing darker patches on the top of the table. In contrast, our dual-buffered strategy, which decorrelates the filter weights from the noise, maintains the correct luminance by smoothing spikes instead of suppressing them. To ensure that the contribution of spikes

was sufficiently distributed, we increased the filter window size to $r = 20$ for this scene (instead of $r = 10$ for the others). We show renderings of this scene using other window sizes in Figure 7.12. The "sanmiguel" scene illustrates the behavior of our method with highly complex scenes. It is path-traced and features environment lighting, single-bounce indirect illumination, and complex textured geometry. Here again, our algorithm significantly improves upon the results of the GEM algorithm. Noisy features in particular, such as the chandelier in the inset, are much better preserved by our algorithm.

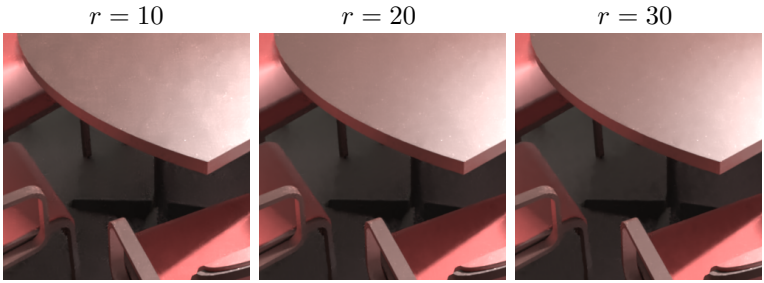


Figure 7.12: Filtering of the "conference" scene using varying filter window sizes. With a smaller window size, the spike contributions to their neighborhoods are not sufficiently spread out, yielding a higher MSE.

The computational overhead of our method is largely dominated by the cost of the filtering step. For each iteration, we filter the pixel variance, and we cross filter the two image buffers. The pixel variance is filtered with $r = 1$, while the image buffers are cross filtered using either $r = 7$ (for the intermediate iterations), or $r = 10$ (for the final iteration). The total filtering cost over the four iterations amounts to 8.5s (1.9s per intermediate iteration, and 2.8s for the final iteration). For the "conference" scene we use $r = 20$ to cross-filter the buffers during the final iteration, which increases the total filtering cost to 16.5s, the final iteration taking 10.8s. For our simplest test scene,

"sibenik", the overall overhead of our method represented less than 10% of the total rendering time, while reducing the MSE by a factor of 8.6 compared to standard low-discrepancy sampling.

Temporal Coherence. Temporal coherence is an important concern when considering video sequences, since inconsistencies between frames will produce flickering artifacts. For unfiltered video sequences these artifacts appear as fine grained temporal noise, but filtered sequences can produce disturbing flickering artifacts because of low-frequency errors. These affect large image patches and are very noticeable. We greatly mitigate these low-frequency artifacts by using a straight-forward space-time extension of the NL-Means filter [BCM08], which simply extends a pixel neighborhood to also include pixels from adjacent frames. Inter-pixel distances are still computed over spatial patches centered on the respective pixels, but the extended neighborhood leverages the full spatio-temporal data set and increases temporal coherence. We believe there is room for further improvement of the temporal filtering, and this would be a fruitful avenue for future work.

Discussion and Limitations. The efficiency of our adaptive sample distribution is quite dependent on the accuracy of the pixel statistics. For scenes featuring moderate to high noise levels, we found it preferable to use a larger set of samples in the first iteration. This lead us to our current approach of distributing evenly the sampling budget over each iteration. Still, for the "killeroos" and "plants-dusk" scenes, rendered using only 16 samples per pixel, each buffer has only two samples per pixel after the first iteration, illustrating our method's robustness to sparsely sampled data. The main limitation of our method is its image-based nature. While this offers some compelling computational advantages, it limits the adaptivity of both the sampling and the filtering. Overlapping elements with different anisotropy should be processed with different filters, which our method cannot accommodate. Similarly, we rely on brute force sampling, which can be highly inefficient when noise comes from a small subset of the sampling do-

main. Our variance estimation for low-discrepancy samples offered significant improvements for all of our test scenes compared to random sampling. Nevertheless, it could be made more accurate to better handle scenes with very high frequencies, or sharp spikes of noise. In particular, we drive the cross filtering in part using a very noisy estimate of the variance of the pixel variance computed using only two samples, the two buffers' pixel variance estimates. We believe that an estimation based on the sample distribution kurtosis may lead to a more robust filtering. In the presence of strong noise spikes, we need to use a large filter window to spread out the spike contribution which can lead to an increased bias. Since noise spikes are often due to indirect light, it should be beneficial to filter separately the direct and indirect illuminations, using a larger filter only for the indirect illumination.

7.4 Conclusion

We described a robust and versatile adaptive method for Monte Carlo rendering that builds upon the core adaptive framework of the GEM algorithm, while offering significant improvement in terms of both numerical error and visual quality. The two key characteristics of our method are its powerful non-linear filter, which can adapt to the anisotropy in the image, and its ability to process samples drawn from low-discrepancy sequences, whereas previous work was limited to samples drawn from a random distribution. The main limitation of our method, inherent to its image-space nature, is that it relies on brute force sampling to resolve noise, which is highly inefficient when useful light paths are difficult to find.

Quite a few paths are opened to improve our method. On the sampling side, it would be interesting to experiment with more robust sampling methods, such as Metropolis Light Transport [VG97]. Another venue of research would be to extend our adaptive sampling to the time domain in order to directly process animations, instead of individual frames. We could also augment the distance computa-

tion by comparing both the pixel mean and the pixel sample variance, which may improve the robustness of the filtering for low-contrast regions where the pixel mean provides insufficient constraints. Similarly, we could augment the distance computation by leveraging scene information such as each sample's normal, depth, etc.

It is this last idea, augmenting the distance computation using scene information, that we explored in the latest implementation of our adaptive framework, which we present in the next chapter.

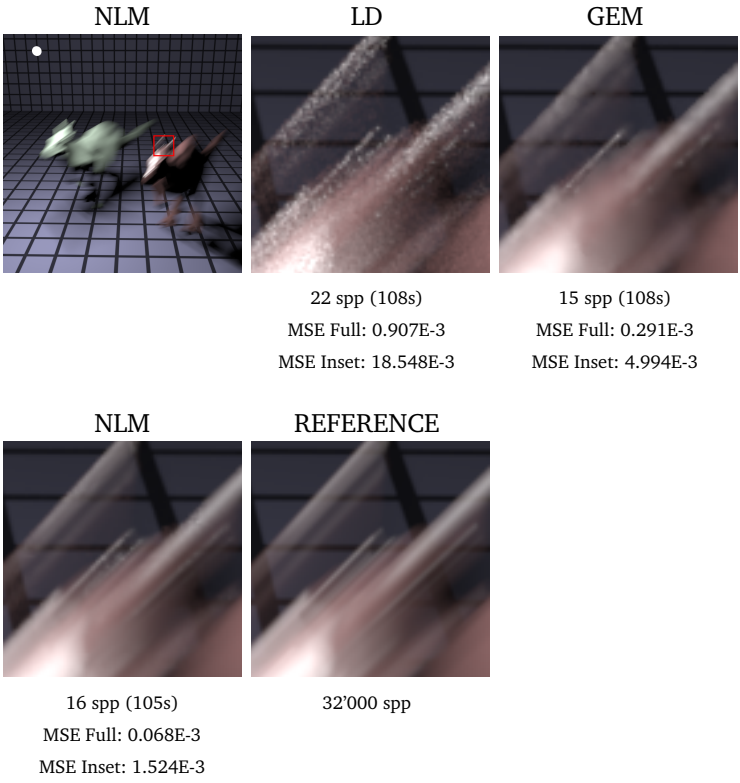


Figure 7.13: Renderings of the "killeroos" scene using a variety of methods. Non-filtered uniform low-discrepancy sampling (LD); our previous method described in Chapter 6 (GEM); our method with adaptive low-discrepancy sampling (NLM). All results for a given scene have an equal rendering time, to account for each method overhead. We use the default filtering window size, $r = 10$. The MSE values are listed under each image and the corresponding SSIM values are given in Table 7.1.

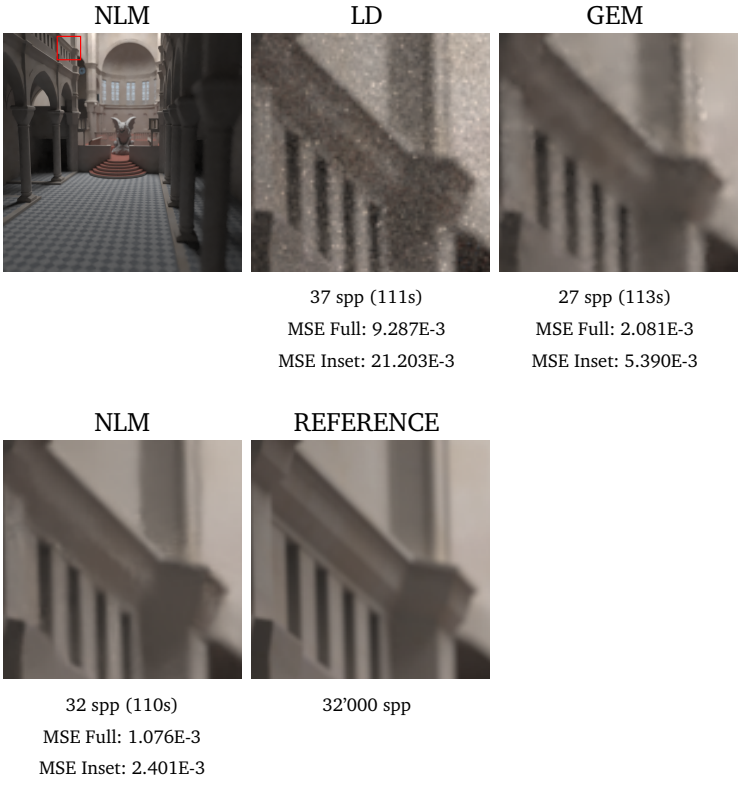


Figure 7.14: Renderings of the "sibenik" scene using a variety of methods. Non-filtered uniform low-discrepancy sampling (LD); our previous method described in Chapter 6 (GEM); our method with adaptive low-discrepancy sampling (NLM). All results for a given scene have an equal rendering time, to account for each method overhead. We use the default filtering window size, $r = 10$. The MSE values are listed under each image and the corresponding SSIM values are given in Table 7.1.

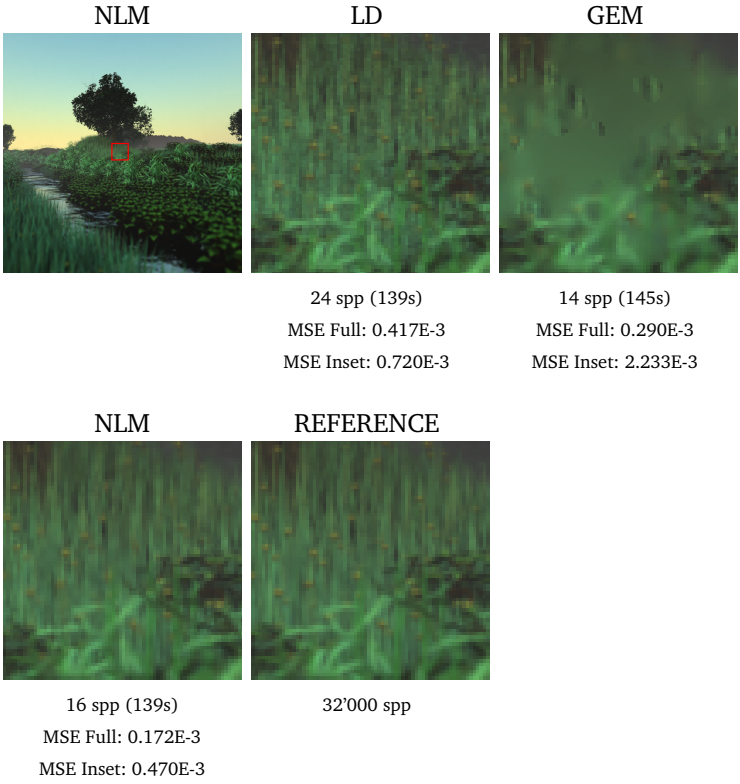


Figure 7.15: Renderings of the "plants-dusk" scene using a variety of methods. Non-filtered uniform low-discrepancy sampling (LD); our previous method described in Chapter 6 (GEM); our method with adaptive low-discrepancy sampling (NLM). All results for a given scene have an equal rendering time, to account for each method overhead. We use the default filtering window size, $r = 10$. The MSE values are listed under each image and the corresponding SSIM values are given in Table 7.1.

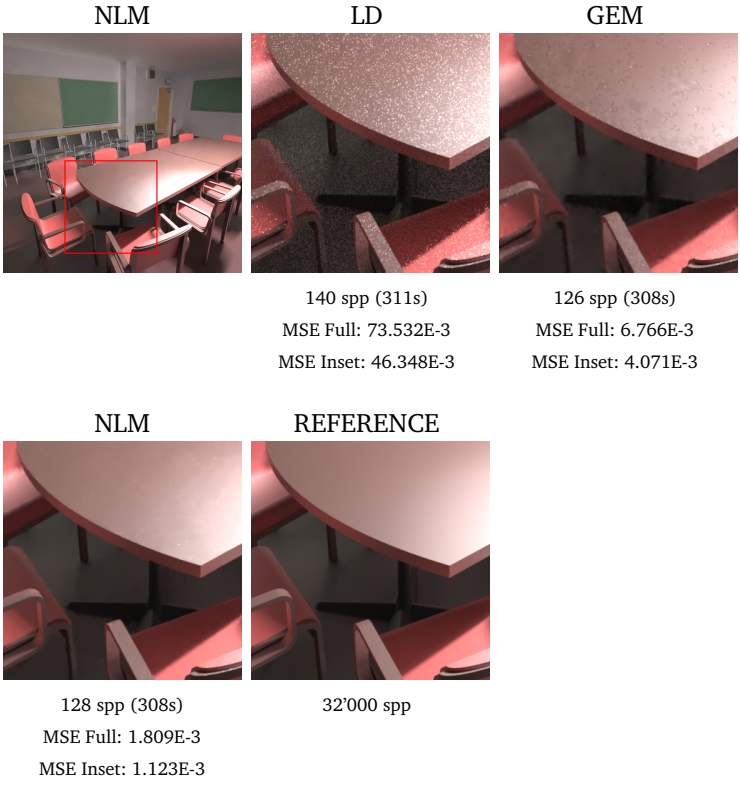


Figure 7.16: Renderings of the "conference" scene using a variety of methods. Non-filtered uniform low-discrepancy sampling (LD); our previous method described in Chapter 6 (GEM); our method with adaptive low-discrepancy sampling (NLM). All results for a given scene have an equal rendering time, to account for each method overhead. We use a larger filtering window size, $r = 20$ (instead of the default $r = 10$) to ensure the noise spikes are sufficiently spread out. The MSE values are listed under each image and the corresponding SSIM values are given in Table 7.1.

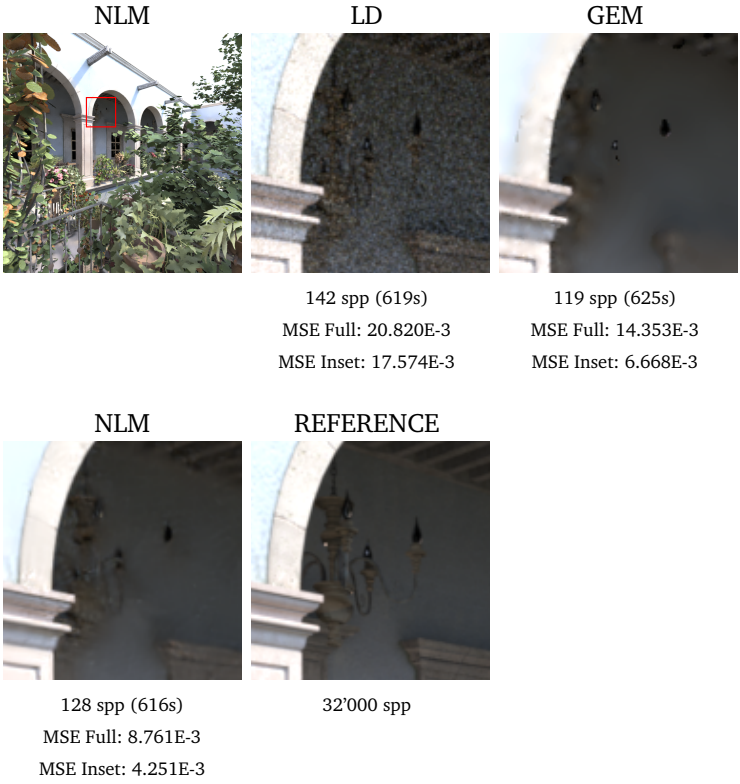


Figure 7.17: Renderings of the "sanmiguel" scene using a variety of methods. Non-filtered uniform low-discrepancy sampling (LD); our previous method described in Chapter 6 (GEM); our method with adaptive low-discrepancy sampling (NLM). All results for a given scene have an equal rendering time, to account for each method overhead. We use the default filtering window size, $r = 10$. The MSE values are listed under each image and the corresponding SSIM values are given in Table 7.1.

Chapter 8

Adaptive rendering using pixel color and feature information

This chapter presents our DFC algorithm, the third, and last, implementation of our adaptive framework. The name DFC stands for Denoising using Features and Color information. This algorithm was developed in 2013 and the content of this chapter is reproduced from [RMZ13].

Image space adaptive rendering methods have proved to be surprisingly effective at addressing noise artifacts in Monte Carlo renderings. These methods are appealing because they are relatively easy to implement, mostly orthogonal to other variance reduction techniques, applicable to general light transport effects, computationally efficient, and often competitive with more specialized approaches targeting specific rendering effects. A particularly successful idea is to use feature buffers, such as per-pixel normals, textures, or depth, to compute denoising filter weights. Feature buffers are highly correlated with the rendered image, since they represent most edges in the image, but

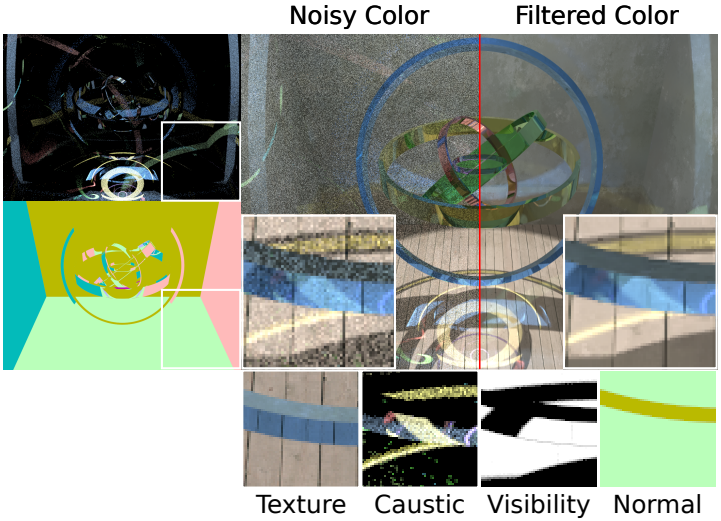


Figure 8.1: We propose a method to denoise Monte Carlo renderings using noisy color (top left) and feature buffers (bottom: texture, caustic, visibility, normal) as an input. We construct a denoising filter by combining color and feature information using a SURE error estimate. Our result (top right) improve visually and quantitatively over the previous state-of-the-art.

they are usually much less noisy than the color output of the Monte Carlo renderer. Therefore, filters based on feature buffers effectively remove noise while preserving most edges. Unfortunately, they are prone to blurring image details that are not well represented by the features. Feature buffers have been used to improve denoising in the context of anisotropic diffusion, guided image filtering, which is based on local regression, and cross-bilateral filtering.

In this chapter we present a method to robustly combine color and feature buffers to improve denoising performance. We use a filtering framework based on NL-Means filter weights for color buffers and

bilateral weights for feature buffers. We control the influence of color and feature buffers by adjusting the parameters of the NL-Means and bilateral filters. To combine color and feature information, we evaluate three candidate filters using different parameters designed to provide a trade-off between fidelity to image detail and robustness to noise. Then we compute a weighted average of the candidate filters on a per-pixel basis using a SURE-based error estimate to minimize the output error. We deal with noisy features by denoising them first in a separate step using an NL-Means filter. This allows us to include novel features, such as a caustics buffer shown in Figure 8.1, and a direct visibility feature. We demonstrate that our approach leads to significant improvements both in subjective and quantitative errors compared to the previous state-of-the-art. In summary, we make the following contributions:

- We propose to combine color and feature buffers to improve image space denoising of Monte Carlo renderings.
- We implement this idea based on NL-Means and cross-bilateral filtering, and a SURE-based error estimate.
- We propose to deal with noisy features by denoising them in a separate step. This allows us to introduce novel features such as caustics and direct visibility.
- We demonstrate significant subjective and numerical improvements in image quality over the previous state-of-the-art.
- We extend our approach to adaptive sampling and space-time filtering for animations.

8.1 Overview

Filtering based on feature buffers, such as per-pixel normal, texture, or depth, has proven extremely effective, in particular for images rendered with very few samples per pixel and high noise levels in the

Monte Carlo output [SD12, LWC12]. On the other hand, image details that are not represented in the feature buffers tend to be blurred by such approaches. Hence our main idea is to construct a filter that implements a balance between filtering using color and feature information. In general, our filter computes a weighted average of neighboring pixels. The filtered color values $F(p) = (F_1(p), F_2(p), F_3(p))$ of a pixel p in a color image $u(p) = (u_1(p), u_2(p), u_3(p))$ are

$$F_i(p) = \frac{1}{C(p)} \sum_{q \in N(p)} u_i(q) w(p, q), \tag{8.1}$$

where $N(p)$ is a $2r + 1 \times 2r + 1$ square neighborhood centered on p, $w(p, q)$ is the weight of the contribution of neighboring pixel q to p, i is the index of the color channel, and $C(p) = \sum_{q \in N(p)} w(p, q)$ is a normalization factor. The fundamental challenge is to determine suitable weights $w(p, q)$.

In our approach, illustrated in Figure 8.2, we construct three *candidate filters*, which we call the *FIRST*, *SECOND*, and *THIRD* candidate filter. We design the filters such that the FIRST filter is most sensitive to details in the color buffer, but also most sensitive to its noise; the THIRD filter is least sensitive to noise in the colors, but also least sensitive to its details; and the SECOND filter is in between. Then we compute the final filter as a weighted average of the candidate filters using a SURE-based per-pixel error estimate. We build the candidate filters from two types of weights, called color and feature weights. We obtain the color weights as NL-Means weights from the noisy color output of the Monte Carlo renderer as described in Section 8.2. We compute the feature weights as bilateral weights from the feature buffers, as presented in Section 8.3. In Section 8.4 we then describe how we construct the FIRST, SECOND, and THIRD candidate filters from the color and feature weights, and how we compute the candidate filter averaging weights using SURE error estimation. We provide a summary of our algorithm in Section 8.5, and in Section 8.6 we present extensions to adaptive sampling and space-time filtering for animations.

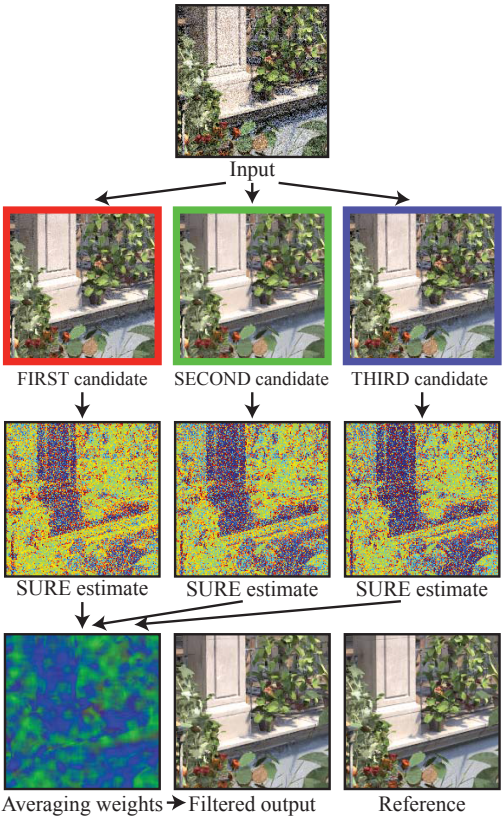


Figure 8.2: Our main idea is to filter using a balance of color and feature information. We evaluate three candidate filters designed to provide a trade-off between sensitivity to image detail and robustness to noise. We compute averaging weights for the candidate filters at each pixel using a SURE-based error estimation. The color coding of the weights corresponds to the three candidates.

8.2 NL-Means Weights from Color Buffer

Our color weights w_c are based on NL-Means filtering [BCM05], which has proven effective for denoising Monte Carlo renderings because it can easily be generalized to spatially varying variances typical in such data (see Section 7.1.1). We compute the NL-Means weights from the noisy color output of the Monte Carlo renderer, and per-pixel variance estimates. We next review NL-Means weights computation and then describe our per-pixel variance estimates.

NL-Means Weights. NL-Means weights for a pixel p and a neighbor q are determined based on the distance $d_c^2(P(p), P(q))$ between a pair of small patches $P(p)$ and $P(q)$ of size $2f+1 \times 2f+1$ centered at p and q,

$$d_c^2(P(p), P(q)) = \frac{1}{3(2f+1)^2} \sum_{i=1}^{3} \sum_{n \in P(0)} \Delta_i^2(p+n, q+n),$$

where $\Delta_i^2(p+n, q+n)$ is is a per-pixel distance in color channel i and $n \in P(0)$ are the offsets to each pixel within a patch. We follow the approach used in our NLM algorithm (see Section 7.1.1) and define the per-pixel distance as

$$\Delta_i^2(p,q) = \frac{(u_i(p) - u_i(q))^2 - (\mathrm{Var}_i[p] + \mathrm{Var}_i[q,p])}{\epsilon + k_c^2(\mathrm{Var}_i[p] + \mathrm{Var}_i[q])}.$$

The term $(u_i(p) - u_i(q))^2$ measures the squared difference between the color values at pixels p and q. Since $u_i(p)$ and $u_i(q)$ are noisy this consistently overestimates the true squared difference. Hence, we subtract a variance cancellation term $(\mathrm{Var}_i[p] + \mathrm{Var}_i[q,p])$ to remove this bias, similar as proposed originally for NL-Means [BCM05], where $\mathrm{Var}_i[p]$ is a variance estimate for the sample mean in pixel p, and $\mathrm{Var}_i[q,p] = min(\mathrm{Var}_i[q], \mathrm{Var}_i[p])$. The denominator $\epsilon + k_c^2(\mathrm{Var}_i[p] + \mathrm{Var}_i[q])$ is a normalization factor, where ϵ is a small value to prevent division by zero, and k_c is a user specified factor that controls the

sensitivity of the filter to color differences. Larger values of k_c lead to more aggressive filtering. Finally, we obtain our color filter weight $w_c(p, q)$ of the contribution of pixel q to p using an exponential kernel,

$$w_c(p, q) = \exp^{-\max(0, d_c^2(P(p), P(q)))}. \tag{8.2}$$

Variance Estimation. In the case of random sampling, we could estimate the variances Var_i of pixel means simply by considering the sample variance within each pixel. This approach is not suitable, however, to support low-discrepancy sampling where it will consistently overestimate the variance. In our previous NLM algorithm, we addressed this problem by splitting the noisy color samples into two half-buffers and computing the empirical variance of these half-buffers (see Section 7.1.1). While this is an unbiased estimate of the pixel variance, it is also very noisy and leads to poor NL-Means filtering performance if used directly. To address this, we observe that the sample variance exhibits the detailed structure of the actual spatially non-uniform variance, but with a systematic bias. Hence we attempt to remove this bias by simply scaling the sample variance to match the magnitude of the two-buffer variance. We smooth both the sample variance and the two-buffer variance with a large, 21×21 box filter and then compute the ratio between the two on a per-pixel basis. We then apply this ratio on the initial unfiltered sample variance. This results in a variance estimate with the lower noise of the sample variance, and the correct magnitude of the two-buffer variance.

8.3 Bilateral Weights from Feature Buffers

We determine the feature weights w_f using the feature buffers and bilateral weights. An important distinction compared to previous work exploiting feature buffers [LWC12, SD12] is that we deal with noisy features by first prefiltering them separately. We next describe our feature prefiltering technique, and then the computation of the bilateral weights using the prefiltered features.

Feature Prefiltering. Our prefiltering approach exploits the fact that features can be denoised effectively because their dynamic range, and hence their variance, is typically limited. We apply an NL-Means filter as described in the previous section including the same method to estimate the input variance, although using individual features instead of color as input. We choose window radius $r = 5$, patch radius $f = 3$, and sensitivity $k_c = 1.0$ for all features.

Output Variance Estimation. To determine the bilateral weights we will also require the residual variance of the prefiltered features, that is, we need the per-pixel variance of the prefiltered output. We obtain the output variance using a two-buffer approach similar to Section 8.2. We split the feature data into two half-buffers that we both filter using the same NL-Means weights determined from the complete data, as described above. Note that given fixed weights, the filter (see Equation 8.1) is linear, and averaging the filtered half-buffers is equivalent to filtering the full data. By processing the half-buffers, however, we can estimate the residual per-pixel variance as the squared per-pixel difference between the filtered half-buffers. We further reduce noise in this two-buffer variance estimate by smoothing it with a small Gaussian kernel with standard deviation of 0.5 pixels.

Bilateral Weights. We denote feature buffers such as normals, textures, or depth, denoised and normalized to unit range, as f_j. The feature distance $\Phi_j^2(p, q)$ for feature j between pixels p and q is based on the squared feature difference including variance cancellation similar to Section 8.2,

$$\Phi_j^2(p,q) = \frac{(f_j(p) - f_j(q))^2 - (\mathrm{Var}_j[p] + \mathrm{Var}_j[p,q])}{k_f^2 \max(\tau, \max(\mathrm{Var}_j[p], \|\mathrm{Grad}_j[p]\|^2))},$$

normalized by two factors. First, the user parameter k_f controls the sensitivity of the filter to feature differences. The second factor depends on the residual variance of the prefiltered feature (as described above) denoted by $\mathrm{Var}_j[p]$ and the squared gradient magnitude $\|\mathrm{Grad}_j[p]\|^2$,

thresholded to a minimum value τ. This factor normalizes the feature distance relative to residual noise left in the filtered feature and the local feature contrast, measured by its gradient magnitude. Finally, we obtain an overall distance $d_f(p,q)$ by taking the maximum distance over all M features,

$$d_f^2(p,q) = \underset{j\in[1\ldots M]}{\operatorname{argmax}} \Phi_j^2(p,q),$$

and the final feature weight $w_f(p,q)$ is obtained using an exponential kernel, similar as in Section 8.2,

$$w_f(p,q) = \exp^{-d_f^2(p,q)}. \tag{8.3}$$

We illustrate the benefit of our feature prefiltering step in Figure 8.3 on a simple scene with depth of field. With prefiltering, we effectively remove noise in out-of-focus regions, while preserving detail otherwise. Feature prefiltering allows us to exploit novel types of features that tend to be too noisy to be useful without prefiltering. For the "rings" scene in Figure 8.1 we define a caustics feature as the per-pixel density of caustic photons. We also introduce a direct illumination visibility feature as the fraction of shadow rays that hit any light source over all direct shadow rays evaluated in a pixel. Figure 8.4 illustrates how considering the feature gradient in the distance normalization improves filtering performance. Without the gradient term feature weights along edges are too restrictive, preventing effective filtering.

8.4 Filter Weighting using SURE

We use a SURE-based approach [Ste81] to estimate the mean squared error (MSE) of three candidate filters, which we design to provide a trade-off between fidelity to image detail and robustness to noise. We then leverage the error estimate to compute a weighted average of the candidate filters minimizing the error on a per-pixel basis. We next describe the candidate filters, the SURE-based error estimate, and the computation of the per-pixel filter averaging weights.

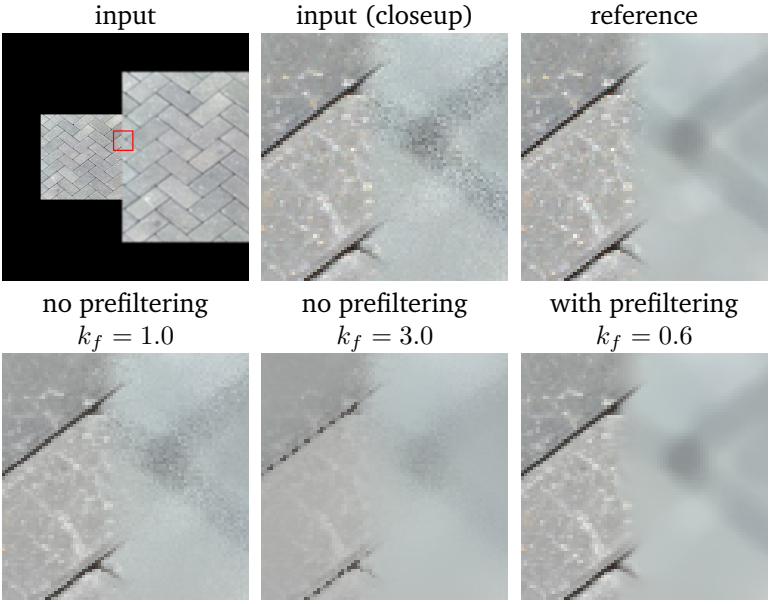


Figure 8.3: We show the effectiveness of feature prefiltering on a scene with depth-of-field, which leads to noisy features. We use only the texture feature. The left quad is in-focus and noise free, while the right quad is out-of-focus and noisy (top left). Our approach with prefiltering preserves detail in the in-focus-region while effectively denoising the out-of-focus region (bottom right). Without prefiltering the bilateral feature weights fail to distinguish between noise and texture detail. While smaller k_f values preserve texture detail, they cannot remove the noise (bottom left). Larger k_f values yield smoother results but blur out the details (bottom middle).

Candidate Filters. Our FIRST, SECOND, and THIRD candidate filters (Figure 8.2) differ in their color sensitivity k_c and patch radius f. The FIRST filter uses $k_c = 0.45$ and a small patch radius of $f = 1$, which makes it sensitive to small image detail but also less robust to

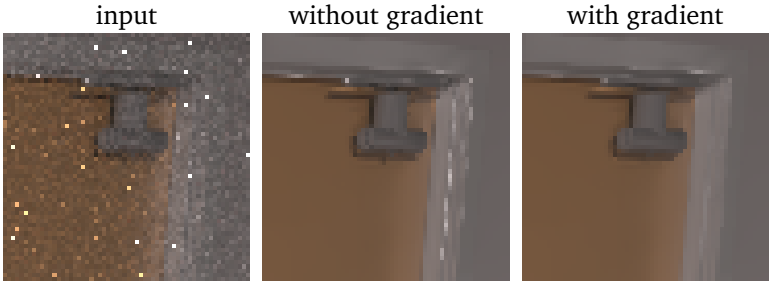


Figure 8.4: Filtered output for the "conference" scene without (middle image) and with the feature gradient (right image). Including the gradient term improves filtering along edges.

noise. The SECOND filter is the same except that it uses a larger patch radius $f = 3$. Hence it is more robust to noise but less effective at filtering intricate image detail at low noise levels. The THIRD filter has $k_c = \infty$, which means that the color information does not influence the filter weights and the patch size f is irrelevant. This filter is most robust towards noise since its weights completely ignore color information. However, it fails to recover image detail that is not represented in the features. All filters use feature sensitivity $k_f = 0.6$ and the same window radius r, which is the only parameter we expose to the user. We determine the final filter weights by taking the minimum of the color and feature weights, that is

$$w(p, q) = \min(w_c(p, q), w_f(p, q)). \tag{8.4}$$

SURE Error Estimate. We explain the SURE error estimate by extending our notation from Equation 8.1 for a generic color image filter. Let us interpret the output $F_{u_i}(p)$ of the filter at pixel p as a function of the noisy value u_i of color channel i at p. The SURE error estimator

at pixel p is then

$$\mathrm{SURE}(p) = \sum_{i=1}^{3} \|F_{u_i}(p) - u_i\|^2 - \sigma_i^2 + 2\sigma_i^2 \frac{\mathrm{d}F_{u_i}(p)}{\mathrm{d}u_i} \tag{8.5}$$

where σ_i^2 is the true variance of the pixel mean of color channel i. Since our color weights w_c include discontinuous terms, they are technically not differentiable. But we found that a finite difference approximation of the derivatives still leads to reliable error estimates in practice. Hence we approximate $\mathrm{d}F_{u_i}(p)/\,\mathrm{d}u_i$ as,

$$\frac{\mathrm{d}F_{u_i}(p)}{\mathrm{d}u_i} \approx \frac{F_{u_i+\delta}(p) - F_{u_i}(p)}{\delta}, \text{ where } \delta = 0.01 \times u_i.$$

Candidate Filter Averaging. While SURE provides an unbiased error estimate, it is very noisy and needs to be filtered for per-pixel error minimization. A straightforward approach would be to spatially smooth the error estimate until its variance is low enough to reliably determine the best candidate filter on a per-pixel basis. Unfortunately, in our typical data the SURE estimate tends to be contaminated by strong outliers. This requires large spatial smoothing filters, introducing bias in the per-pixel error estimates.

Instead, we achieved better results with a two-step strategy that is more robust to outliers in the initial error estimate. In a first step, we smooth the error estimate with a small kernel and obtain three binary selection maps for the candidate filters, containing a 1 in each pixel if the filter had lowest error, and 0 otherwise. Since the FIRST candidate filter is most sensitive to noise it may occur that it preserves noise, but the SURE estimate does not register the error. We avoid such residual noise by selecting the FIRST candidate only if it filters more than the SECOND, that is, the derivative term in the SURE estimate is lower for the FIRST candidate. In the second step we smooth the binary maps with a larger kernel. This approach has the advantage that the binary selection maps suppress outliers, which allows us to use smaller smoothing kernels in the second step. We found that the

smoothing step of the initial error estimate is necessary to obtain sufficiently discriminatory binary maps.

8.5 Algorithm

We summarize our algorithm in Algorithm 1. The workhorse of our filtering pipeline, which we use to compute the three candidate filters and some auxiliary filtering, is the function

$$[out, d\_out\_d\_in] = \mathit{flt}(in, u, var\_u, f[], var\_f[], p).$$

It filters an input in by constructing color weights (Equation 8.2) from a color buffer u with variance var_u, and feature weights (Equation 8.3) from a set of feature buffers $f[]$ with variances $var\_f[]$, and taking their minimum as in Equation 8.4. The function also returns the derivative $d\_out\_d\_in$ that is needed for the SURE estimate (Equation 8.5). The parameter values are specified in the structure p. We further assume that the input in and output out of the filter consist of two half-buffers to support two-buffer output variance estimation (Section 8.3, output variance estimation). We implement the SURE error estimate (Equation 8.5) in a function

$$SURE(u, var\_u, F, dF\_du)$$

that takes a noisy buffer u with its variance var_u, and its filtered version F with derivative dF_du as input. The algorithm also uses an auxiliary function $scale\_svar$ that takes a buffer (consisting of two half-buffers) and its sample variance as input and implements the sample variance scaling technique described in Section 8.2. Finally, the function $buffer\_var$ computes the two-buffer variance of a buffer (consisting of two half-buffers) smoothed with a Gaussian kernel (Section 8.3, output variance estimation). In addition to the steps described so far, our algorithm includes a final filtering pass that removes residual noise (lines $27 - 29$), which we detect by computing the two buffer variance of the output. This residual noise is typically located along edges, where the filter is more constrained.

8.6 Extensions

Adaptive Sampling. We follow the same adaptive sampling strategy as in our NLM algorithm (see Chapter 7), which we summarize here. We distribute samples over multiple iterations, each having an equal share of the sample budget. The first iteration performs uniform sampling, and the subsequent ones perform adaptive sampling. In the adaptive iterations, the sampling density is proportional to the estimated relative MSE returned by SURE, scaled by a weight W. The weight W represents the error reduction potential of a single sample, and accounts for the number of samples used in the filter, as well as the filter support,

$$W(p) = \frac{\sum_{q \in N(p)} w(p,q)}{1 + n_p},$$

where p is a pixel, $N(p)$ the filter window around p, $w(p,q)$ the weight of a neighbor q within the window, and n_p the number of samples already contributing to the filtered value of p. The resulting weighted error is quite noisy, so we filter it aggressively with our flt function and parameters $r = 10$, $k_c = 1.0$, and $k_f = \infty$. Lastly we clamp the number of samples allocated to each pixel to a fixed value to prevent spending too many samples on outliers.

Space-time Filtering for Animations. Filtering animations on a per-frame basis suffers from disturbing flickering artifacts due to low-frequency residual noise. We can greatly mitigate these problems by space-time filtering. We implement space-time filtering in our framework by extending our filtering window from a spatial to a spatio-temporal window across several frames, as proposed by Buades et al. [BCM08]. Otherwise our algorithm remains the same as before.

8.7 Results

We integrated our method as an extension of the PBRT rendering framework [PH10] and implemented the filtering operations themselves in CUDA for GPU acceleration. The complexity of each filtering step is proportional to image resolution, window radius f, and patch radius r. For an image resolution of 1024×1024 pixels and user specified window radius $r = 10$, the complete filtering pipeline summarized in Algorithm 1 takes 5.4 seconds on an NVidia GeForce GTX TITAN GPU, and 8.0 seconds on a Geforce GTX 580 GPU. All timings reported below were measured on a workstation with dual 6-core Intel Xeon processor at 2.3 GHz, 12 rendering threads, and a Geforce GTX TITAN GPU.

The "rings" scene in Figure 8.1 is rendered using PBRT's photon mapper using 32 samples per pixel, 450k caustic photons, 1000k indirect photons, and 4 final gather rays per sample. It took 869s to render, and 19.5s to filter with a window radius $r = 20$ (the filtering cost is higher for this scene, since we have the additional caustics buffer and a larger window radius). It uses our novel caustics feature, which stores the density of caustics photons. We use ray differentials to compute pixel-sized radiance estimation radii to avoid blurring of caustics. Figure 8.5 further highlights the contribution of our novel features. We show closeups of the "sanmiguel" scene filtered with and without the visibility feature, and the "rings" scene with and without the caustics feature. Including the novel features clearly improves our results.

Figure 8.6 contains log-log convergence plots for the "sibenik", "conference", and "sanmiguel" scenes. We show results for our complete filter using uniform sampling (DFC in cyan) and adaptive sampling (dotted cyan), our candidate filters (red, green, and blue), and the work by Li et al. [LWC12] (SBF in magenta). The plots indicate that our SURE-based weighted averaging of the candidate filters consistently improves the error of the individual candidate filters. It is also interesting that both our FIRST and SECOND filters generally outperform SBF, which underlines the usefulness of including color

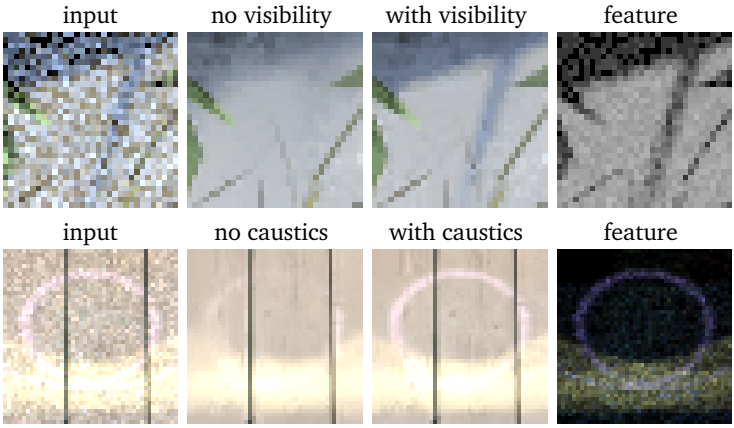


Figure 8.5: Closeups of "sanmiguel" filtered with and without the visibility feature, and "rings" with and without the caustics feature. The features shown on the right (before prefiltering) help preserving important details.

weights in the filter.

In Figure 8.7 we use the mean of the BRDF samples as a feature buffer instead of a texture. The BRDF value is readily available in a raytracer independent of rendered materials, whereas a texture may not be well-defined or easily extracted for many materials. While the mean of the BRDF samples is typically noisier than a texture, we can still handle this case with our feature prefiltering approach, although finer details may not be preserved as well.

In Figure 8.8, we illustrate our adaptive rendering scheme on the "sibenik" scene with an average of 16 samples per pixels. We also compare to SBF using the original authors' implementation. Adaptive sampling improves our MSE by roughly 18% compared to the result presented in Figure 8.11 with uniform sampling. The convergence plots of Figure 8.6 also indicate the MSE obtained with adaptive sampling with dotted lines, showing a consistent improvement over

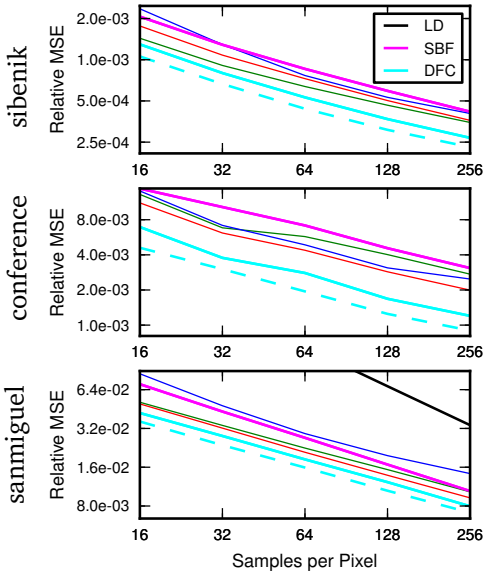


Figure 8.6: Convergence plots for the scenes of Figures 8.9 to 8.11 for our method and the SURE-based approach by Li et al. [LWC12]. For our method, we show the errors of the FIRST, SECOND, and THIRD candidate filters and the final output. We indicate results from adaptive sampling with dotted lines.

uniform sampling.

In Figures 8.9 to 8.13, we compare rendering with low-discrepancy sampling and no filtering (LD), the approach by Li et al. [LWC12] (SBF) using the implementation provided by the authors, and our technique (DFC) at roughly equal render time. We also include a reference image (REFERENCE). We use uniform sampling in this comparison since we found that SBF often gave worse results with adaptive sampling. Our feature buffers include texture, depth, normal, and visibility. We attribute our improvements over previous work to three main factors: the consideration of the color buffer to construct the

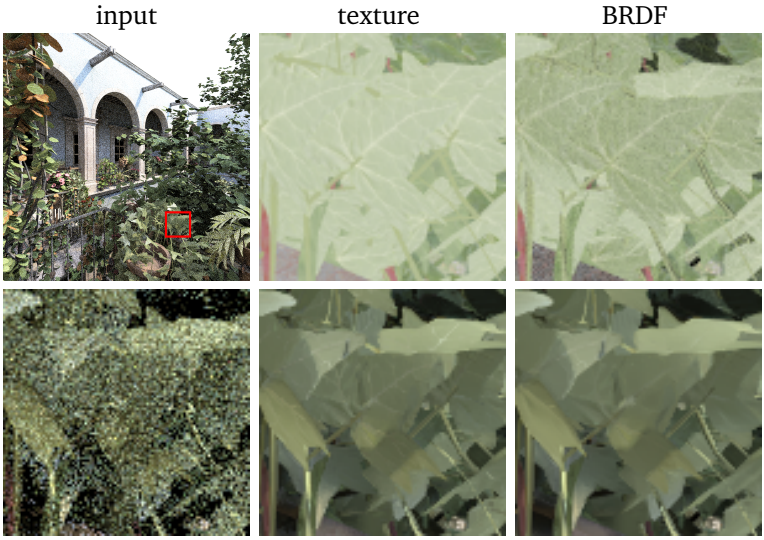


Figure 8.7: Using the mean of BRDF samples instead of textures as a feature, with the "sanmiguel" scene at 32 samples per pixel. The feature buffers are given in the top row, with the corresponding filtered output and MSE values below. Noise in the BRDF samples may lead to loss of some fine texture details at low sampling rates.

final filter, the use of the visibility feature, and the improved handling of noisy features. The "conference" scene highlights the benefits of the visibility feature. We preserve the shadows, while SBF, which does not use visibility features, is not able to do so. For this scene we used a larger window radius of $r = 20$ to remove spike noise. In the comparison with SBF we obtain a much lower MSE. In the "sanmiguel" scene, our filter yields a smoother result while preserving the foliage details as well as the small direct shadow details, which SBF blurs because of the absence of the visibility feature. In the "sibenik" scene we

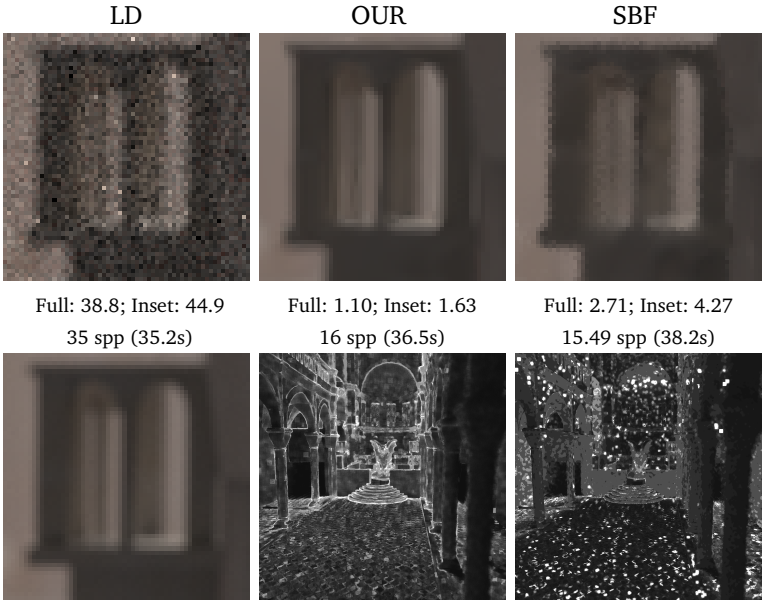


Figure 8.8: Adaptive rendering of the "sibenik" scene at an average of 16 spp using our method and the SBF algorithm, including MSE values (multiplied by $10E3$) and sampling density maps. The bottom left image is the reference.

obtain results largely free of artifacts in out-of-focus regions, where typically all features are noisy. In contrast, SBF suffers from residual noise in these areas. In the "teapot-metal" scene, our filter can better preserve the chaotic structure of the floor, as well as the glossy highlights that are not captured in the feature buffers. The "dragonfog" scene includes participating media to demonstrate the flexibility of our approach in dealing with a variety of rendering effects, and was also filtered with a window radius $r = 20$.

A limitation of our approach is that at very low sampling rates, typically less than 10 spp, the color weights tend to become less useful

and SURE-error estimation less reliable because of excessive variance in the color buffer. In such situations it is possible that our final filter using candidate filter averaging fails to improve the global MSE over the best candidate filter. At such low sampling rates it also becomes challenging to prefilter noisy features such as the visibility. In these cases the filter parameters need to be adjusted to obtain reasonable results. It is important to note, however, that all images were rendered using fixed parameters as in Algorithm 1.

8.8 Conclusions

We have presented a method for denoising Monte Carlo renderings by constructing filters using a combination of color and feature information. We construct three candidate filters based on NL-Means weights for color buffers and cross-bilateral weights for feature buffers. We determine robust averaging weights of the three candidate filters on a per-pixel basis using SURE error estimation. We also introduce a novel approach to dealing with noisy features using a prefiltering step, and we apply it to new caustics and visibility features. Together, candidate filter weighting including color information, feature prefiltering, and the novel caustics and visibility features provide significant improvements in terms of both visual and numerical quality over the previous state-of-the-art. While the computational cost of our filter is insignificant in the context of off-line rendering, the technique is not suitable for interactive rendering. In the future, we will explore extensions of our approach targeted at real-time applications. It would also be interesting to further improve methods targeted at extremely low sampling rates.

Algorithm 1: DENOISE

Input: Noisy color buffer c with sample variance $svar\_c$; set of M noisy feature buffers $f[]$ with sample variances $svar\_f[]$; window radius R
Output: Denoised image $pass2$

begin
  /* **Sample variance scaling.** */
  $var\_c = scale\_svar(svar\_c, c)$
  for *all features* $j = 1 \dots M$ **do**
    $var\_f[j] = scale\_svar(svar\_f[j], f[j])$
  /* **Feature prefiltering.** */
  $p = \{k_c = 1, k_f = \infty, f = 3, r = 5\}$
  for *all features* $j = 1 \dots M$ **do**
    $flt\_f[j] = flt(f[j], f[j], var\_f[j], nil, nil, p)$
    $var\_flt\_f[j] = buffer\_var(flt\_f[j])$
  /* **Candidate filters.** */
  $p = \{k_c = 0.45, k_f = 0.6, f = 1, r = R, \tau = 10E - 3\}$
  $[r, d\_r] = flt(c, c, var\_c, flt\_f[], var\_flt\_f[], p)$
  $p = \{k_c = 0.45, k_f = 0.6, f = 3, r = R, \tau = 10E - 3\}$
  $[g, d\_g] = flt(c, c, var\_c, flt\_f[], var\_flt\_f[], p)$
  $p = \{k_c = \infty, k_f = 0.6, f = 1, r = R, \tau = 10E - 4\}$
  $[b, d\_b] = flt(c, c, var\_c, flt\_f[], var\_flt\_f[], p)$
  /* **Filtered SURE error estimates.** */
  $p = \{k_c = 1.0, k_f = \infty, f = 1, r = 1, \tau = 10E - 3\}$
  $e\_r = flt(SURE(c, var\_c, r, d\_r), c, var\_c, nil, nil, p)$
  $e\_g = flt(SURE(c, var\_c, g, d\_g), c, var\_c, nil, nil, p)$
  $e\_b = flt(SURE(c, var\_c, b, d\_b), c, var\_c, nil, nil, p)$
  /* **Binary selection maps.** */
  $sel\_r = e\_r < e\_g$ && $e\_r < e\_b$ && $d\_r < d\_g$? 1 : 0
  $sel\_g = e\_g < e\_r$ && $e\_g < e\_b$? 1 : 0
  $sel\_b = e\_b < e\_r$ && $e\_g < e\_b$? 1 : 0
  /* **Filter selection maps.** */
  $p = \{k_c = 1, k_f = \infty, f = 1, r = 5, \tau = 10E - 3\}$
  $sel\_r = flt(sel\_r, c, var\_c, nil, nil, p)$
  $sel\_g = flt(sel\_g, c, var\_c, nil, nil, p)$
  $sel\_b = flt(sel\_b, c, var\_c, nil, nil, p)$
  /* **Candidate filter averaging.** */
  $pass1 = r * sel\_r + g * sel\_g + b * sel\_b$
  /* **Second pass filtering.** */
  $p = \{k_c = 0.45, k_f = \infty, f = 1, r = R, \tau = 10E - 4\}$
  $var\_pass1 = buffer\_var(pass1)$
  $pass2 = flt(pass1, pass1, var\_pass1, nil, nil, p)$
end

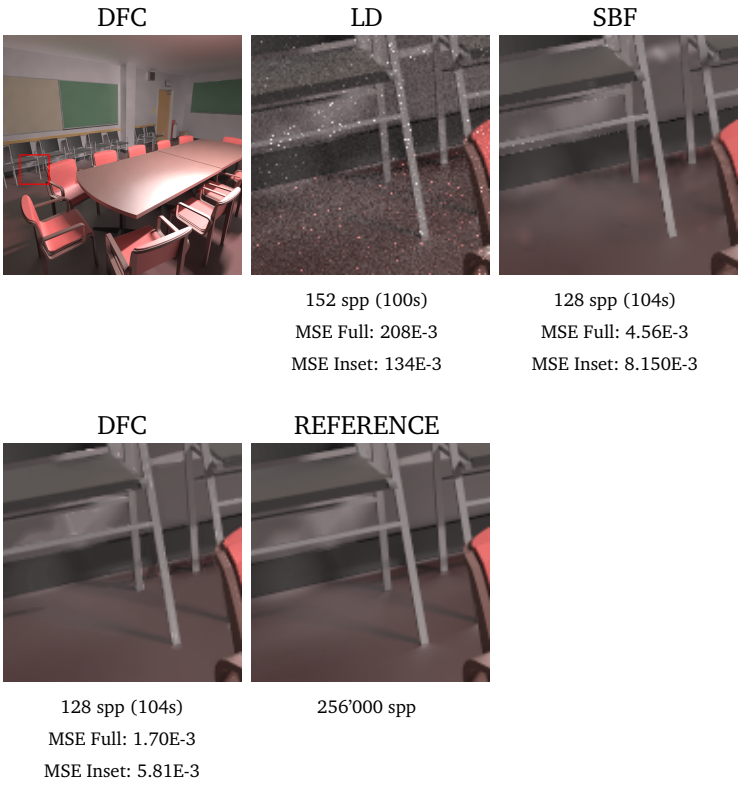


Figure 8.9: Renderings of the "conference" scene, comparing our approach (DFC) to path tracing with low-discrepancy (LD) and the SURE-based approach by Li et al. [LWC12] (SBF). Images are rendered at a resolution of 1024×1024 using PBRT at roughly equal render time. Under each image, we give the number of samples per pixel (spp), rendering time in seconds, and MSE. We provide consistently better MSE values, which we attribute to three main factors: the consideration of the color buffer to construct the final filter, the use of visibility features, and the improved handling of noisy features.

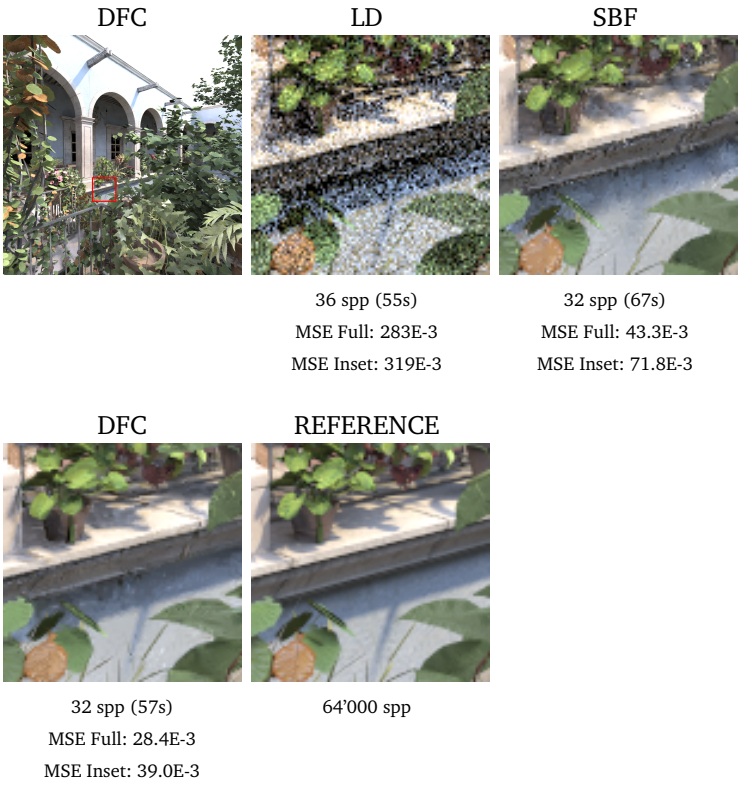


Figure 8.10: Renderings of the "sanmiguel" scene, comparing our approach (DFC) to path tracing with low-discrepancy (LD) and the SURE-based approach by Li et al. [LWC12] (SBF). Images are rendered at a resolution of 1024×1024 using PBRT at roughly equal render time. Under each image, we give the number of samples per pixel (spp), rendering time in seconds, and MSE. We provide consistently better MSE values, which we attribute to three main factors: the consideration of the color buffer to construct the final filter, the use of visibility features, and the improved handling of noisy features.

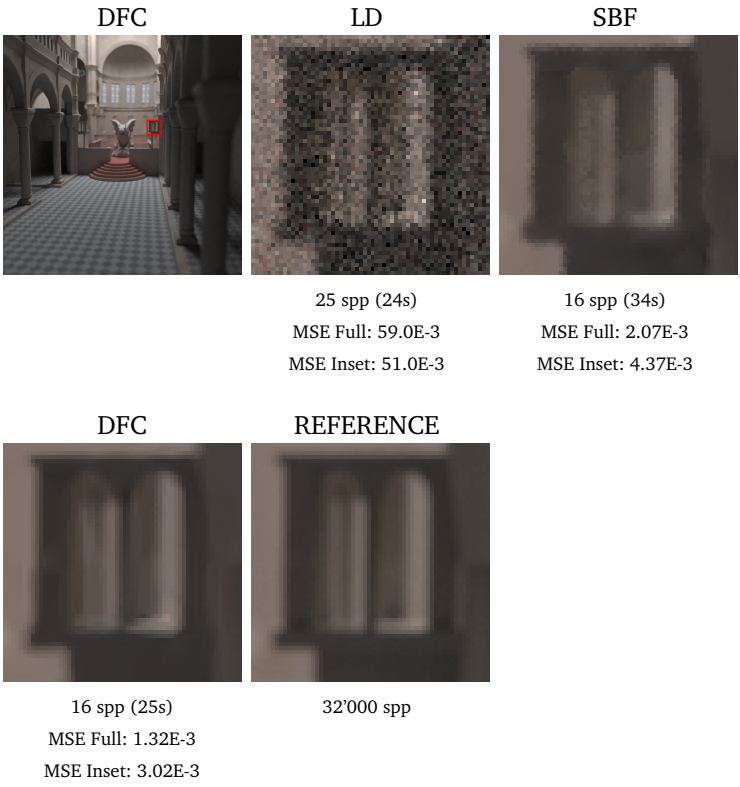


Figure 8.11: Renderings of the "sibenik" scene, comparing our approach (DFC) to path tracing with low-discrepancy (LD) and the SURE-based approach by Li et al. [LWC12] (SBF). Images are rendered at a resolution of 1024×1024 using PBRT at roughly equal render time. Under each image, we give the number of samples per pixel (spp), rendering time in seconds, and MSE. We provide consistently better MSE values, which we attribute to three main factors: the consideration of the color buffer to construct the final filter, the use of visibility features, and the improved handling of noisy features.

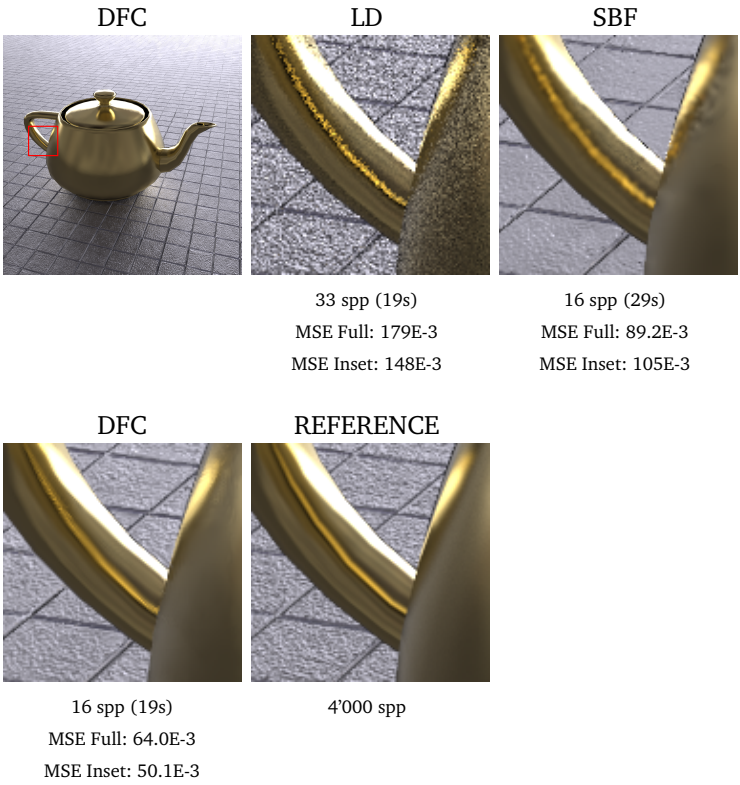


Figure 8.12: Renderings of the "teapot-metal" scene, comparing our approach (DFC) to path tracing with low-discrepancy (LD) and the SURE-based approach by Li et al. [LWC12] (SBF). Images are rendered at a resolution of 1024×1024 using PBRT at roughly equal render time. Under each image, we give the number of samples per pixel (spp), rendering time in seconds, and MSE. We provide consistently better MSE values, which we attribute to three main factors: the consideration of the color buffer to construct the final filter, the use of visibility features, and the improved handling of noisy features.

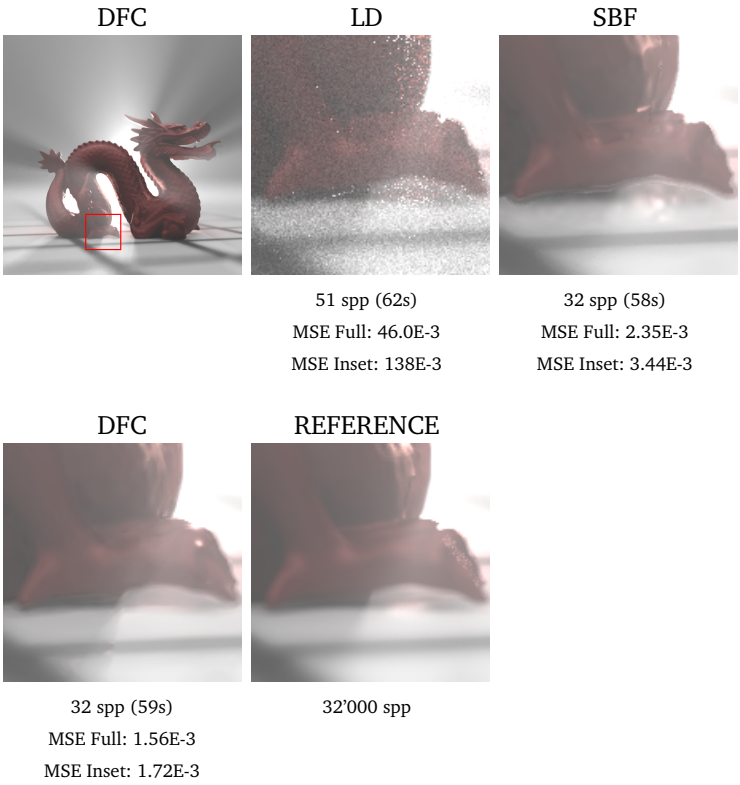


Figure 8.13: Renderings of the "dragonfog" scene, comparing our approach (DFC) to path tracing with low-discrepancy (LD) and the SURE-based approach by Li et al. [LWC12] (SBF). Images are rendered at a resolution of 1024×1024 using PBRT at roughly equal render time. Under each image, we give the number of samples per pixel (spp), rendering time in seconds, and MSE. We provide consistently better MSE values, which we attribute to three main factors: the consideration of the color buffer to construct the final filter, the use of visibility features, and the improved handling of noisy features.

Chapter 9

Conclusion

To conclude, we will reflect back on the initial problement of this thesis, and evaluate its main results in this light. We will then summarize the main avenues for further research.

In Chapter 1, when presenting our problem statement, we mentionned that we wanted to tackle the high computational cost of MCPT, while preserving its physically based nature, its ability at handling a wide array of light transport effects, and its consistency. Essentially, we wanted to preserve all appealing characteristics of MCPT, save for its unbiasedness.

Let us start by looking at the overall goal, lowering the computational cost of MCPT. While our focus was not on realtime, or even interactive, rendering, we still designed our framework to have a modest overhead. This lead to a framework operating in image space, and operating on pixels, rather than individual samples. This choice had the direct result of indeed offering a light computational overhead. For instance, the filtering step of our GEM algorithm would take a fraction of second once implemented on a GPU, and the filtering step of our DFC algorithm takes 5 seconds, which is negligeable for offline rendering. Our adaptive framework design also has the advantage of having a low memory overhead, which allow us to apply our filtering

techniques as a post-process, since we can easily store all data on the disk. In contrast, methods that operate directly on sample values scale badly to high sampling rates and are generally more intrusive to the rendering process. Lastly, the overhead of our framework depends only on the rendering resolution and filter window size, but not on the scene complexity.

Let us now look at the attributes of MCPT we wanted to preserve. First, MCPT's physically based nature is simply not affected by our adaptive framework, which is orthogonal to the rendering itself. Second, MCPT's ability at handling a wide variety of light transport effects is preserved thanks to the statistical nature of our approach. In our framework, each path contribution is considered as a generic sample, independently of how it was generated. The use of scene information in our DFC algorithm breaks somewhat this convention, but the various feature buffers are still handled in a uniform way, and the only feature specific aspect comes in the extraction of the feature itself. Third, MCPT's consistency is preserved by having the aggressiveness of the filtering step of our framework proportional to the variance in the rendering. As the sampling rate goes to infinity, the variance vanishes and our filter behaves as the identity transform, leaving the rendering untouched, which ensures the consistency of the overall approach.

While we did meet our initial goal, doing so constrained our framework design, which lead to some notable shortcomings in practice. The image space nature of our algorithm prevents us from leveraging high dimensional shearing patterns, such as those generated by motion blur. Similarly, operating on pixels instead of samples, prevents us from adapting our filtering step to conflicting constrains. Consider for instance the case of a focused scene observed through an out of focus fence. Ideally, we would want to filter heavily the out of focus fence, while using a much more conservative filter for the focused scene behind, but this is not possible in our current design. Lastly, our framework efficiency is highly dependant on the quality of the statistics gathered, in particular the per pixel variance of the rendering. This makes our framework ill suited at low sampling rates, where es-

timates of statistics such as variance are very unreliable. Lastly, some important perceptual cues, such as secondary shadows due to indirect illumination, often have a relatively small magnitude, which is easily drowned by noise and therefore harder to preserve.

Despite these shortcomings, our adaptive framework, in particular the DFC algorithm, is remarkably robust and efficient. Image space adaptive rendering has been surprisingly neglected for a long time, but saw a largely increased interest over the last five years. We found such techniques, and our own DFC algorithm in particular, to be particularly efficient and worthy of further investigation. We believe these techniques will greatly facilitate the deployment of MCPT and its derivatives in production environments, and be a key enabler of these techniques in interactive or realtime scenarios. For future work, we believe our adaptive framework could be extended to operate in an augmented space, where data is not processed in image space, nor directly in the sampling space, but in an alternative space that allows a better segmentation of data while still using a relatively low dimensionality to keep the computational overhead in control. We would also be interested in experimenting with state of the art image space filters that recently came out of the image processing community. Lastly, we believe that focus should be given to producing a new framework tailored specifically for video sequences, which could efficiently leverage the formidable data redundancy inherent in video.

Bibliography

[ARBJ03] Sameer Agarwal, Ravi Ramamoorthi, Serge Belongie, and Henrik Wann Jensen. Structured importance sampling of environment maps. *ACM Trans. Graph.*, 22(3):605–612, July 2003.

[BCM05] Antoni Buades, Bartomeu Coll, and Jean-Michel Morel. A review of image denoising algorithms, with a new one. *Multiscale Modeling & Simulation*, 4(2):490–530, 2005.

[BCM08] A. Buades, B. Coll, and J.M. Morel. Nonlocal image and movie denoising. *International Journal of Computer Vision*, 76(2):123–139, 2008.

[BEM11] Pablo Bauszat, Martin Eisemann, and Marcus Magnor. Guided image filtering for interactive high-quality global illumination. *Computer Graphics Forum*, 30(4):1361–1368, 2011.

[BM98] Mark R. Bolin and Gary W. Meyer. A perceptually based adaptive sampling algorithm. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, pages 299–309, New York, NY, USA, 1998. ACM.

[BPD06] Soonmin Bae, Sylvain Paris, and Frédo Durand. Two-scale tone management for photographic look. In *ACM*

Transactions on Graphics (TOG), volume 25, pages 637–645. ACM, 2006.

[BSS+13] 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.

[CCC87] Robert L. Cook, Loren Carpenter, and Edwin Catmull. The reyes image rendering architecture. *SIGGRAPH Comput. Graph.*, 21(4):95–102, August 1987.

[CD99] Emmanuel J Candès and David L Donoho. Ridgelets: A key to higher-dimensional intermittency? *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 357(1760):2495–2509, 1999.

[CETC06] David Cline, Parris K. Egbert, Justin F. Talbot, and David L. Cardon. Two stage importance sampling for direct lighting. In *Proceedings of the 17th Eurographics Conference on Rendering Techniques*, EGSR'06, pages 103–113, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[CFLB06] Per H Christensen, Julian Fong, David M Laur, and Dana Batali. Ray tracing for the moviecars'. In *Interactive Ray Tracing 2006, IEEE Symposium on*, pages 1–6. IEEE, 2006.

[CHM+12] Martin Cadík, Robert Herzog, RafałMantiuk, Karol Myszkowski, and Hans-Peter Seidel. New measurements reveal weaknesses of image quality metrics in evaluating graphics artifacts. *ACM Trans. Graph.*, 31(6):147:1–147:10, November 2012.

[Chr08] P Christensen. Point-based approximate color bleeding. *Pixar Technical Notes*, 2(5):6, 2008.

[CJAMJ05] 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.

[Coo86] Robert L. Cook. Stochastic sampling in computer graphics. *ACM Trans. Graph.*, 5(1):51–72, January 1986.

[CPC84] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. *SIGGRAPH Comput. Graph.*, 18(3):137–145, January 1984.

[CTCS00] 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.

[CWW+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.

[DD02] Frédo Durand and Julie Dorsey. Fast bilateral filtering for the display of high-dynamic-range images. In *ACM Transactions on Graphics (TOG)*, volume 21, pages 257–266. ACM, 2002.

[DFKE07] K.. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian. Image denoising by sparse 3-d transform-domain collaborative filtering. *Image Processing, IEEE Transactions on*, 16(8):2080–2095, 2007.

[DHS+05] 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.

[DJ95] David L Donoho and Iain M Johnstone. Adapting to unknown smoothness via wavelet shrinkage. *Journal of the american statistical association*, 90(432):1200–1224, 1995.

[Don99] David L Donoho. Wedgelets: Nearly minimax estimation of edges. *The Annals of Statistics*, 27(3):859–897, 1999.

[DSHL10] 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 the Conference on High Performance Graphics*, HPG '10, pages 67–75, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association.

[DW85] Mark A. Z. Dippé and Erling Henry Wold. Antialiasing through stochastic sampling. *SIGGRAPH Comput. Graph.*, 19(3):69–78, July 1985.

[ED04] Elmar Eisemann and Frédo Durand. Flash photography enhancement via intrinsic relighting. *ACM Trans. Graph.*, 23(3):673–678, August 2004.

[EDR11] 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.

[EHDR11] 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.

[EL99] A.A. Efros and T.K. Leung. Texture synthesis by non-parametric sampling. In *Computer Vision, 1999. The*

Proceedings of the Seventh IEEE International Conference on, volume 2, pages 1033–1038 vol.2, 1999.

[ETH+09] 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.

[FP04] Jean-Philippe Farrugia and Bernard Péroche. A progressive rendering algorithm using an adaptive perceptually based image metric. In *Computer Graphics Forum*, volume 23, pages 605–614. Wiley Online Library, 2004.

[GTGB84] Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, and Bennett Battaile. Modeling the interaction of light between diffuse surfaces. *SIGGRAPH Comput. Graph.*, 18(3):213–222, January 1984.

[HJW+08] 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(3):33:1–33:10, August 2008.

[HST10] Kaiming He, Jian Sun, and Xiaoou Tang. Guided image filtering. In *Proceedings of the 11th European Conference on Computer Vision: Part I*, ECCV'10, pages 1–14, Berlin, Heidelberg, 2010. Springer-Verlag.

[Jen96] Henrik Wann Jensen. Global illumination using photon maps. In *Rendering Techniques' 96*, pages 21–30. Springer, 1996.

[JSKJ12] Wojciech Jarosz, Volker Schönefeld, Leif Kobbelt, and Henrik Wann Jensen. Theory, analysis and applications of 2d global illumination. *ACM Trans. Graph.*, 31(5):125:1–125:21, September 2012.

[Kaj86] James T. Kajiya. The rendering equation. *SIGGRAPH Comput. Graph.*, 20(4):143–150, August 1986.

[Kat99] Vladimir Katkovnik. A new method for varying adaptive bandwidth selection. *IEEE Transactions on Signal Processing*, 47(9):2567–2571, 1999.

[KGPB05] Jaroslav Krivanek, Pascal Gautron, Sumanta Pattanaik, and Kadi Bouatouch. Radiance caching for efficient global illumination computation. *Visualization and Computer Graphics, IEEE Transactions on*, 11(5):550–561, 2005.

[KK02a] Thomas Kollig and Alexander Keller. Efficient multidimensional sampling. *Computer Graphics Forum*, 21(3):557–563, 2002.

[KK02b] Thomas Kollig and Alexander Keller. Efficient multidimensional sampling. *Computer Graphics Forum*, 21(3):557–563, 2002.

[KK03] Thomas Kollig and Alexander Keller. Efficient illumination by high dynamic range images. In *Proceedings of the 14th Eurographics Workshop on Rendering*, EGRW '03, pages 45–50, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[KS13] 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.

[KT09] Pierre Kornprobst and Jack Tumblin. *Bilateral filtering: Theory and applications*. Now Publishers Inc, 2009.

[LAC+11] 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, July 2011.

[LALD12] 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.

[LKL+13] Jaakko Lehtinen, Tero Karras, Samuli Laine, Miika Aittala, Frédo Durand, and Timo Aila. Gradient-domain metropolis light transport. *ACM Trans. Graph.*, 32(4):95:1–95:12, July 2013.

[LRR04] Jason Lawrence, Szymon Rusinkiewicz, and Ravi Ramamoorthi. Efficient brdf importance sampling using a factored representation. *ACM Trans. Graph.*, 23(3):496–505, August 2004.

[LW93] Eric P Lafortune and Yves D Willems. Bi-directional path tracing. In *Proceedings of CompuGraphics*, volume 93, pages 145–153, 1993.

[LWC+08] Y.L. Liu, J. Wang, X. Chen, Y.W. Guo, and Q.S. Peng. A robust and fast non-local means algorithm for image denoising. *Journal of Computer Science and Technology*, 23(2):270–279, 2008.

[LWC12] 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.

[McC99] Michael D. McCool. Anisotropic diffusion for monte carlo noise reduction. *ACM Trans. Graph.*, 18(2):171–194, April 1999.

[Mit87] Don P. Mitchell. Generating antialiased images at low sampling densities. *SIGGRAPH Comput. Graph.*, 21(4):65–72, August 1987.

[Mit91] Don P. Mitchell. Spectrally optimal sampling for distribution ray tracing. *SIGGRAPH Comput. Graph.*, 25(4):157–164, July 1991.

[Mit96] 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.

[MJL+13] Bochang Moon, Jong Yun Jun, JongHyeob Lee, Kunho Kim, Toshiya Hachisuka, and Sung-Eui Yoon. Robust image denoising using a virtual flash image for monte carlo ray tracing. *Computer Graphics Forum*, 32(1):139–151, 2013.

[MN88] Don P. Mitchell and Arun N. Netravali. Reconstruction filters in computer-graphics. *SIGGRAPH Comput. Graph.*, 22(4):221–228, June 1988.

[MWR12] Soham Uday Mehta, Brandon Wang, and Ravi Ramamoorthi. Axis-aligned filtering for interactive sampled soft shadows. *ACM Trans. Graph.*, 31(6):163:1–163:10, November 2012.

[MWRD13] 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.

[Nie92] Harald Niederreiter. *Quasi-Monte Carlo Methods*. Wiley Online Library, 1992.

[ODJ04] Victor Ostromoukhov, Charles Donohue, and Pierre-Marc Jodoin. Fast hierarchical importance sampling with blue noise properties. *ACM Trans. Graph.*, 23(3):488–495, August 2004.

[ODR09] Ryan S. Overbeck, Craig Donner, and Ravi Ramamoorthi. Adaptive wavelet rendering. *ACM Trans. Graph.*, 28(5):140:1–140:12, December 2009.

[Owe97] Art B Owen. Monte carlo variance of scrambled net quadrature. *SIAM Journal on Numerical Analysis*, 34(5):1884–1910, 1997.

[PBD+10] 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.

[PH10] Matt Pharr and Greg Humphreys. *Physically Based Rendering, Second Edition: From Theory To Implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition, 2010.

[PM90] Pietro Perona and Jitendra Malik. Scale-space and edge detection using anisotropic diffusion. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 12(7):629–639, 1990.

[PSA+04] 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.

[PSWS03] Javier Portilla, Vasily Strela, Martin J Wainwright, and Eero P Simoncelli. Image denoising using scale mixtures of gaussians in the wavelet domain. *Image Processing, IEEE Transactions on*, 12(11):1338–1351, 2003.

[RCL+08] Fabrice Rousselle, Petrik Clarberg, Luc Leblanc, Victor Ostromoukhov, and Pierre Poulin. Efficient product sampling using hierarchical thresholding. *The Visual Computer*, 24(7-9):465–474, 2008.

[RKZ11] 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.

[RKZ12] 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.

[RMZ13] Fabrice Rousselle, Marco Manzi, and Matthias Zwicker. Robust denoising using feature and color information. *Computer Graphics Forum*, 32(7):121–130, 2013.

[RPG99] Mahesh Ramasubramanian, Sumanta N. Pattanaik, and Donald P. Greenberg. A perceptually based physical error metric for realistic image synthesis. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '99, pages 73–82, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[RW94] Holly E. Rushmeier and Gregory J. Ward. Energy preserving non-linear filters. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, SIGGRAPH '94, pages 131–138, New York, NY, USA, 1994. ACM.

[SD12] 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.

[SDDVA+98] J. Sijbers, AJ Den Dekker, J. Van Audekerke, M. Verhoye, and D. Van Dyck. Estimation of the noise in magnitude mr images. *Magnetic Resonance Imaging*, 16(1):87–90, 1998.

[Sil86] B.W. Silverman. *Density estimation for statistics and data analysis*, volume 26. Chapman & Hall/CRC, 1986.

[SJJ12] Jorge Schwarzhaupt, Henrik Wann Jensen, and Wojciech Jarosz. Practical hessian-based error control for irradiance caching. *ACM Trans. Graph.*, 31(6):193:1–193:10, November 2012.

[SSD+09] Cyril Soler, Kartic Subr, Frédo Durand, Nicolas Holzschuch, and François Sillion. Fourier depth of field. *ACM Trans. Graph.*, 28(2):18:1–18:12, May 2009.

[Ste81] Charles M Stein. Estimation of the mean of a multivariate normal distribution. *The annals of Statistics*, pages 1135–1151, 1981.

[TM98] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Computer Vision, 1998. Sixth International Conference on*, pages 839–846, 1998.

[Vea97] Eric Veach. *Robust Monte Carlo methods for light transport simulation*. PhD thesis, Stanford University, 1997.

[VG95a] Eric Veach and Leonidas Guibas. Bidirectional estimators for light transport. In *Photorealistic Rendering Techniques*, pages 145–167. Springer, 1995.

[VG95b] 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.

[VG97] 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 Press/Addison-Wesley Publishing Co.

[WABG06] Bruce Walter, Adam Arbree, Kavita Bala, and Donald P. Greenberg. Multidimensional lightcuts. *ACM Trans. Graph.*, 25(3):1081–1088, July 2006.

[WBSS04a] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *Image Processing, IEEE Transactions on*, 13(4):600 –612, april 2004.

[WBSS04b] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: From error visibility to structural similarity. *Image Processing, IEEE Transactions on*, 13(4):600–612, 2004.

[WFA+05] Bruce Walter, Sebastian Fernandez, Adam Arbree, Kavita Bala, Michael Donikian, and Donald P. Greenberg. Lightcuts: A scalable approach to illumination. *ACM Trans. Graph.*, 24(3):1098–1107, July 2005.

[WH92] Gregory J Ward and Paul Heckbert. Irradiance gradients. In *Third Eurographics Workshop on Rendering*, volume 8598. Alan Chalmers and Derek Paddon, 1992.

[Whi79] Turner Whitted. An improved illumination model for shaded display. *SIGGRAPH Comput. Graph.*, 13(2):14–, August 1979.

[WOG06] Holger Winnemöller, Sven C Olsen, and Bruce Gooch. Real-time video abstraction. *ACM Transactions On Graphics (TOG)*, 25(3):1221–1226, 2006.

[WRC88] Gregory J. Ward, Francis M. Rubinstein, and Robert D. Clear. A ray tracing solution for diffuse interreflection. *SIGGRAPH Comput. Graph.*, 22(4):85–92, June 1988.

[XP05] R. Xu and S.N. Pattanaik. A novel monte carlo noise reduction operator. *Computer Graphics and Applications, IEEE*, 25(2):31–35, 2005.

Erklärung

gemäss Art. 28 Abs. 2 RSL 05

Name/Vorname: .....

Matrikelnummer: .....

Studiengang: .....

Bachelor ☐ Master ☐ Dissertation ☐

Titel der Arbeit: .....

.....

.....

LeiterIn der Arbeit: .....

.....

Ich erkläre hiermit, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Alle Stellen, die wörtlich oder sinngemäss aus Quellen entnommen wurden, habe ich als solche gekennzeichnet. Mir ist bekannt, dass andernfalls der Senat gemäss Artikel 36 Absatz 1 Buchstabe r des Gesetztes vom 5. September 1996 über die Universität zum Entzug des auf Grund dieser Arbeit verliehenen Titels berechtigt ist.

.....

Ort/Datum

.....

Unterschrift

Curriculum Vitae

Personal Information

Name	Fabrice Rousselle
Citizenship	Canadian, French
Date of birth	June 14, 1977
Place of birth	Montreal, Canada

Education

11/2009–2/2014 *Ph.D., Computer Science*
Computer Graphics Group, University of Bern
Bern, BE, Switzerland
Thesis : *Image Space Adaptive Rendering*
Adviser : Prof. Matthias Zwicker

1/2005–10/2007 *M.S., Computer Science*
University of Montreal
Montreal, QC, Canada
Thesis : *Hierarchical Product Sampling*
Advisor : Prof. Victor Ostromoukhov

11/1996–5/2000 *B.S., Computer Engineering*
Ecole Polytechnique de Montréal
Montreal, QC, Canada

Professional Experience

9/2007–9/2009 Research Assistant
Peripheral Systems Laboratory, Ecole Polytechnique Fédérale de Lausanne
Lausanne, VD, Switzerland

6/2000–12/2004 Programmer (C++) in the video group.
Matrox Electronic Systems Ltd.
Dorval, QC, Canada