Filtering Techniques for Low-Noise Previews of Interactive Stochastic Ray Tracing

Vom Fachbereich Informatik
der Technischen Universität Darmstadt
genehmigte

DISSERTATION

zur Erlangung des akademischen Grades
Doktor-Ingenieur (Dr.-Ing.)

von
Dipl.-Inf. Karsten Schwenk
geb. in Bad Kreuznach

Referenten der Arbeit:	Prof. Dr. techn. Dieter W. Fellner Technische Universität Darmstadt
	Prof. Dr.-Ing. Carsten Dachsbacher Karlsruher Institut für Technologie
Tag der Einreichung:	13.05.2013
Tag der Disputation:	05.07.2013

D17
Darmstadt 2013

Abstract

Progressive stochastic ray tracing algorithms are increasingly used in interactive applications such as design reviews and digital content creation. This dissertation contains three contributions to further advance this development.

The first contribution is a noise reduction method for stochastic ray tracing that is especially tailored to interactive progressive rendering. High-variance light paths are accumulated in a separate buffer, which is filtered by a high-quality, edge-preserving filter. Then a combination of the noisy unfiltered samples and the less noisy (but biased) filtered samples is added to the low-variance samples in order to form the final image. A novel per-pixel blending operator combines both contributions in a way that respects a user-defined threshold on perceived noise. For progressive rendering, this method is superior to similar approaches in several aspects. First, the bias due to filtering vanishes in the limit, making the method consistent. Second, the user can interactively balance noise versus bias while the image is rendering, leaving the possibility to hide filtering artifacts under a low level of dithering noise. Third, the filtering step is more robust in the presence of reflecting/refracting surfaces and high-frequency textures, making the method more broadly applicable than similar approaches for interactive rendering. The dissertation also contains some optimizations that improve runtime, recover antialiased edges, reduce blurring, and withhold spike noise from the preview images.

The second contribution is the radiance filtering algorithm, another noise reduction method. Again, the basic idea is to exploit spatial coherence in the image and reuse information from neighboring pixels. However, in contrast to image filtering techniques, radiance filtering does not simply filter

pixel values. Instead, it only reuses the incident illumination of neighboring pixels in a filtering step with shrinking kernels. This approach significantly reduces the variance in radiance estimates without blurring details in geometry or texture. Radiance filtering is consistent and orthogonal to many common optimizations such as importance, adaptive, and stratified sampling. In addition to the practical evaluation, the dissertation contains a theoretical analysis with convergence rates for bias and variance. It also contains some optimizations that improve the performance of radiance filtering on reflecting/refracting surfaces and highly glossy surfaces.

The last contribution of this dissertation is a system architecture for exchangeable rendering back-ends under a common application layer in distributed rendering systems. The primary goal was to find a practical and non-intrusive way to use potentially very different rendering back-ends without impairing their strengths and without burdening the back-ends or the application with details of the cluster environment. The approach is based on a mediator layer that can be plugged into the OpenSG infrastructure. This design allows the mediator to elegantly use OpenSG's multithreading and clustering capabilities. The mediator can also sync incremental changes very efficiently. The approach is evaluated with two case studies, including an interactive ray tracer.

Acknowledgments

There are many people who helped and supported me during the last years, when I was working on this dissertation.

First of all, I would like to express my gratitude to my supervisor Dieter Fellner for his support and the stimulating discussions. I am also grateful to my second assessor Carsten Dachsbacher and the other members of my reading committee.

Next, I would like to thank Arjan Kuijper, who has been my mentor throughout the whole process of preparing and writing this dissertation.

Then I would like to express thanks to my colleagues at Fraunhofer IGD for their advice and support, especially to Yvonne Jung, Gerrit Voß, and Tobias Franke. In addition to that, my immediate superiors Uli Bockholt and Johannes Behr have my gratitude for "keeping me on the long leash".

I also thank the developers of Elite, Wolfenstein, X-Wing, Doom, Quake and all the other games that kindled my interest in 3D computer graphics when I was young and impressionable. I doubt this dissertation would have been written without them.

Furthermore, I would like to thank the people and companies that provided the assets I used in this dissertation. The Dragon, Buddha, and Bunny models were provided by the Stanford 3D Scanning Repository; the Sponza scene was provided by Crytek GmbH and Marko Dabrovic; the Streets of Asia 2 scene was provided by Stonemason; the Powerplant model was provided by the GAMMA research group at UNC; the Viper model was provided by Chrysler Group and thecgizone.com; the Uffizi environment map was pro-

vided by Paul Debevec; some textures were provided by texturez.com and the Bonn BTF database.

Finally, “we” would like to thank the tradition of writing computer science theses in third person plural. It may sound slightly awkward in places, but in the end the “we” makes writing down the whole thing in a coherent style much easier.

Contents

List of Figures

List of Tables

1 Introduction

This chapter states the problem that motivated this dissertation. It also lists original contributions and relevant publications. Finally, it provides an overview of the remaining chapters.

The fields of interactive and offline photorealistic rendering are currently undergoing a dramatic change. With the advent of interactive ray tracing on consumer-grade hardware and in the cloud these two areas, mostly separated in the past, are gradually growing together. Stochastic ray tracing algorithms, in particular variants of path tracing [68], find their way into interactive rendering [18, 92, 65, 51]. These algorithms are based on a very realistic model of light transport and naturally capture many global illumination effects that were previously not available in interactive applications or only as very crude approximations (e.g. soft shadows, caustics, and interreflections).

Being essentially a Monte Carlo integration over the space of light paths (see Chapter 2), the beauty of these methods lies in their conceptual simplicity and generality. However, they can exhibit high variance in the estimator, which manifests itself as noise in the rendered image. The noise gradually disappears as more and more samples are averaged, but for interesting scenes convergence is still too slow to generate noise-free images at interactive rates. Therefore, interactive *progressive* rendering setups are typically used, in which the renderer can react instantaneously to discrete changes in the scene, but then has a few seconds to let the image converge.

If unbiased methods for reducing the variance such as importance sampling or stratification are not sufficient, one can try to further reduce noise by biasing the result. A common approach is to exploit spatial coherence in the image and reuse information of neighboring pixels or light paths by filtering. However, the relatively new combination of *interactive* and *progressive* stochastic ray tracing entails some unique requirements (detailed in the following section), which existing methods do not take into account sufficiently.

The primary goal of this dissertation is to develop robust and general noise reduction methods for interactive progressive stochastic ray tracing algorithms. Robust means the methods should work well for a wide range of input scenes. General means the methods should be compatible with a wide range of path sampling methods and common optimizations. The primary application of the methods is to provide fast, low-noise previews of global illumination.

Of course, noise reduction for stochastic ray tracing is not the only approach to rendering such previews. Progressive photon mapping [56, 52] and virtual point lights (VPLs) [71, 27] are other popular methods. We chose classic Monte Carlo path integration as a basis because of its generality and conceptual simplicity. The first means a wide range of scenes can be handled, the latter means efficient implementations on modern graphics hardware are possible.

In a recent study, Ou et al. [93] evaluated the potential of the aforementioned three classes of algorithms for preview rendering in appearance design tasks. The subjects preferred path tracing over progressive photon mapping and VPL rendering. For appearance design, they judged the high-frequency noise of path tracing to be less objectionable than the banding artifacts of VPL rendering or the extreme low-frequency noise of progressive photon mapping. Although *unfiltered* path tracing was used in the study, we like to think this study provides a retroactive empirical justification for our decision. At the very least it suggests that with careful noise reduction, path tracing would have performed even better.

1.1 Problem Statement

The merging of interactive and offline photorealistic rendering brings up two problems that motivated this dissertation.

Problem 1: Fast, reliable previews; consistent renderer. This is an algorithmic problem and the main motivation for our work. Applications such as interactive design reviews and digital content creation greatly benefit from photorealistic rendering with interactive feedback. However, as mentioned above, full global illumination simulations are expensive – and although interactive feedback is possible with current technology, truly interactive performance (> 10Hz) is still out of reach for interesting scenes. Therefore, techniques that can provide fast, reliable previews of global illumination are of much interest. At the same time, these techniques should take the special requirements of interactive progressive rendering into account. In particular:

- The methods should produce acceptable results as early as possible, with only a few samples per pixel.
- The user watches the image while it is being generated, this means the methods should continually provide updates.
- The system must remain responsive, that is it has to react to changes in less than a second.
- The rendering should still converge to the correct result, this means the estimator for each pixel should be consistent.
- The techniques should be compatible with common optimizations such as importance sampling and stratification.

Problem 2: System design for flexible rendering back-ends. This is a software engineering problem. As new rendering algorithms find their way into interactive systems, the need to support multiple, exchangeable rendering back-ends increases. This is an important concern for

generic distributed visualization packages. They typically have to support a wide range of applications (from visualization of large CAD models to photorealistic rendering) on a wide range of platforms (from CAVEs to Laptops). Yet, one usually wants to use the same application layer for all scenarios. These systems need practical and efficient ways to use potentially very different rendering back-ends without impairing their strengths and without burdening the back-ends or the application with details of the cluster environment.

This dissertation addresses the problem concretely for Fraunhofer's InstantReality [43] platform, a virtual/augmented reality system based on X3D [8]. This distributed system needs an extension to support specialized rendering back-ends (from GPU-rasterization to progressive ray tracing) under a common application layer (X3D).

1.2 Contributions

This dissertation makes the following contributions:

Filtering and blending with perceptual control. (Addresses Problem 1.) In Chapter 3, we present a practical noise reduction method for interactive progressive stochastic ray tracing. The radiance of high-variance light paths is accumulated in a separate buffer and filtered by a high-quality edge-preserving filter. A novel per-pixel blending operator combines the filtered and unfiltered pixels in a way that respects a user-defined threshold on perceived noise. The method can provide fast low-noise previews with only a few samples per pixel and is more robust than related approaches in the presence of complex features such as high-frequency textures and specular reflection/refraction. At the same time, it is consistent in that the bias due to filtering vanishes in the limit. This is a two-fold contribution, consisting of the perceptual-based blending operator and the adapted version of the cross bilateral filter.

Also in Chapter 3, we present four optimizations that improve the over-

all performance of the original approach and make it more broadly applicable. The primary target of these optimizations is the method described in Chapter 3, but some of the ideas and techniques are applicable to similar approaches. In particular, the antialiasing recovery step, the heuristic bandwidth adaption, and the spike noise removal step are valuable contributions.

Radiance filtering. (Addresses Problem 1.) In Chapter 4, we present another noise reduction method for interactive progressive path tracing. Instead of filtering pixel values, only the incident illumination of neighboring pixels is reused in a filtering step with shrinking kernels. The filter's bandwidth is adapted to reach a user-defined target variance. This approach significantly reduces the variance in radiance estimates without blurring details in geometry or texture. It also handles antialiased edges better and works naturally with distribution effects such as depth of field and motion blur. As an additional contribution, we provide a theoretical analysis of the method, including derivations of convergence rates for bias and variance.

Also in Chapter 4, we present a hybrid approach that combines the approaches described in Chapters 3 and 4. The hybrid filter performs better than the constituent approaches alone for scenes that contain both, perfect specular reflecting/refracting surfaces and non-specular surfaces. In addition, we present optimizations for highly glossy surfaces and sharp features in the indirect illumination (e.g. caustics), both of which cause problems with the original approach.

System architecture for flexible rendering back-ends. (Addresses Problem 2.) In Chapter 5, we describe a novel approach to using different rendering back-ends with a common application layer. The approach is based on a mediator layer that can be plugged into the OpenSG [32] infrastructure. The mediator and OpenSG provide multithreading and clustering capabilities as well as building blocks for efficient incremental updates. This way, the back-ends can retain their individual strengths and neither the back-ends nor the application layer are burdened with details of the cluster environment.

A related contribution are several X3D extensions for photorealistic rendering that are sketched in the appendices of Chapter 5. They provide X3D with a modern, physically-based material description that is portable across a wide range of rendering back-ends. These extensions are indicative of the growing overlap of photo-realistic and interactive rendering and the practical problems that arise from this development.

1.3 Publications

Large parts of this dissertation were already published as papers. The following publications are directly relevant (i.e. material was copied verbatim or with minimal editing).

[125] Schwenk, K., Kuijper, A., Behr, J., and Fellner, D. W. Practical noise reduction for progressive stochastic ray tracing with perceptual control. *IEEE Computer Graphics and Applications 32* (2012), 46–55.

[121] Schwenk, K., Behr, J., and Fellner, D. W. Filtering noise in progressive stochastic ray tracing – four optimizations to improve speed and robustness. *The Visual Computer 29*, 5 (2013), 359–368. Also appeared in *Proceedings of CGI 2012*.

[122] Schwenk, K., and Drevensek, T. Radiance filtering for interactive path tracing. In *ACM SIGGRAPH 2012 Posters* (New York, NY, USA, 2012), SIGGRAPH '12, ACM, pp. 109:1–109:1.

[119] Schwenk, K. Radiance filtering: Interactive low-noise previews of path traced global illumination, 2012. Submitted to *Computer Graphics Forum*, under review.

[126] Schwenk, K., Voss, G., Behr, J., Jung, Y., Limper, M., Herzig, P., and Kuijper, A. Extending a distributed virtual reality system with exchangeable rendering back-ends. *The Visual Computer* (2013),

1–11. Online first. This is an extended version of our CW2012 paper *A System Architecture for Flexible Rendering Back-ends in Distributed Virtual Reality Applications*.

[123] Schwenk, K., Jung, Y., Behr, J., and Fellner, D. W. A modern declarative surface shader for X3D. In *Proceedings of the 15th International Conference on Web 3D Technology* (2010), Web3D '10, ACM, pp. 7–16.

[124] Schwenk, K., Jung, Y., Voss, G., Sturm, T., and Behr, J. CommonSurfaceShader revisited: improvements and experiences. In *Proceedings of the 17th International Conference on 3D Web Technology* (New York, NY, USA, 2012), Web3D '12, ACM, pp. 93–96.

[120] Schwenk, K., Behr, J., and Fellner, D. W. CommonVolumeShader: simple and portable specification of volumetric light transport in X3D. In *Proceedings of the 16th International Conference on 3D Web Technology* (2011), Web3D '11, ACM, pp. 39–43.

Supplementary materials containing the original images and some videos for most of these papers can be found at `http://karsten-schwenk.de/papers/`. Especially the videos are worth a look, as they convey the interactive aspects of the techniques presented here much better than still images. Also, in some of the image comparisons the differences are hard to see in print and it is better to compare the original images on-screen (or view the PDF version of this dissertation).

1.4 Outline

The remainder of this dissertation is organized as follows.

Chapter 2 briefly reviews relevant background information on radiometry, light transport, and Monte Carlo rendering. That chapter is merely a refresher and introduces the most important symbols and equations. It

is by no means a complete reference and may be skipped by readers familiar with these topics.

Chapter 3 contains our first contribution: a method to reduce noise in stochastic ray tracing that is especially tailored to interactive progressive rendering. We show that this method can provide fast, reliable previews, even in the presence of complex features such as specular surfaces and high-frequency textures. At the same time, it is consistent in that the bias due to filtering vanishes in the limit. The chapter also describes a number of optimizations that make the method more robust and faster.

Chapter 4 contains another contribution: radiance filtering, a noise reduction method based on filtering incident radiance. In contrast to image filtering techniques, such as the one presented in Chapter 3, this method does not simply filter pixel values. Instead, it only reuses the incident illumination of neighboring pixels. We show that this approach significantly reduces the variance in radiance estimates without blurring details in geometry or texture. In a theoretical analysis, we derive convergence rates and compare the algorithm conceptually to related approaches. The chapter also discusses some optimizations.

Chapter 5 contains our final contribution: a system architecture for exchangeable rendering back-ends in distributed system. Focus is on the photorealistic ray tracer used in Chapters 3 and 4, but the architecture can (and does) support other renderers. We demonstrate results with two case studies.

Chapter 6 concludes the dissertation with a summary of contributions and findings. It also provides some directions for future work.

The observant reader may have noticed that there is no dedicated chapter about related work. Instead of providing one inflated chapter with related research for all chapters, each chapter includes a separate review of related work. This makes it easier to classify the contributions of each chapter individually and to discuss them in their respective contexts.

2 Background

This chapter provides a brief review of the physical and algorithmic background of this dissertation. It introduces the radiometric quantities necessary to formulate the light transport problem and describes how path tracing solves this problem by Monte Carlo integration.

The purpose of this chapter is twofold. First, we want to introduce the quantities, equations and concepts that are relevant for this dissertation. Second, we want to provide a more formal, mathematical view on the motivation for our work. To do this, we will first state the rendering problem as an unbiased Monte Carlo integration. Then, we will show that biasing the estimator can decrease the variance and the overall error. If such an estimator is consistent, it will still converge to the correct result, but because it may have a much lower error at the beginning, it can be used for fast previews.

The discussion is based on Eric Veach's dissertation [137] and we will mostly adopt his notation. Further Reading is discussed in Section 2.4, at the end of this chapter.

2.1 Radiometry

Radiometry is the field of study involved with the measurement of electromagnetic radiation, including, but not limited to, visible light. This section reviews the most important quantities for measuring light.

2.1.1 Domains and Measures

In the following, integrals over surfaces and solid angles will appear. Let $\mathcal{M} \subset \mathbb{R}^3$ be the union of all surfaces in the scene and $A(M), M \subseteq \mathcal{M}$ the usual surface area measure on $\mathcal{M}$. Further let $\mathcal{S}^2 \subset \mathbb{R}^3$ be the unit sphere and $\sigma(S), S \subseteq \mathcal{S}^2$ the usual surface area measure on $\mathcal{S}^2$ (the solid angle occupied by S). For a function $f : \mathcal{M} \times \mathcal{S}^2 \to \mathbb{R}$

$$I = \int_{\mathcal{M}} \int_{\mathcal{S}^2} f(x, \omega) \, \mathrm{d}\sigma(\omega) \mathrm{d}A(x) \tag{2.1}$$

is the Lebesgue integral of f with respect to solid angle and surface area.

Often the measure is clear from the context and one simply writes

$$I = \int_{\mathcal{M}} \int_{\mathcal{S}^2} f(x, \omega) \, \mathrm{d}\omega \mathrm{d}x. \tag{2.2}$$

For a given surface point $x \in \mathcal{M}$ with normal $n(x) \in \mathcal{S}^2$ the upward (or positive) hemisphere is defined as

$$\mathcal{H}^2_+(x) = \{\omega \in \mathcal{S}^2 | (\omega \cdot n(x)) > 0\}, \tag{2.3}$$

where $(\omega \cdot n(x))$ is the dot product in $\mathbb{R}^3$. Analogously, the downward (or negative) hemisphere consist of all directions facing away from the normal:

$$\mathcal{H}^2_-(x) = \{\omega \in \mathcal{S}^2 | (\omega \cdot n(x)) < 0\}. \tag{2.4}$$

2.1.2 Radiant Energy

Radiant energy is the energy carried by an electromagnetic wave (or photons). In particular, it is the total energy emitted by a light source over a given period of time. The symbol is Q and it is measured in Joules [J].

2.1.3 Radiant Power

Radiant power is radiant energy per unit time:

$$\Phi = \frac{\mathrm{d}Q}{\mathrm{d}t}, \tag{2.5}$$

usually measured in Watts [W = $^{\mathrm{J}}/_{\mathrm{s}}$]. In particular, it is the amount of radiant energy flowing through (or to or from) a surface (real or imaginary) per unit time. This is why it is also called *radiant flux*.

2.1.4 Irradiance and Radiant Exitance

Irradiance is incident radiant power per unit surface area. It is expressed in Watts per square-meter [$^{\mathrm{W}}/_{\mathrm{m}^2}$] and can be thought of as the area density of radiant flux. It is defined as

$$E(x) = \frac{\mathrm{d}\Phi(x)}{\mathrm{d}A(x)}. \tag{2.6}$$

Usually, one implicitly restricts Φ to the incident power from one hemisphere.

Radiant exitance (M) is the power leaving per unit surface area. The formula and units are the same as for irradiance, but with Φ restricted to the exitant power from one hemisphere, not the incident.

Radiosity (usually denoted B) is radiant exitance, but typically with the implicit assumption of an uniform energy distribution over the hemisphere. Using the definition of radiance (see below) this means the surface is a perfectly diffuse emitter and reflector producing uniform exitant radiance over the hemisphere.

2.1.5 Radiance

Radiance is radiant flux per unit projected area per unit solid angle. The definition in terms of radiant flux is:

$$L(x,\omega) = \frac{\mathrm{d}^2\Phi(x,\omega)}{\mathrm{d}\sigma(\omega)\,\mathrm{d}A(x)|\cos\theta|}, \tag{2.7}$$

where θ is the angle between the normal of $\mathrm{d}A(x)$ and the direction ω. The SI unit of radiance is Watts per steradian per square meter $\left[\mathrm{W}/\mathrm{sr\,m^2}\right]$. Intuitively, radiance can be thought of as the power traveling through a point x in direction ω. A nice property of radiance is that it remains constant along straight paths through empty space, so it is easy to propagate through a scene. This makes radiance the most important radiometric quantity in computer graphics.

When talking about radiance, one often prepends different qualifiers to make clear what fraction of radiance is meant in a given context. The most common ones are *incident radiance* $L_i(x,\omega)$ and *exitant radiance* $L_o(x,\omega)$, meaning the radiance arriving *from* direction ω or the radiance leaving *in* direction ω, respectively. This distinction primarily makes sense at surfaces. Here, L_i usually refers to incident flux ("photons just arriving") and L_o to exitant flux ("photons just leaving"), analogously to irradiance and radiant exitance. Exitant radiance is sometimes further divided into *reflected radiance* L_r and *self-emitted radiance* L_e.

2.1.6 Spectral Radiance

Spectral radiance (L_λ or more explicitly $L_\lambda(x,\omega,\lambda)$) represents radiant flux per unit projected area per unit solid angle *per unit wavelength*:

$$L_\lambda(x,\omega,\lambda) = \frac{\mathrm{d}^3\Phi(x,\omega,\lambda)}{\mathrm{d}\sigma(\omega)\,\mathrm{d}A(x)|\cos\theta|\,\mathrm{d}\lambda}. \tag{2.8}$$

It has units $\mathrm{W}/\mathrm{sr\,m^3}$.

In computer graphics, one often uses pre-integrated spectra (colors) and works with three separate radiances instead of the full spectrum. This is not correct, but for most use cases the error is deemed negligible compared to the gain in runtime [113]. For an RGB color space, the "color radiances" are

$$L_c = \int_\Lambda m_c(\lambda) L_\lambda(\lambda)\,\mathrm{d}\lambda \quad c \in \{r, g, b\}, \tag{2.9}$$

for some set of color matching functions m_c and a spectrum of wavelengths Λ. Usually, the subscript is dropped and one implicitly assumes three color channels when referring to a radiance.

2.2 Light Transport

Now that the fundamental quantities for measuring light have been introduced, we can review how light propagates through a scene. Photorealistic rendering in the sense of this dissertation *is* physically based simulation of light transport. Usually linear geometrical optics is used as the underlying physical model, with some extensions toward wave optics (e.g. for interference). We will also assume a simplified scene model here. In particular, light only interacts with the scene directly at surfaces, where interaction means emission, absorption, or scattering. In other words, there is no participating medium and no subsurface scattering or similar effects.

2.2.1 Bidirectional Scattering Distribution Function

The *bidirectional scattering distribution function* (BSDF) of a surface describes the relationship between incoming and scattered light. More formally, it is the ratio of the differential radiance scattered in direction ω_o to the differential irradiance incident from direction ω_i at a surface point x:

$$f_s(x, \omega_i, \omega_o) = \frac{\mathrm{d}L_o(x, \omega_o)}{\mathrm{d}E(x, \omega_i)} = \frac{\mathrm{d}L_o(x, \omega_o)}{L_i(x, \omega_i)\,|\cos\theta_i|\mathrm{d}\omega_i}. \tag{2.10}$$

Intuitively, f_s describes the "throughput" of a surface point for a pair of directions.

A special case of BSDF that only describes reflection (ω_i and ω_o in the same hemisphere) is the bidirectional reflectance distribution function (BRDF, f_r). Another special case is the bidirectional transmittance distribution function (BTDF, f_t), which describes transmission (ω_i and ω_o in different hemispheres). So, the BSDF of a surface can be split into two BRDF/BTDF pairs (one for each hemisphere).

There are a number of properties that are expected from physically valid BSDFs (the most important one being energy conservation) but we will not go into details here. Furthermore, there exist a number of generalizations for effects beyond simple reflection and transmission (e.g. subsurface scattering) [103].

To make the direction of light flow more explicit, one often writes $L_i(x \leftarrow \omega_i)$, $f_s(\omega_i \rightarrow x \rightarrow \omega_o)$, and $L_o(x \rightarrow \omega_o)$, a notation we will use for the following transport equations.

2.2.2 Local Scattering

With the BSDF, we can calculate the exitant (scattered) radiance due to the incident radiance from all directions. This quantity is given by

$$L_o(x \rightarrow \omega_o) = \int_{\mathcal{S}^2} f_s(\omega_i \rightarrow x \rightarrow \omega_o)\, L_i(x \leftarrow \omega_i) |\cos\theta_i|\, \mathrm{d}\omega_i. \tag{2.11}$$

This integral describes the appearance of a surface under a particular lighting condition. If restricted to the upper hemisphere and the BRDF, this equation becomes the *reflection equation* (or *local reflectance integral*).

2.2.3 Rendering Equation

To model general radiance transport, we have to add the emitted radiance L_e. Also, the exitant radiance of one point is potentially part of the incident radiance at other points (indirect illumination). Thus extended, Equation 2.11 becomes the *rendering equation* [68, 64]:

$$L(x \to \omega_o) = L_e(x \to \omega_o) + \int_{\mathcal{S}^2} f_s(\omega_i \to x \to \omega_o) L(x \leftarrow \omega_i) |\cos \theta_i| \, \mathrm{d}\omega_i. \quad (2.12)$$

This recursive integral equation describes the radiance function of a scene (the *light field*).

The equation given here is actually a special case of a more general *light transport equation* with certain restrictions and boundary conditions [103, Ch. 16].

2.2.4 Measurement Equation

Working with radiance is practical for simulating light propagation, but in the end we usually need to calculate a sensor response (make a measurement in the virtual scene). The sensor can be a pixel in a digital image, an area on a simulated chemical film, or similar concepts. The general *measurement equation* for a sensor is

$$R = \int_{I \times \mathcal{S}^2} W_e(x \leftarrow \omega) L_i(x \leftarrow \omega) \, \mathrm{d}x \, \mathrm{d}\omega, \quad (2.13)$$

where I is the surface of the sensor and W_e is the sensitivity (also called *emitted importance*, importance is the adjoint quantity of radiance; remember that the arrow indicates the direction of light flow, so importance is emitted opposite to the arrow's direction). Note that we use the "color radiance" here. With spectral rendering, we would have to integrate over wavelengths against a spectral sensitivity in addition to the integral over surface and solid angle. With time-dependent sensitivity or incident radiance, another integral over (exposure) time would be necessary.

2.2.5 Path Integral Formulation

The rendering equation in the form given above is a surface-centric formulation of the problem: it describes light transport as recursive scattering events at surface points. Veach [137, Ch. 8] introduced a more elegant formulation, which combines the measurement equation and the rendering equation into a single non-recursive integral. In the so-called *path integral formulation*, each measurement is expressed as an integral over the space of all possible light transport paths:

$$R = \int_{\mathcal{X}} f(\bar{x}) \, \mathrm{d}\mu(\bar{x}), \tag{2.14}$$

where $\mathcal{X}$ is the set of transport paths of finite length, μ is a measure on $\mathcal{X}$, and f is the *measurement contribution function*. Each path is a sequence of vertices that are surface points:

$$\bar{x} = x_0 x_1 \cdots x_k \quad x_i \in \mathcal{M}. \tag{2.15}$$

In theory, the path length is not bounded, that is $0 < k < \infty$. In practice, however, we usually have to impose some upper limit. The measurement contribution function returns the contribution a path makes to the final measurement. The exact definitions and derivations are not particularly complicated but a bit lengthy, which is why we omit them here and refer to Veach [137, Ch. 8] for details.

2.3 Monte Carlo Rendering

Equation 2.14 states the light transport problem as an integral over a high-dimensional space with an integrand that may contain discontinuities and singularities. The method of choice for such problems is Monte Carlo integration. We will quickly review the concept and show how it is applied to light transport.

2.3.1 Monte Carlo Integration

The basic idea of Monte Carlo integration is to recast the integral as the expected value of a random variable. Given the integral

$$I = \int_{\mathbb{R}} f(x)\,\mathrm{d}x, \tag{2.16}$$

we can define the estimator

$$\hat{I} = \frac{f(X)}{p(X)}, \tag{2.17}$$

with X distributed according to a probability density function (pdf) p that is non-zero whenever f is non-zero (so p does not miss any regions that contribute to the integral).

The expected value of this estimator is

$$\mathrm{E}\left[\hat{I}\right] = \int_{\mathbb{R}} \frac{f(x)}{p(x)} p(x)\,\mathrm{d}x = I. \tag{2.18}$$

According to the strong law of large numbers, the sample average will converge to the expected value (with probability one). So, to compute the integral, we can repeatedly sample the estimator and average:

$$I \approx \bar{\hat{I}}_N = \frac{1}{N}\sum_{i=1}^{N} \frac{f(X_i)}{p(X_i)}, \tag{2.19}$$

with X_i distributed according to p. Sometimes $\hat{I}$ is called the *primary estimator* and $\bar{\hat{I}}_N$ the *secondary estimator*.

2.3.2 Path Sampling

With the little review above, the elegance of the path integral formulation immediately becomes apparent. We can directly apply Monte Carlo integration to Equation 2.14. For each measurement (e.g. a pixel), an estimator can

be defined:

$$\bar{\hat{R}}_N = \frac{1}{N} \sum_{i=1}^{N} \frac{f(\bar{X}_i)}{p(\bar{X}_i)} \approx R = \int_{\mathcal{X}} f(\bar{x}) \, \mathrm{d}\mu(\bar{x}). \tag{2.20}$$

To compute an image, we simply generate random light paths according to some sampling strategy, evaluate their contributions $f(\bar{X})$, and average them (weighted by the pdf p).

Different path sampling strategies result in different algorithms. The most important ones are

- *path tracing* [68, 103] (construct paths starting from camera),
- *light tracing* [3, 38] (construct paths starting from lights),
- *bidirectional path tracing* [138, 80, 103] (construct paths from both sides),
- *Metropolis light transport* [139, 103] (construct paths by perturbing already sampled paths), and
- *instant radiosity* [71, 38] (construct many light paths in pre-process, connect eye paths of length 1 to all vertices of all light paths).

This is only a top-level classification, each algorithm can be subdivided further on the basis of sampling strategies. A complete description and classification is not the subject of this dissertation and we refer to Veach [137] and Pharr & Humphreys [103] for further details.

Some methods allow fuzzy connections between paths, the most popular one is *(progressive) photon mapping* [66, 55]. These methods can efficiently sample paths that classic path sampling cannot handle very well, e.g. caustic paths (so called SDS paths in Heckbert's notation [60]; see Veach [137] and Hachisuka [53]). They can be integrated into the path sampling framework with a little extension [57, 48].

A way to approach the light transport problem without Monte Carlo is to

divide the scene (and sometimes the unit sphere) into discrete patches. This transforms the rendering equation (Eq. 2.12) into a system of linear equations, which can be solved with the usual methods. This is an application of the finite element method to rendering and the idea behind the classic radiosity methods. However, these algorithms typically impose some severe restrictions and have increasingly fallen out of favor with rendering engineers, at least if generality is a major concern.

2.3.3 Unbiased and Consistent Estimators

Important concepts for this dissertation are unbiasedness and consistency. In general, an estimator approximates an unknown quantity with N samples:

$$I \approx \bar{\hat{I}}_N(X_1, X_2, \ldots, X_N). \tag{2.21}$$

The error of this approximation is

$$\bar{\hat{\epsilon}}_N = \bar{\hat{I}}_N - I. \tag{2.22}$$

The expected value of the error $\mathrm{E}\left[\bar{\hat{\epsilon}}_N\right]$ is called bias. If $\mathrm{E}\left[\bar{\hat{\epsilon}}_N\right] = 0$ the estimator is called *unbiased*. An alternative formulation is to say the expected value of the estimator is the quantity we want to compute. Loosely speaking, an unbiased estimator has no systematic error, that means the error is only due to variance and averages out.

An estimator is said to be *consistent*, if it converges in probability to the quantity we want to compute as the sample count approaches infinity:

$$\operatorname*{plim}_{N\to\infty} \bar{\hat{I}}_N = I. \tag{2.23}$$

Note that this does not require unbiasedness, but it requires that the bias vanishes in the limit (with probability one). So, there may be a systematic error, but it can be made arbitrarily small by investing more samples. It should be mentioned that an estimator can have vanishing bias and still be

not consistent. In fact, an estimator can be unbiased and not consistent. For example, if we used only the first sample (X_1) in Eq. 2.19 to estimate the integral, the expected value would still be the integral I. But the estimator would not converge to its expected value. However, if the variance *and* the bias vanish simultaneously, the estimate converges and the estimator is consistent.

In rendered images, variance typically manifests itself as noise. (Imagine several pixel estimators looking at the same point: the variance in the estimator will lead to a different estimate for each pixel, producing a noisy image.) Bias typically takes the form of blurring, banding, or similar artifacts. (Imagine several pixel estimators that reuse samples from their neighbors: each estimator may be unbiased, but "pulling in" samples from neighbors may change the expected value and thus introduce bias (blurring).)

In general, we want at least consistent estimators in photorealistic rendering, that means estimators that converge to the correct result, but may be biased. Nevertheless, unbiased estimators have some advantages over consistent estimators. The biggest advantage is that we can directly estimate the error from the variance of the estimator, since the variance is the only source of error. This is explored in the following subsection.

2.3.4 Variance Reduction and Expected Error

The expected squared error (MSE) of an estimator can be decomposed into bias and variance:

$$\underbrace{\mathrm{E}\left[\hat{\epsilon}^2\right]}_{\text{MSE}} = \underbrace{\mathrm{E}\left[\hat{\epsilon}\right]^2}_{\text{squared bias}} + \underbrace{\mathrm{Var}\left[\hat{\epsilon}\right]}_{\text{variance}}. \tag{2.24}$$

This is a rearrangement of the well-known relation

$$\mathrm{Var}\left[X\right] = \mathrm{E}\left[X^2\right] - \mathrm{E}\left[X\right]^2 \tag{2.25}$$

and called *bias variance decomposition* or *bias variance trade-off*. So, the expected squared error can be reduced by reducing variance or bias.

Let us look at the variance of an estimator of the form

$$\bar{\hat{I}}_N = \frac{1}{N}\sum_{i=1}^{N}\hat{I}_i. \tag{2.26}$$

If the variances of all $\hat{I}_i$ are equal and finite, the variance of $\bar{\hat{I}}_N$ decreases linearly with N:

$$\mathrm{Var}\left[\bar{\hat{I}}_N\right] = \mathrm{Var}\left[\frac{1}{N}\sum_{i=1}^{N}\hat{I}_i\right] = \frac{1}{N^2}\sum_{i=1}^{N}\mathrm{Var}\left[\hat{I}_i\right] = \frac{1}{N}\mathrm{Var}\left[\hat{I}\right]. \tag{2.27}$$

As stated above (Eq. 2.24), for unbiased estimators the variance is the only source of error, so the expected (RMS) error decreases as the standard deviation with a rate of $O\left(N^{-1/2}\right)$. Variance reduction without introducing bias obviously leads to faster convergence. The classic unbiased variance reduction techniques are importance sampling, correlated sampling (control variates), and stratification.

However, as is evident from Equation 2.24, we can balance bias and variance to some extent. It is often possible to reduce variance significantly by allowing a certain amount of bias. If the bias is relatively low and the variance reduction is relatively high, we can reach a lower overall error faster with biased techniques.

This is the essence of the algorithms presented in this dissertation: to trade variance for bias in order to produce fast previews with low error. We also require consistency to ensure the methods converge to the correct result eventually.

2.4 Further Reading

Of course, we have only scratched the surface in this chapter. In large parts, this chapter is a condensed version of several chapters of Eric Veach's dissertation "Robust Monte Carlo Methods for Light Transport Simulation"

[137, Ch. 1,2,3,8]. This is a good reference for unbiased Monte Carlo rendering based on path sampling. A more practice-oriented approach is taken by Matt Pharr and Greg Humphreys in their book "Physically Based Rendering: From Theory to Implementation" [103], which cannot be recommend highly enough.

Henrik Wann Jensen's book "Realistic Image Synthesis Using Photon Mapping" [66] is a good reference on photon mapping, a biased Monte Carlo technique. For modern variants of photon mapping we refer to the work by Toshiya Hachisuka and his colleagues, in particular the recent SIGGRAPH course "State of the Art in Photon Density Estimation" [52]. These methods are particularly interesting in the context of this dissertation since they too try to achieve faster convergence by trading variance for bias. A very recent work with potential toward preview-rendering is adaptive progressive photon mapping [70].

An overview of other global illumination techniques, with focus on radiosity, is given in "Advanced Global Illumination" [38] by Philip Dutré, Kavita Bala, and Philippe Bekaert.

The SIGGRAPH courses "Practical Physically Based Shading in Film and Game Production" [85] and "Global illumination Across Industries" [77] provide an overview of techniques used in production rendering.

Good references for *interactive* stochastic ray tracing on GPUs are Jacco Bikker's PhD Thesis [14] and Dietger van Antwerpen's Master thesis [136]. A general survey of interactive methods with focus on real-time is given by Ritchel et al. in their state-of-the-art report [110].

A nice survey of importance sampling techniques is "A Survey of Importance Sampling Applications in Unbiased Physically Based Rendering" [135] by Dietger van Antwerpen. Another worthwhile reference is the SIGGRAPH course "Importance Sampling in Production Rendering" [26].

For Quasi Monte Carlo methods have a look at the work by Alexander Keller and his colleagues. The SIGGRAPH course "Advanced (Quasi) Monte Carlo Methods for Image Synthesis" [73] is a good entry point.

Readers interested in the physics of light transport may want to look at Eugene Hecht's aptly titled book "Optics" [59]. For an extensive treatment of light as an electromagnetic wave see "Principles of Optics" [17] by Max Born and Emil Wolf.

3 Filtering Pixel Radiance

This chapter presents a noise reduction method based on filtering image pixels. The radiance of high-variance light paths is accumulated in a separate buffer and filtered by a high-quality edge-preserving filter. A novel per-pixel blending operator combines the filtered and unfiltered pixels according to a user-defined threshold on perceived noise.

3.1 Introduction

Using filtering to reduce noise in images rendered by Monte Carlo techniques has a long history in computer graphics. Several approaches exist, but they are all based on the idea of exploiting inter-pixel coherence by averaging the estimates for adjacent pixels. In general, including neighbors into a pixel estimator will not only have the desired effect of reducing variance, but will also change the expected value. So, usually filtering means trading noise for bias.

Until recently, most techniques have been developed with offline systems in mind, which use the filter in one final pass to clean up a rendered image. These approaches are usually designed to operate on relatively low noise levels and are not directly applicable to interactive renderers, where the filter has to be applied from the beginning of rendering. With the advent of interactive stochastic ray tracing, methods aimed at real-time performance were developed. These approaches usually prefer speed over quality and their filters may introduce blur or other artifacts in difficult scenes with high-frequency textures and (nearly) perfect specular objects.

We present a noise reduction method that combines filtering with blending, using a user-supplied decomposition of path space into high-variance and low-variance paths. The key idea is to maintain two separate buffers for high-variance paths. One buffer contains the unbiased original paths and one contains a filtered version that is biased, but has reduced variance. Then a simple self-adapting blending operator adds the optimal amount of both buffers to the contribution of the low-variance paths to form the final image. Here "optimal" means the highest amount of unbiased contribution that is possible while respecting a user-defined threshold on perceived noise. Early in the rendering process, the blending operator will assign high weights to the filtered buffer, thus producing an image with a low noise level but bias. As more samples are gathered, the unbiased buffer will become more reliable and take over. So, the method introduces bias, but is consistent (the bias will vanish eventually). For the filtering step, we use an adapted cross bilateral filter that is able to produce a low-noise image without visible blur in many scenarios. The combination of this filter and the blending operator can present a high-quality image to the viewer after only a few samples per pixel.

The two key contributions of this chapter are the development of the perceptual-based blending operator and the adaption of the cross bilateral filter. In addition, we present some optimizations for the original method which improve the overall performance. The new method is easy to implement and has acceptable overhead per frame.

Throughout this dissertation, we will refer to the specific method presented here as "pixel filtering", mainly to emphasize the conceptual difference to "radiance filtering", which is presented in the next chapter. To refer to the general class of algorithms that are based on filtering image pixels, we will use "image filtering".

3.2 Related Work

Reducing the noise in images produced by Monte Carlo techniques has been a research subject for a long time. Popular methods include stratification, importance sampling, control variates, and adaptive sampling. Colbert et al. [26] provided a recent survey. In contrast to these techniques, our work belongs to the class of approaches that filter the resulting image to remove noise and do not alter the underlying sampling process.

3.2.1 Filtering for Monte Carlo Noise Reduction

Lee and Redner [81] advertised the use of nonlinear median and alpha-trimmed mean filters to reduce spike noise in rendered images. These filters can produce nearly noise-free images, but can also introduce artifacts and heavily bias the result. For example, the median filter is not energy-preserving and can offset edges [86] or shift colors [29]; similarly, the alpha-trimmed mean simply removes offending "outliers", although they are carrying a valid contribution. Jensen and Christensen [67] applied non-linear filters only to the indirect diffuse component of the image to reduce these artifacts. We follow their insight and take it one step further: we only filter light paths that have been classified as high-variance paths by the user (e.g. only indirect illumination or only caustic paths).

A relatively straightforward technique is to locally adapt the bandwidth of filters in order to balance smoothing and preservation of high-frequency features. Rushmeier and Ward [117] introduced a class of energy-preserving, non-linear filters that spread out high-variance samples with a variable-width kernel. The adaptive kernel width reduces blurring artifacts, but since the shape of the kernel itself does not respect edges, high-variance samples may still be spread out over a large area and cause undesirable blurring.

McCool [86] investigated the use of anisotropic diffusion for Monte Carlo noise reduction. Anisotropic diffusion preserves edges and energy. The main drawback of the method is its incremental nature, which makes it expensive

to compute and requires a carefully chosen stopping criterion. It is also not immediately clear how to apply the technique to progressive rendering (other than recomputing all iterations every frame).

Suykens and Willems [132] adapted the idea of splatting samples with a variable-width kernel to progressive rendering. This means they applied their filter not as a post process but during image formation. The method is similar in spirit to Rushmeier and Ward's approach [117] and suffers from similar problems. In particular, the blurring of high-variance regions is objectionable (although as the number of samples increases the kernel size is reduced and the blurring will eventually vanish).

Xu and Pattanaik [149] applied a variant of bilateral filtering to the indirect illumination in order to reduce noise. They used a Gauss-filter as a pre-process to spread out visual outliers and then used the smoothed values as reference values for each pixel. (But the values for the gathered pixels in the range kernel were still taken from the original image, which is an important difference to cross bilateral filtering. In that regard, the method lies in between the classic bilateral filter and the cross bilateral filter.) They applied their filter only as a post-process to clean up images with relatively low noise levels, which permitted a narrow spatial kernel and made the algorithm comparatively fast. The method works well for untextured scenes where direct illumination dominates, but tends to blur textures and geometric edges in areas where indirect illumination is strong. In addition, filtering higher noise levels requires larger kernels and significantly more processing time (the bilateral filter is not separable).

DeCoro et al. [29] developed a method to specifically remove spike noise. The idea is to withhold samples with high energy but low probability (one could say perceived outliers) from the image, so that existing filtering methods can be used to remove the remaining noise. Pajot et al. [96] recently presented a similar approach based on density estimation. These techniques are orthogonal to ours and combining their outlier rejection with pixel filtering could be an interesting direction for future work. (Pixel filtering spreads out extreme perceived outliers; sometimes it may be desirable to remove or withhold them, see Section 3.6.4.)

With the advent of interactive stochastic ray tracing, methods aimed at real-time performance started to emerge. In an early effort, Keller [72] and Wald et al. [142] used a discontinuity buffer for geometry-aware filtering of interleaved sampling patterns. More recently, Dammertz et al. [28] used an edge-avoiding À-Trous wavelet transform to filter noisy images from an interactive path tracer. Edges were defined using multiple edge stopping functions (normals, positions, direct illumination), similar to cross bilateral filtering. But the À-Trous transform allowed them to compute large kernels in real-time. They achieved impressive results and geometric edges seem to be preserved very well in general, but the filter can blur across high-frequency texture details. (Especially for non-diffuse scenes, when the deferred rendering approach described in the paper cannot be used.) Furthermore, the À-Trous scheme can produce objectionable ringing artifacts; and aliasing in the geometry buffers can produce artifacts, too.

In work developed parallel to ours, Bauszat et al. [7] introduced a very interesting and fundamentally different approach to illumination filtering. Instead of filtering a noisy illumination estimate using geometry information as guidance, they used a guided image filter [58] to locally fit the noise-free geometry information (normals and depth) to the illumination estimate. This led to a very fast algorithm that produces results comparable to (sometimes even superior to) cross bilateral filtering in terms of mean square error. The biggest limitation of the method is that is does not respect edges that are not present in the geometry buffers. How the method performs for textured glossy surfaces, where filtering irradiance becomes less viable, is not shown in the paper. Also, it is not clear how reflecting and refracting objects can be handled, as only results for (nearly) diffuse surfaces are shown.

Very recently, after our original paper [125] was accepted, Chen et al. [23] introduced a screen-space statistical filtering method based on incremental PCA that exploits temporal coherence. We have not yet had the opportunity to evaluate this approach, but the noise-reduction seems to be on par with bilateral filtering. However, it is intended for flicker-free interactive (even real-time) animations, not progressive rendering, and is not consistent.

The main difference between these filtering methods and our method (pixel filtering) is that pixel filtering does not display the filtered buffer directly, but blends it with the original samples in a way that tries to respect a user-defined noise level. This has several advantages: First, pixel filtering remains consistent if used with a consistent progressive renderer, because as the noise level in the unbiased buffer decreases the weights for filtered (biased) buffer will approach zero. Second, pixel filtering can leave a controlled amount of equally distributed low-amplitude, high-frequency noise in the image, which helps masking filtering artifacts. Third, pixel filtering does not have to filter *every* presented frame, because new samples contribute via the unbiased buffer, even though they are not yet in the filtered buffer. This allows us to use the relatively expensive bilateral filter *during* image creation. This filter results in very strong noise reduction with good edge-preservation and acceptable overhead from the beginning of image formation, which is important for progressive rendering.

These properties of pixel filtering reflect the fact that it was designed with a slightly different application area in mind: interactive progressive rendering. In contrast to related approaches, pixel filtering does not try to present a noise-free image "at all costs" (i.e. possibly with severe filtering artifacts). Instead, the objective is to deliver a reliable preview of the image as early as possible – with a carefully chosen balance of noise and bias.

3.2.2 Bilateral Filtering

Pixel filtering uses an adapted cross bilateral filter. The bilateral filter is a technique to smooth images while preserving important edges proposed by Aurich and Weule [4], Smith and Brady [129], and Tomasi and Manduchi [134]. The filter weights for each pixel are computed using the distance in the spatial domain *and* the distance in the range domain. Usually two Gaussians are used as the weighting functions. The metric in the spatial domain is the typically the Euclidian distance and in the range domain some intensity or color difference is used.

The cross (or joint) bilateral filter separates the image defining the edges from

the image to be smoothed. Using a separate "range image" (sometimes called "edge image", although it does not contain the edges directly) is useful if the image to be smoothed has been heavily corrupted by noise. Then the classic bilateral filter cannot discern real edges and outliers due to noise. Eisemann and Durand [41] and Petschnigg et al. [101] applied this idea to "flash no-flash" image pairs, where the "flash" image provides the edges to smooth the "no-flash" image. Kopf et al. [76] used the cross bilateral filter in a general upsampling framework.

Direct implementations of the bilateral filter are comparatively slow, because it is not a linear convolution with a constant kernel and not separable. Several acceleration techniques exist, some of the approaches mentioned here were evaluated for the optimizations in Section 3.6.1.

The bilateral grid was proposed by Chen et al. [22] and generalized previous approaches by Durand and Dorsey [35] and Paris and Durand [97, 98]. The basic idea is to recast the bilateral filter as a linear convolution in a low-resolution higher-dimensional space (space $\times$ intensity) followed by a non-linear *slicing* step that extracts the approximately bilaterally filtered data. The biggest weakness of the method is that it poorly scales with increasing range dimensions – often only a 3d grid with two spatial and one range dimension (typically intensity) is practical. The idea was recently generalized to the Gaussian kd-tree by Adams et al. [1]. Gaussian kd-trees scale better with increasing range dimensions and are more accurate for large spatial standard deviations, but generally slower.

Weiss [146] described a histogram-based acceleration technique, but the spatial weighting is restricted to a step function and the technique can lead to color bleeding [97]. Pham and van Vliet [102] applied a one-dimensional bilateral filter first horizontally and then vertically, effectively treating it as if it was separable. The technique is very fast and works surprisingly well for small spatial kernels (3-5 pixels), but for larger kernels the artifacts quickly become too objectionable.

Porikli [104] and Yang et al. [151] proposed fast bilateral filters that run in constant time per pixel, regardless of the kernel size. Some more interest-

ing approaches were published after the work described in this chapter was done [46, 5]. We have not evaluated these approaches with pixel filtering, but they should be compatible.

3.2.3 Other Edge-preserving Filters

Finally, we want to briefly discuss other edge-preserving filters that we considered for pixel filtering, but eventually dismissed.

Anisotropic diffusion was already mentioned. It was proposed as an image filter by Perona and Malik [100] and later extended to robust anisotropic diffusion by Black et al. [15]. Although it is very flexible, we did not choose this method because of its iterative nature and unintuitive parameters. However, anisotropic diffusion is closely related to bilateral filtering as was pointed out by Barash [6] and Buades et al. [21].

Felsberg et al. [42] described channel smoothing, a technique similar to the bilateral grid based on a B-Spline encoding of the data. We have not directly compared the two, but the bilateral grid seemed conceptually simpler, faster, and altogether better suited to a GPU-implementation.

Choudhury and Tumblin [24] proposed a new *trilateral* filter that incorporates local image gradients and smoothes towards a piecewise linear image (the bilateral filter smoothes towards a piecewise constant image). The method should provide better noise reduction in high-gradient regions, but since we did not know of any effective acceleration techniques, we chose the bilateral filter.

3.3 Method

Before we begin, a note on terminology: in a general setting pixels measure radiant energy. However, because pixel filtering does not work well with depth of field and motion blur, we will assume a pinhole camera with instantaneous

L_{lv}	radiance of low-variance paths
L_u	unfiltered radiance of high-variance paths
L_f	filtered radiance of high-variance paths
L_{hv}	combined (blended) radiance of high-variance paths
s	blend factor for combining unfiltered and filtered pixels
t	user-defined threshold on variance
$\mathrm{Var}_{L_b}[\cdot]$	perceived variance wrt. background radiance L_b
$u = \mathrm{Var}_{L_b}[L_u]$	perceived variance of unfiltered buffer
$f = \mathrm{Var}_{L_b}[L_f]$	perceived variance of filtered buffer
a_i	pixel weights for filtering step
σ_c	std. dev. for colors in cross bilateral filter
σ_d	std. dev. for depths in cross bilateral filter
σ_s	std. dev. for spatial distance

Table 3.1: *Quick reference for the most important symbols used in this chapter.*

exposure and a high-resolution pixel grid. This means pixel values can be regarded as averaged radiance over a small area and solid angle, and we will formulate pixel estimates as radiance estimates. Table 3.1 provides an overview of the most important symbols used in this chapter.

Figure 3.1 shows a schematic of pixel filtering. As was already outlined in the introduction, we split light paths into two categories which are accumulated in two separate buffers. Paths that potentially have high variance go into the high-variance buffer (L_u), and those that have a comparatively low variance go into the low-variance buffer (L_{lv}). This path classification is specified by the user before the rendering starts. Typically, the low-variance buffer holds the direct illumination and the high-variance buffer the indirect illumination, but other classifications are possible. For example, one could only classify caustic paths as high-variance paths.

Furthermore, we maintain an additional buffer that holds a filtered version of the high-variance buffer. So there are in fact two buffers for high-variance paths: one holds the original unfiltered radiance L_u, the other one a filtered version, L_f. To get the final contribution of high-variance paths for a pixel, L_{hv}, we combine the two with a simple linear blend:

$$L_{hv} = sL_u + (1 - s)L_f. \tag{3.1}$$

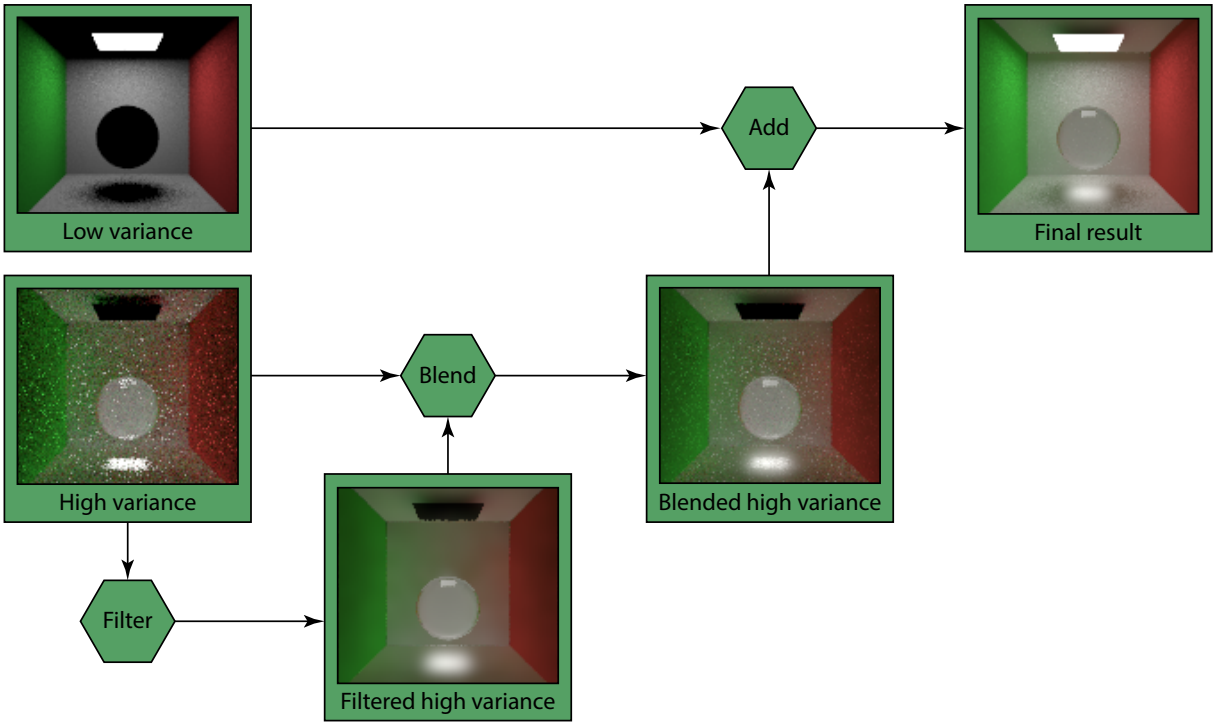

Figure 3.1: *Schematic data flow of pixel filtering. The user classifies light paths into those that potentially carry high variance and those that carry low variance. These two classes of paths are accumulated in separate buffers. The high-variance buffer is filtered by an edge-preserving filter to reduce noise. The blending operator mixes the filtered and unfiltered versions of the high-variance buffer (attempting to reach a user-defined noise level) and adds the result to the low-variance buffer.*

Conceptually, the blend factor s balances an unbiased but noisy contribution (L_u) with a biased but less noisy one (L_f). Given a threshold on the noise level, we want the largest possible contribution of L_u and blend in just enough of L_f to get below the threshold. The remainder of this section deals with how to find s.

3.3.1 Perceived Variance

At most adaption levels, the ratio of a just noticeable difference (JND) ΔL_{JND} against some background radiance L_b is constant (about 1% under good viewing conditions). This relationship is known as Weber's law [16, 107], the ratio

is the Weber fraction K:

$$K = \frac{\Delta L_{\text{JND}}}{L_b}. \tag{3.2}$$

Let $\hat{L}$ be an estimate for a real radiance L. If the error in the estimator is in the order of a JND, we can define the perceived error with respect to a background radiance L_b as

$$\epsilon_{L_b} = \frac{\hat{L} - L}{L_b}. \tag{3.3}$$

The same absolute error triggers a larger response in the visual system when viewed against a low background radiance than it does when viewed against a high background radiance.

For unbiased estimators, this error is due only to the variance in the estimator. (Of course the renderer may be biased, too, but bias in the renderer is not addressed by our algorithm and we simply ignore it.) We define the perceived variance as a measure for the perceived noise level as

$$\text{Var}_{L_b}[\hat{L}] := \text{Var}[\epsilon_{L_b}] = \frac{1}{L_b^2}\text{Var}[\hat{L}]. \tag{3.4}$$

For a given background radiance, this perceived variance can be seen as a function that scales the absolute variance according to Weber's law. Similar measures have been used by Mitchell [89] and Hachisuka et al. [54]. It should be noted that the variance of the estimator is a measure for the *expected* amount of noise, not the actual amount, and that in practice *estimates* of variances are used (see Section 3.3.2).

We would also like to point out that, although we will be using the perceived variance as defined above as a measure for the perceived noise level, only the last step of our algorithm (Sec. 3.3.5) uses this measure. All other components are based on the absolute variance and other functions could be used to map absolute variance to a perceived noise level in the last step. For example, we could include characteristics of the tone mapping operator into

the metric. Our perceived variance assumes a perfect tone mapping operator, in the sense that it perfectly recreates the sensation a viewer would experience when watching the real scene. More precisely: the perceived error when watching the image equals the perceived error when watching the real scene. An advantage of our perceived variance is that it makes it easy to specify thresholds, because it can be thought of as the variance of a relative noise corrupting the signal.

3.3.2 Variances of the Buffers

In order to find the blend factor in Equation 3.1, the per-pixel variances of both buffers, filtered and unfiltered, must be estimated.

$\mathrm{Var}[L_u]$, the variance in the unfiltered buffer, can be estimated with a method for incremental variance calculation while the image is rendering. In practice, however, a problem arises because at the beginning of rendering the estimated variance will not be very reliable, and estimates of neighboring pixels can differ greatly, although they have very similar variances. If the blend factors are computed per pixel using these variance estimates, the blend factors may differ greatly from pixel to pixel and noise will creep back into the blended image – due not to the variance in the filtered radiance buffer but to the variance in the per-pixel blend factors. To prevent this, we filter the variance values along with the radiance values in the filtering step (described in the following section) and use this smoothed variance as $\mathrm{Var}[L_u]$. Variances and radiances use the same range buffer (i.e. the same weights), so they can be computed in one pass.

$\mathrm{Var}[L_f]$, the variance of the filtered pixels, depends on the filter kernel. To estimate the variance of filtered pixels we will assume that neighboring pixels are independent realizations of the same random variable. In practice, neighboring pixels will represent different variables, but usually they are very similar so this is a reasonably approximation (although it will break down in the presence of hard edges, see Section 3.3.3). Under this assumption, a filter can reduce variance by computing a weighted sum of independent realizations of the same random variable (the pixels inside the kernel). The variance of a

filtered pixel is

$$\mathrm{Var}[L_{f,j}] = \mathrm{Var}\left[\sum_i a_i L_{u,i}\right] \overset{(a)}{\approx} \left(\sum_i a_i^2\right) \mathrm{Var}[L_{u,j}], \tag{3.5}$$

where the a_i are the filter weights and the $L_{u,i}$ are the radiances of the pixels inside the kernel. Step (a) in Equation 3.5 uses the approximation described above to estimate the variance of a filtered pixel only with knowledge of its own variance. For reasonable filters $|a_i| < 1$ and $\sum_i a_i = 1$ hold, so the term $\sum_i a_i^2 < 1$ will reduce variance. A wide, strongly blurring filter with equally distributed weights will reduce variance better then a narrow filter with weights concentrated around the center.

We use an adapted version of the cross bilateral filter (described in the following section), but it should be noted that the blending operator does not depend on a particular choice for the filter. As long as an estimate for the variance reduction can be computed, any filter can be used.

3.3.3 The Filter

The assumption made in the previous section that neighboring pixels are independent realizations of the same random variable breaks down in the presence of discontinuities (edges) in the image. This offers a probabilistic view on blurring artifacts: they are the failed attempt to reduce variance by incorporating samples into an estimate that were drawn from a distribution with a different expected value (and thus bias the estimator). To ensure that pixels whose expected values differ from the current pixel's do not contribute significantly, we use a bilateral filter.

Unfortunately, the classic bilateral filter is not directly applicable to the scenario at hand, because, especially at the beginning of image formation, the noise level is too high and the range domain kernel cannot reliably identify pixels with similar expected values. (Remember that the samples in the buffer have been explicitly classified as potential high-variance samples.) Therefore, we use a cross bilateral filter to decouple the data defining

Figure 3.2: *Left: a test scene containing reflecting/refracting surfaces. Middle: color range buffer generated by rendering with a single unoccluded head light and ambient light. It captures normals and textures (even in reflections/refractions). Right: depth range buffer containing geometric edges.*

the edges (the range buffer) from the data being filtered (the high-variance buffer).

The range buffer should exhibit the edges of the image to be filtered but be (almost) noise-free. Furthermore, its computation should involve little overhead. At first sight, the low-variance buffer (all samples not classified as high-variance) seems suitable, but this approach has two problems. Consider the typical case where the low-variance buffer contains the direct illumination and the high-variance buffer the indirect illumination. In shadowed regions, the low-variance buffer will be completely black. So exactly in the regions where the indirect illumination is likely to make a large contribution, no edges would be defined. Conversely, hard direct shadows would define edges that are irrelevant to indirect illumination. The second problem is that even direct illumination can contain spike noise in the presence of difficult to sample area light sources.

We chose to explicitly generate a separate range buffer. For each viewing ray, we store the hemispherical reflectivity of the first non-specular surface it hits (perfect specular surfaces reflect and refract rays Whitted-style). This roughly corresponds to an image rendered with only unoccluded ambient lighting and captures the edges present in textures. In addition, we light the scene with a single unoccluded head light. This produces most of the edges that result from variation in the normals. We believe this to be a simple but quite elegant solution, because lighting with a headlight conveys information

similar to that in a normal buffer (as far as edges are concerned), but avoids the aliasing problems and the storage costs of an additional buffer. Especially in the presence of realistic specular surfaces (that reflect and refract at the same time) or alpha blended surfaces, pixels can be a linear combination of many reflected and refracted light paths. It is not easy to decide what to store as the normal in these cases. High-frequency normal maps can make it even worse. The only geometry information we use directly is the depth of the first surface a primary ray hits, in order to resolve objects with similar normals and textures that are partially occluding each other. The depth is stored in the alpha channel of the range buffer. Figure 3.2 shows a test scene and the corresponding range buffer.

Note that this range buffer generated with a single headlight and an ambient term can be regarded as an artificially generated "flash image", in the sense of the "flash no-flash" pairs of Eisemann and Durand [41] and Petschnigg et al. [101]. Recently, after our paper was published, Moon et al. [90] used the same idea for noise reduction and coined the term "virtual flash image", which is actually quite appropriate.

Generating the range image usually has little overhead: No shadow rays have to be traced, no lighting calculations have to be done (apart from the diffuse response to the single unshadowed head light). If one has access to the underlying renderer, even the hit points of the primary rays can be reused, and the range image can be computed alongside the other buffers. To antialias the range buffer, we usually accumulate 4 samples per pixel per filter pass, this means the first filter pass uses a range buffer with 4x stochastic supersampling, the second pass 8x, and so on. For depths, we keep the nearest depth per pixel, although this produces slightly jagged edges.

Now we have all the components to define our cross bilateral filter and calculate by which amount it reduces the variance of a pixel. The filter is defined

as

$$L_{f,j} = \frac{1}{A} \sum_i a_i L_{u,i}, \tag{3.6}$$
$$\text{with} \quad a_i = g_{\sigma_s}(||p_0 - p_i||) g_{\sigma_c}(||c_j - c_i||) g_{\sigma_d}(|d_j - d_i|)$$
$$\text{and} \quad A = \sum_i a_i.$$

$L_{f,j}$ is the value of the filtered pixel, $g_\sigma(t) = \exp\left(-\frac{1}{2}\frac{t^2}{\sigma^2}\right)$ is a Gaussian with zero mean and standard deviation σ (σ_s for the spatial domain, σ_c for colors, and σ_d for depths), p_i are the pixel positions, c_i and d_i are the colors and depths in the range buffer, and $L_{u,i}$ are the entries of the unfiltered high-variance buffer. We use the Euclidean distance in pixel space and linear sRGB color space (although other color differences may produce better results at the expense of additional computation).

Compared to the unfiltered pixel, the variance of the filtered pixel is approximately reduced by a factor of $r = \sum_i a_i^2$ (Eq. 3.5). We calculate this factor during the filtering process and store it in the alpha channel of the filtered image.

Applying a wide bilateral filter is an expensive operation. However, with our approach it is not necessary to apply the filter every frame. With a wide filter new samples have little influence on the filtered image, because (in most regions) the filter has already averaged a lot of similarly distributed samples. So, the filtered image does not change much, even during the early frames. In addition, new samples *do* contribute via the unfiltered buffer, even if they are not yet present in the filtered buffer. This allows pixel filtering to apply the filter only every 2^ith frame; usually starting with frame 4. With this progression, even a brute-force implementation of the bilateral filter becomes practical. Further speed-up can be achieved with the optimizations described in Section 3.6.

3.3.4 Variance of the Blended Result

With estimates for $\mathrm{Var}[L_u]$ and $\mathrm{Var}[L_f]$, we can estimate the variance of the blended result:

$$\begin{aligned}\mathrm{Var}[L_{hv}] &= \mathrm{Var}\left[sL_u + (1-s)L_f\right] \\ &= s^2\mathrm{Var}[L_u] + (1-s)^2\mathrm{Var}[L_f] + 2s(1-s)\mathrm{Cov}[L_u, L_f]. \quad (3.7)\end{aligned}$$

The covariance is not zero, because (in general) L_u is a part of the filtered pixel L_f. It depends on the filter kernel and the radiance values:

$$\begin{aligned}\mathrm{Cov}[L_u, L_f] &= \mathrm{E}[L_u L_f] - \mathrm{E}[L_u]\,\mathrm{E}[L_f] \\ &= \mathrm{E}\left[L_{u,0}\sum_i a_i\,\mathrm{E}[L_{u,i}]\right] - \mathrm{E}[L_u]\,\mathrm{E}\left[\sum_i a_i L_{u,i}\right] \\ &= \sum_i \mathrm{E}[L_{u,0}a_i L_{u,i}] - \mathrm{E}[L_u]\,\mathrm{E}\left[\sum_i a_i L_{u,i}\right] \\ &\overset{(a)}{\approx} a_0\,\mathrm{E}[L_{u,0}^2] + \sum_{i>0} a_i\,\mathrm{E}[L_{u,i}]^2 - \mathrm{E}[L_u]\,\mathrm{E}\left[\sum_i a_i L_{u,0}\right] \\ &= a_0\left(\mathrm{E}[L_{u,0}^2] - \mathrm{E}[L_{u,0}]^2\right) \\ &= a_0\mathrm{Var}[L_{u,0}] \quad (3.8)\end{aligned}$$

where a_0 and $L_{u,0}$ are the weight and the radiance of the central pixel. $\mathrm{E}[\cdot]$ is the expected value. The approximation (a) uses the assumption that neighboring pixels are independent realizations of the same random variable. Note that the covariance tends to increase with a large central weight or a central spike in variance, as one would intuitively expect. Equation 3.8 can be used to estimate $\mathrm{Cov}[L_u, L_f]$, but for the broad filters we use a_0 is quite small (typically < 0.1). We usually set it to zero in practice, sacrificing a small amount of accuracy for runtime. (If the filter is not able to gather many similar pixels in the neighborhood, the covariance and the error in the blend factor will increase, but in this case the filtered and the original pixel will be very similar. In that case, the error in the blend factor is acceptable, because we are blending very similar values.)

3.3.5 The Blend Factor

In order to find s in Equation 3.1, we have to rewrite Equation 3.7 in terms of perceived variance

$$\mathrm{Var}_{L_b}[L_{hv}] = \frac{1}{L_b^2}\mathrm{Var}\left[sL_u + (1-s)L_f\right], \tag{3.9}$$

and need to find the background radiance L_b. Although we use a global tone mapping operator (Reinhard's photographic operator [107]) using the global adaption luminance proved to be a poor choice. In our current implementation, we use the sum of the filtered radiance of the high-variance buffer and the radiance of the low-variance buffer:

$$L_b = L_f + L_{lv}. \tag{3.10}$$

This way, we can compute L_b in the filtering pass without much additional overhead. Another option is to calculate L_b from the local adaption luminance of a local tone mapping operator, but we have not tried this yet.

With Equation 3.9, finding a blend factor that satisfies some user-defined threshold on perceived variance

$$\mathrm{Var}_{L_b}[L_{hv}] \leqslant t \tag{3.11}$$

consists simply of solving a quadratic equation for s. Note that in practice an *estimate* for $\mathrm{Var}_{L_b}[L_{hv}]$ is used, so the real variance remaining in the image may be larger.

The solution that maximizes the unbiased contribution (assuming $\mathrm{Cov}[L_u, L_f] = 0$) is

$$s' = \frac{f + \sqrt{tu + tf - uf}}{u + f}, \tag{3.12}$$

$$\text{with} \quad u = \mathrm{Var}_{L_b}[L_u] \quad \text{and} \quad f = \mathrm{Var}_{L_b}[L_f]$$

The actual blend factor requires handling two special cases:

$$s = \begin{cases} 0 & \text{if } tu + tf - uf < 0 \quad \text{(case 1)} \\ 1 & \text{if } u \leqslant t \quad \text{(case 2)} \\ s' & \text{otherwise.} \quad \text{(case 3)} \end{cases} \tag{3.13}$$

In case 1, s' is not a real number, this means it is not possible to satisfy the threshold because even the filtered buffer has too high a variance. In case 2, the unfiltered buffer already fulfills the threshold, this means the filtering step is unnecessary. (Note that if $u = f = 0$ case 2 applies; if $t = 0$ and $u > 0$ case 1 applies; so $s(u)$ is well-defined for $u \geqslant 0$, $t \geqslant 0$.)

The definition of s clearly shows that the bias due to filtering vanishes in the limit, if the unfiltered image converges (and $t > 0$). At some point, the variance of the unfiltered buffer will drop below the threshold ($u \leqslant t$). This is case 2 in Equation 3.13, so $s = 1$ and only the unfiltered buffer is used, which does not contain bias. Appendix 3.A contains a formal analysis of the blend factor.

3.4 Results

In this section, we show results and discuss several important aspects of pixel filtering. We implemented pixel filtering for a CUDA-based progressive path tracer build on top of the OptiX ray tracing engine [99]. If not otherwise stated, all images and numbers were produced with this configuration: NVIDIA GeForce GTX 470 1.2GB; 768×768 resolution; $\sigma_s = 8$ pixels, $\sigma_c = 0.02$, $\sigma_d = 2m$; spatial kernel truncated after $3\sigma_s$; next event estimation; maximum path length 6 vertices, Russian roulette starts after 3; one path per pixel traced per presented frame; box filter as pixel reconstruction filter. Since we want to evaluate filtering performance, we filter exitant radiance, even for diffuse surfaces, where one could use irradiance filtering ([75], [28], [7]) to preserve the textures better. Chapter 4 shows and discusses filter variants that do that.

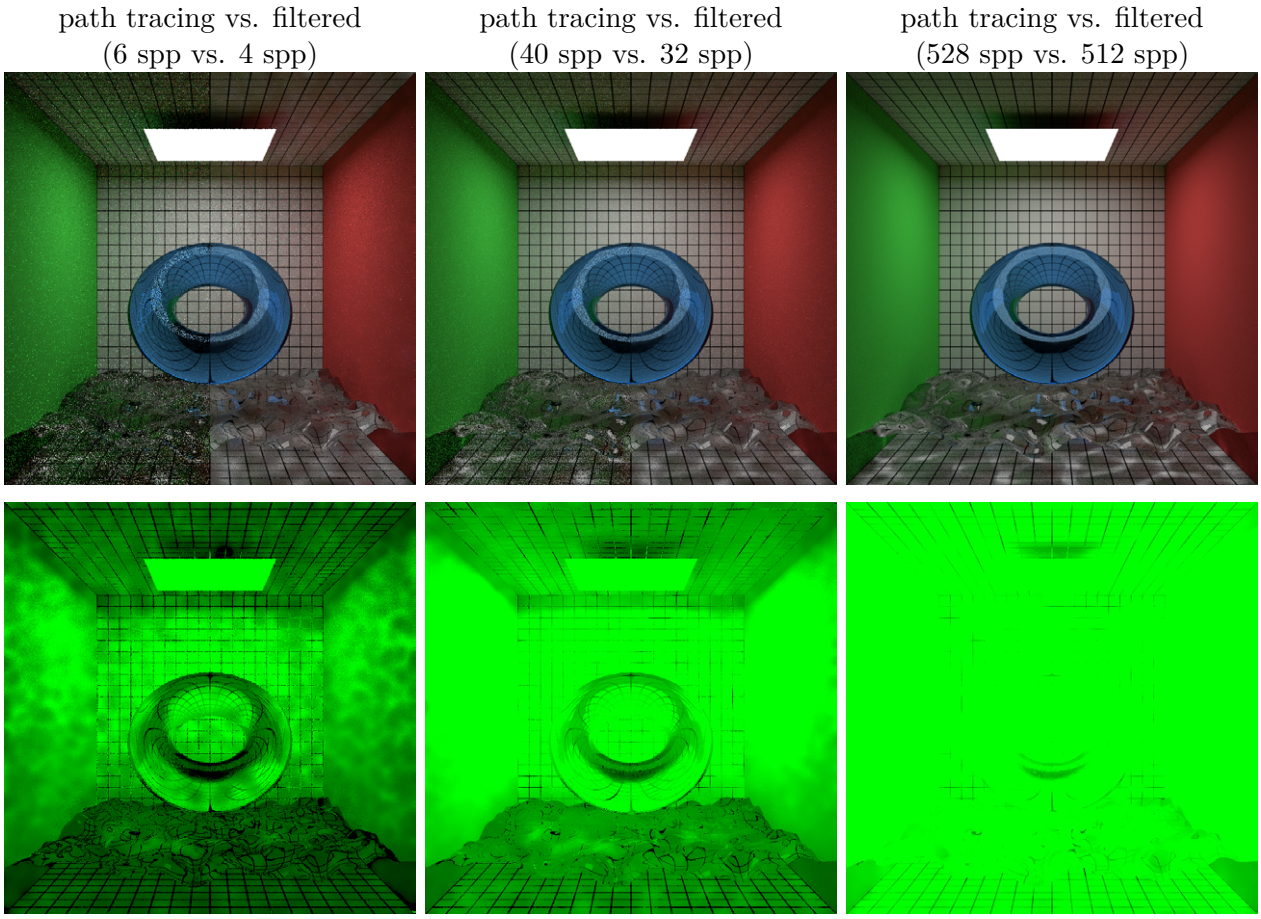

Figure 3.3: *Side-by-side comparison of unfiltered path tracing (left half of each image) and pixel filtering (our method, right half). Rendering times are comparable, $t = 0.1$. Top: final images with a different number of samples per pixel (spp). Bottom: corresponding per-pixel blend factors.*

Figure 3.3 shows a typical application of pixel filtering to a progressive rendering. The two perfect specular objects lead to a high variance in the indirect illumination, resulting in a very noisy image in regions where indirect light dominates (top and bottom). Pixel filtering can provide a strongly reduced noise level very early in rendering process, after 4 samples per pixel (spp). The original path traced image is hardly recognizable after a comparable rendering time (6 spp). Applying a classic bilateral filter to this image would yield no usable results, but our adapted cross bilateral filter can smooth the image. Note that reflected and refracted edges on the specular objects are preserved, too; only caustics are blurred. After 32 spp, pixel filtering reaches a noise level comparable to 200 spp of unfiltered path tracing in the high-

variance areas. The walls and the back of the box have comparatively low variance and the blending operator can already assign a high weight to the unfiltered buffer after 32 paths in these areas. Note that the method does not try to remove all visible noise in this case, it attempts to maintain a uniform noise level just below a user-specified threshold. As a result, the formerly blurred caustics are becoming sharper. After 512 paths the unfiltered buffer has reached the threshold almost everywhere and the blending operator has effectively switched completely to the original path traced image.

Figure 3.4 shows results for three distinct scenes. A scene with clearly defined edges and perfect specular objects causing caustics, a scene with textures (including normal and specular maps) dominated by indirect illumination, and a scene lit by partially occluded area lights. The threshold used in these images corresponds approximately to a standard deviation of 1 JND, so the blending operator tries to remove almost all noise. Again, pixel filtering achieves a strong noise reduction, but it cannot satisfy the threshold everywhere. Although the Cornell box scene has clearly defined edges, the filter has problems to find similar neighbors for antialiased pixels directly on those edges. In the Sponza scene, the filter has the complementary problem: it blurs across the very fine variations in the texture of the stones (but clear edges in the textures are very well preserved). In the kitchen scene, the filter works very well, but much of the variance in that image is due to direct illumination, which is not included in the high-variance buffer and therefore not smoothed. The filtering artifacts can be addressed by fine-tuning the filter parameters, but this is tedious and sometimes it is not possible to find parameters that are optimal for all possible views of a scene. However, a nice property of pixel filtering is that filtering artifacts can be hidden to some extend by specifying a more generous threshold, which leaves a high-frequency, low-amplitude "dithering" noise in the image. Still, especially in the early frames, artifacts may be present in the resulting image – although seldom to an extent where one would judge the unfiltered image to be "better".

Table 3.2 shows some performance measurements for the scenes in Figure 3.4. For the Cornell box scene, the range buffer generation is relatively expensive, because the specular surfaces require a lot of secondary rays to be traced.

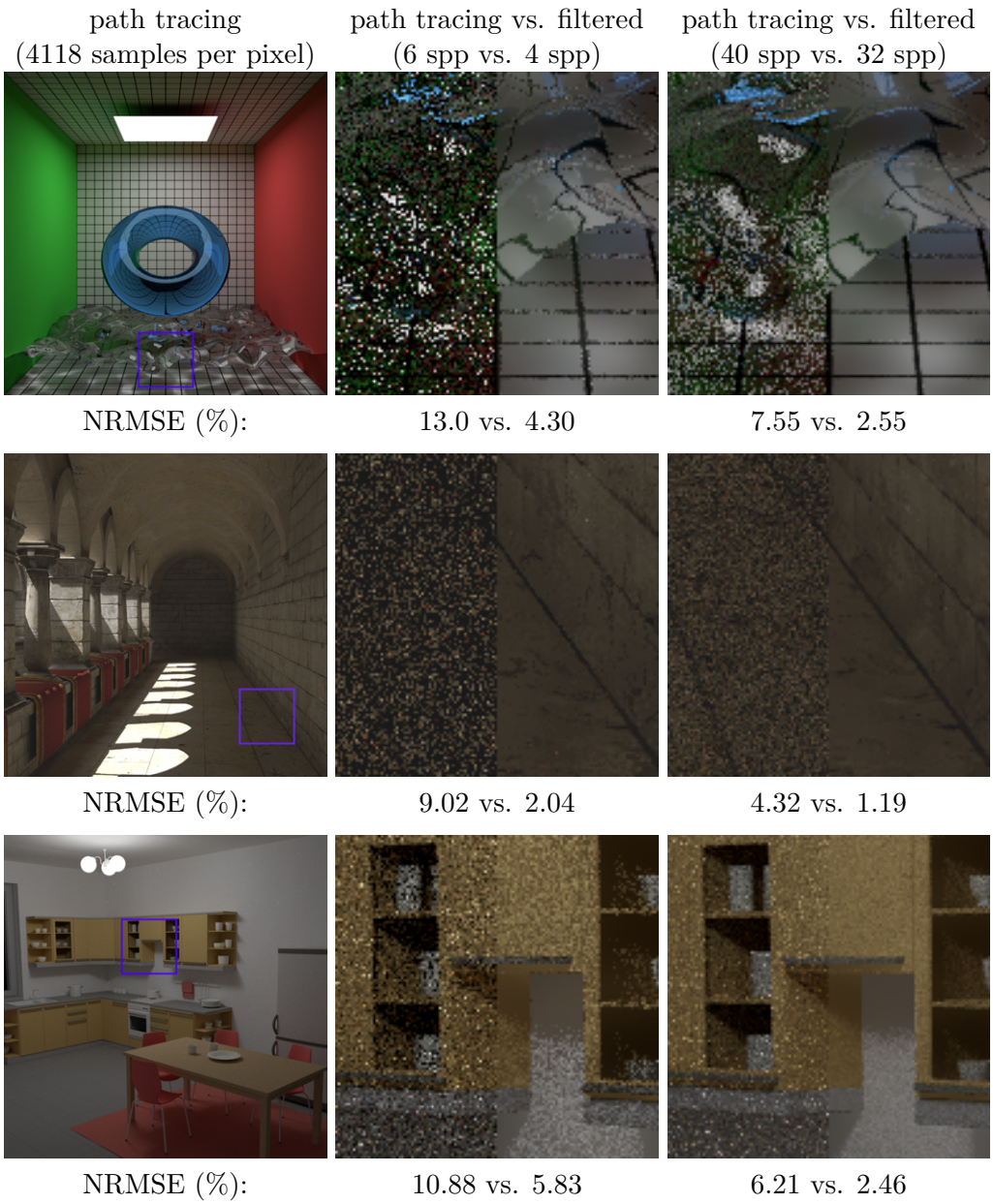

Figure 3.4: *Pixel filtering applied to three scenes, $t = 0.001$. Rendering times in side-by-side comparisons are similar. The number below each comparison is the normalized RMSE (in percent) of the whole image compared to the reference solution to the left. The images in the second row had their brightness increased for print.*

scene	path tracing (ms)	range buffer (ms)	filtering (ms)	relative overhead 32 frames
cornell	499	547	349	18%
sponza	731	95	333	7%
kitchen	688	69	337	7%

Table 3.2: *Performance measurements for the three scenes in Figure 3.4. Timings are given for one filtered frame (except right column) with* 1024^2 *pixels.*

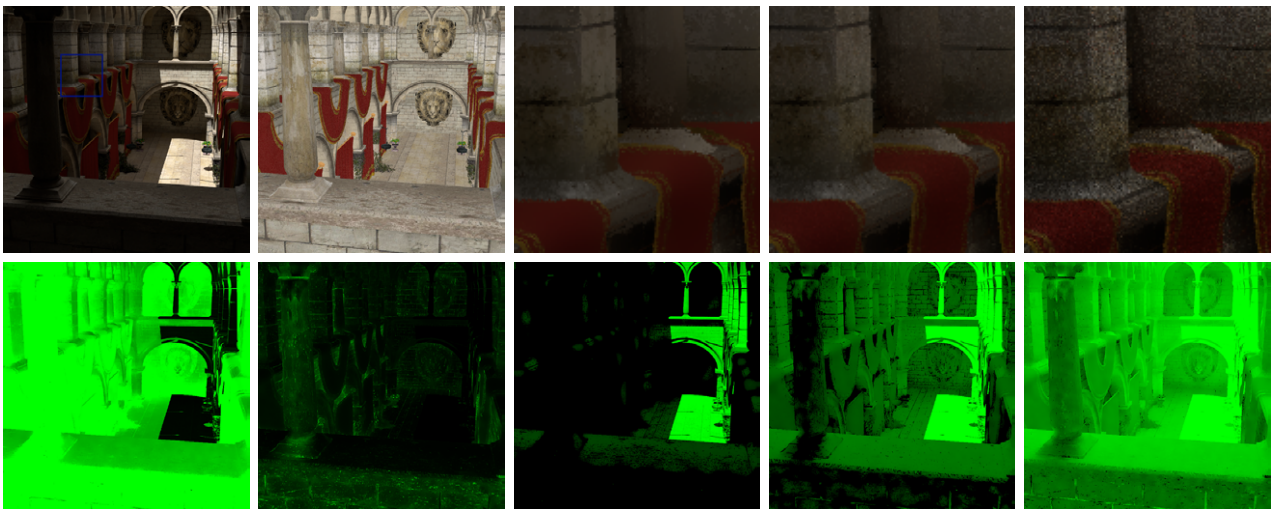

Figure 3.5: *Effects of adjusting the threshold. Top row, left to right: reference, range buffer, blended result with* $t = 0.002$*,* $t = 0.02$*, and* $t = 0.2$*. Bottom row: variance of unbiased high-variance buffer, variance of filtered high-variance buffer, blend factors for* $t = 0.002$*,* $t = 0.02$*, and* $t = 0.2$*. All images used 16 spp, only the reference used 4118.*

For the other scenes, filtering dominates. As a rule of thumb, one application of the filter (including range buffer generation) costs 1-2 path tracing frames. However, since pixel filtering does not filter every frame, the relative overhead for a fixed number of frames (Tab. 3.2, right column) is the more interesting number. Considering the results (Fig. 3.4, right column) this overhead should be acceptable.

Figure 3.5 illustrates the effects of the threshold on the blend factor and how it can be used to mask filtering artifacts at the beginning of the rendering process (16 spp). A threshold of $t = 0.002$ leaves little noise in the image; and although the enlarged part does not contain any striking artifacts, it has a filtered, slightly unnatural look. A threshold of $t = 0.02$ attenuates the

noise significantly (compared to unfiltered path tracing), but leaves enough noise in the image to hide the unnatural look of the filtered contribution. A threshold of $t = 0.2$ does not reduce noise to an acceptable level. With pixel filtering the user can balance noise versus bias at any time, even while the image is rendering. For quick, reliable previews this behavior may be preferable to a noise-free but "overfiltered" image.

Figure 3.6 shows the effects σ_s and σ_c have on the final blended result of pixel filtering. The depth-kernel has been deactivated for this experiment, so that the edge between the pillar and the wall in the background is hardly discernible in the range buffer. (Admittedly, this example is a bit contrived, but the intention was to clearly show the effects.) $\sigma_c = 0.001$ is too restrictive and the filter will not be able to gather enough similar pixels to smooth out the noise. $\sigma_c = 0.1$, on the other hand, leads to a range kernel that is too wide and blurs across the edge. (It also blurs the texture of the pillar.) $\sigma_c = 0.01$, is a reasonable compromise of noise reduction and blurring. For the spatial kernel $\sigma_s = 4$ or $\sigma_s = 8$ (or a value in between) are a good choices. $\sigma_s = 16$ leaks too much light into the shadowed region.

Figure 3.7 shows pixel filtering applied to a difficult case with environment lighting. The dragon on the left has extremely detailed texture maps (diffuse, specular, and normal), the one on the right is perfectly specular. Because direct light with a relatively high variance dominates the scene, we have classified *all* light paths as high-variance paths (i.e. this whole image is filtered, not just the indirect illumination). Filtering artifacts (blurring on the dragons) are present during the early frames, but vanish quickly. Considering the difficulties involved in filtering this image, pixel filtering performs still satisfactorily. Related work has not yet been demonstrated with such scenes.

Figure 3.8 compares variants of the bilateral filter. The original cross bilateral filter (using only the color range buffer) cannot reliably detect the edge between the pillar and the wall in the background and partially filters across this edge. Adding a kernel that weighs samples based on depth information reduces this problem, but leads to aliasing along the edges. Color bleeding is even worse for the bilateral grid, because it can only use a single scalar

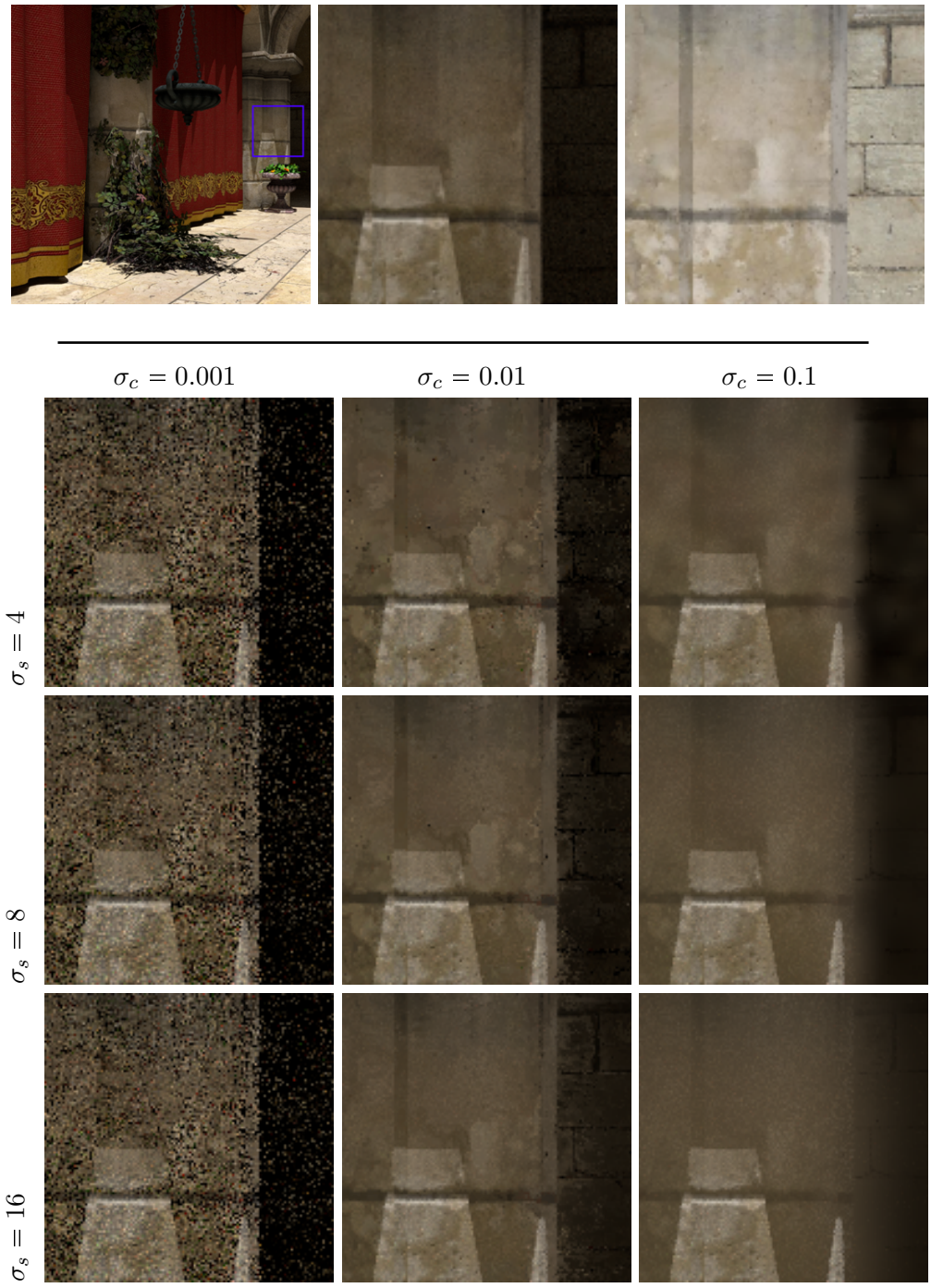

Figure 3.6: *Effects of varying filter parameters. Top: reference, reference enlarged, range buffer enlarged. Bottom: matrix of different values for σ_s and σ_c.*

in its range kernel. We used the luminance, so the filter also bled across isoluminant edges. The artifact is particularly bad for the yellow parts of the curtain bleeding into the background. The separated filter shows streaking artifacts, created by applying the filter first horizontally and then vertically. These streaking artifacts are visually more disturbing than the color bleeding, because the human brain instantly recognizes a vertical streaking pattern all over the image. The À-Trous scheme lies in between the bilateral grid and our method.

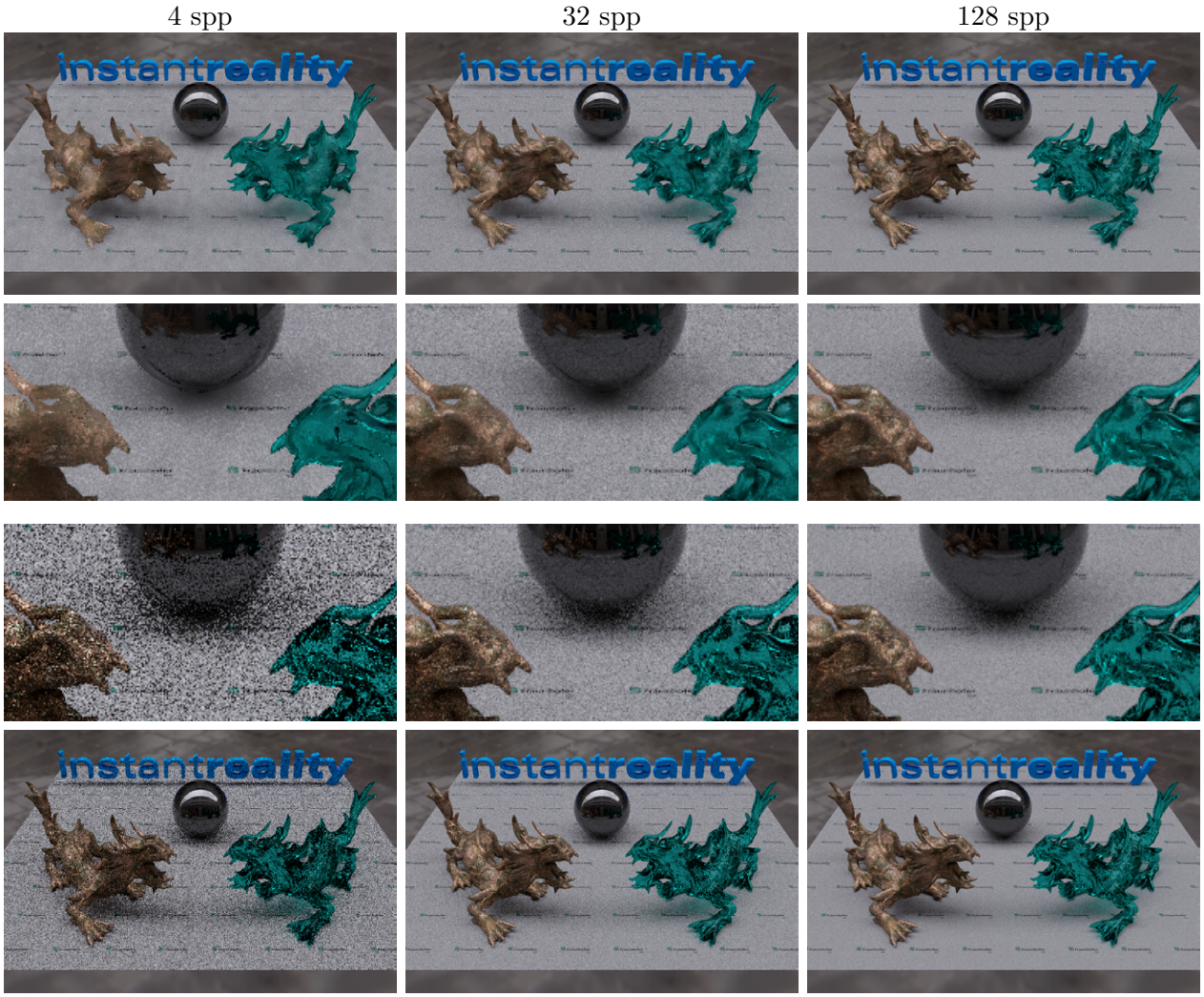

Figure 3.7: *Example with environment lighting and complex materials. Top: pixel filtering (our method, $t = 0.05$, $\sigma_s = 6$, $\sigma_c = 0.007$, all light paths classified as high-variance paths). Bottom: unfiltered path tracing after similar rendering time. Filtering artifacts are visible after 4 spp (blurring on the dragons' heads); after 32 spp they are less objectionable but still visible; after 128 spp they have practically vanished.*

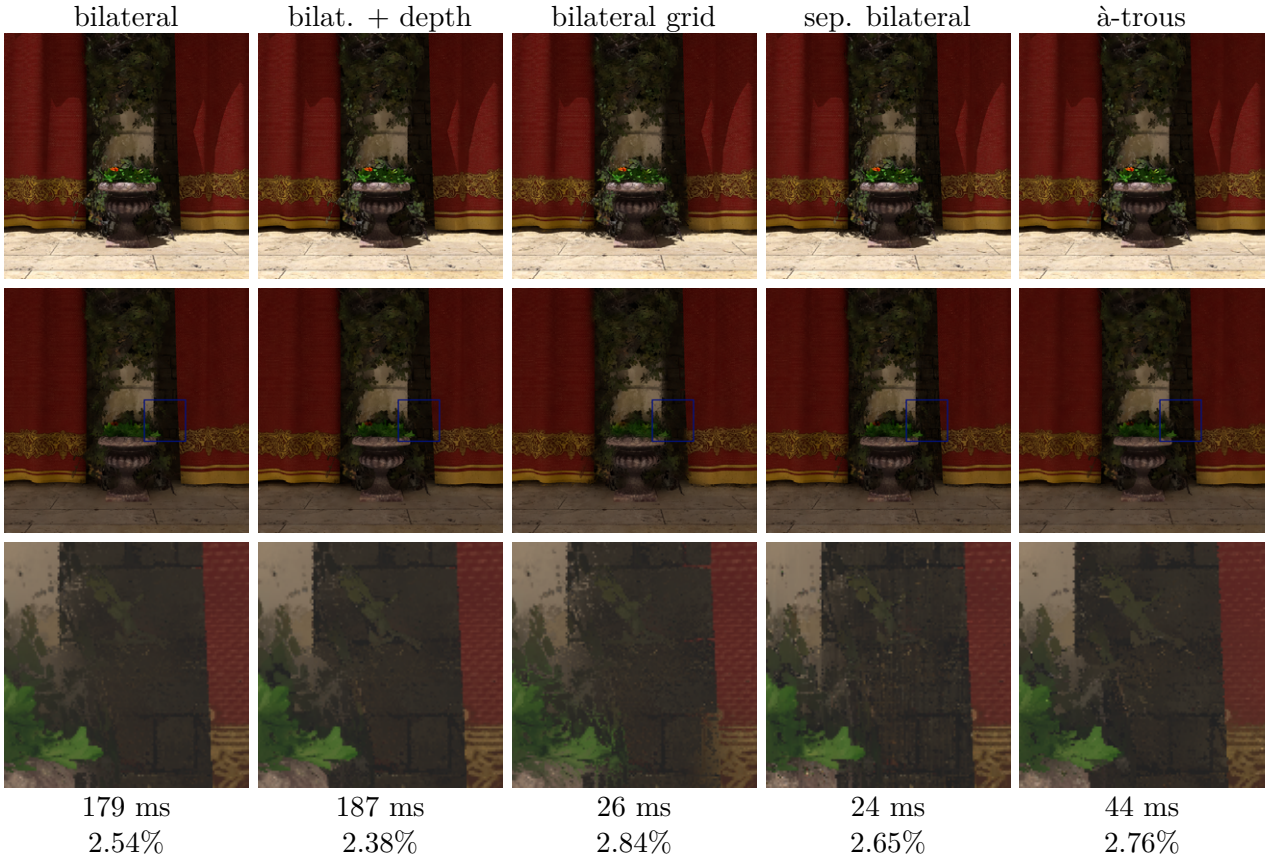

Figure 3.8: *Comparison of different variants of the bilateral filter. From left to right: cross bilateral with color range buffer, cross bilateral with color + depth (our method), bilateral grid, separated cross bilateral, À-Trous filter. First row: direct + filtered indirect; second row: only filtered indirect; third row: filtered indirect enlarged. Parameters:* $\sigma_s = 8$, $\sigma_c = 0.01$, $\sigma_d = 2m$. *Because the bilateral grid only works with luminance values, it used* $\sigma_c = 0.006$ *(which gave comparable filtering performance to the other techniques that used the Euclidean distance in linear sRGB). The filters where applied to the indirect component of a noisy (4 spp) path traced image. The last row shows performance measurements for the filtering pass (excluding range buffer creation) and the normalized RMSE of the filtered version compared to a reference solution (only the indirect component was compared). For the blow-ups in the bottom row the brightness was increased slightly to make the difference better visible in print.*

3.5 Discussion

3.5.1 Comparison with Related Work

Direct comparison with related work is not possible, since our method features a unique blending component. Therefore, we only compare our filtering component with several other approaches. We consider the Gauss-filtered range buffer by Xu and Pattanaik [149], the À-Trous filter by Dammertz et al. [28], and the guided image filter by Bauszat et al. [7] to be the most closely related work. However, it should be noted that the area of application is slightly different for these approaches. Our method aims at filtering for progressive interactive rendering. Xu and Pattanaik provide a filter to clean up an image in a final pass. Dammertz et al. and Bauszat et al. focus on providing a noise-free solution for interactive (non-progressive) rendering.

3.5.1.1 Comparison with Gauss-filtered Range Buffer

Xu and Pattanaik's method is a practical approach to clean up images that have a relatively low noise level, but still some visual outliers. However, it has massive problems dealing with high noise levels in areas that are dominated by indirect illumination, because edges cannot be detected reliably. In that case, the filter blurs high-frequency details in geometry and texture. Lowering the extent of the spatial kernel can reduce blurring, but then the filter smoothes not enough and produces a splotched image. In some of our test cases, this issue rendered the filter effectively unusable until 100-200 samples per pixel were collected (Fig. 3.9). In addition, the asymmetry introduced in the range kernel (comparing Gauss-filtered pixel against original pixels) can lead to energy loss when filtering very noisy images. This results in filtered images that are substantially darker than the reference solution. Our filter can better cope with this scenario, mainly due to the use of a separate range buffer (Fig. 3.4). On the other hand, our filtering step is slightly slower, due to the generation of the very same separate range buffer.

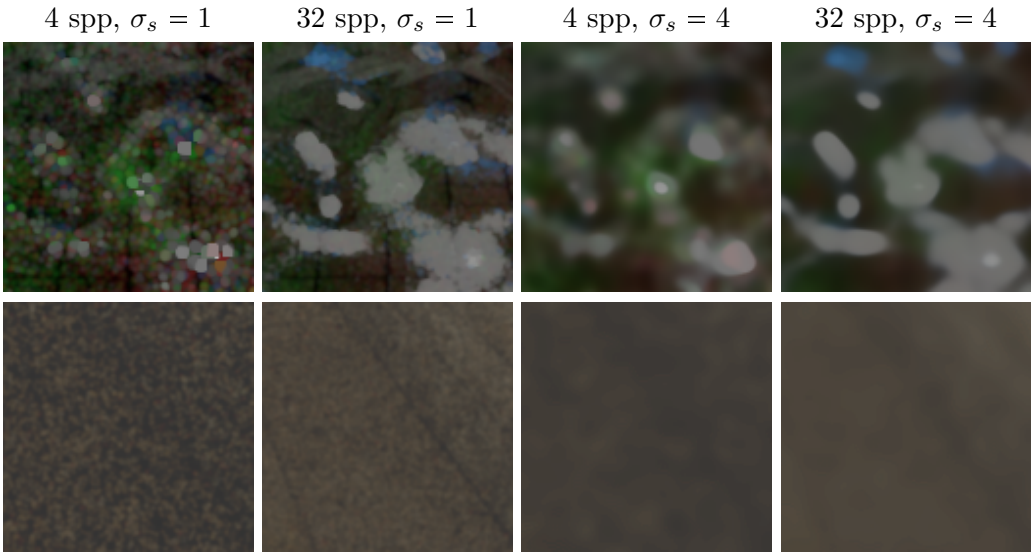

Figure 3.9: *Cases where the Gauss-filtered range buffer technique breaks down (strong noise in areas dominated by indirect illumination). Because the noise level demands large kernels, but the filter offers only weak edge preservation, it is impossible to find an acceptable balance between blurring and noise reduction. The images show the same areas as Figure 3.4, which shows how our filter handles these cases. For the bottom row brightness was increased by 50% to compensate the energy loss.*

3.5.1.2 Comparison with À-Trous Filter

We have slightly modified the À-Trous technique presented by Dammertz et al. [28] for this comparison. First, we have added our range buffer (Sec. 3.3.3) as source for an additional edge-stopping function and dropped the noisy "rt" buffer used in the original paper as well as the position and normal buffers. This was necessary to limit the blurring of high-frequency texture details. Second, we apply only 3 iterations (with a wider kernel) of the filter instead of the 5 (with a narrower kernel) in the original paper. This was necessary to reduce the ringing/stippling artifacts to an acceptable level (a problem also reported by Bausatz et al. [7]).

The filtering quality of the modified À-Trous filter is comparable to our cross bilateral filter (Figs. 3.8 and 3.10), but slightly worse. The artifacts mentioned above are still present in our variant of the À-Trous filter (Fig. 3.10, right), but attenuated to a level where they are hardly visible. Unfortunately, the wider kernel needed to suppress the artifacts eats up some portion of the

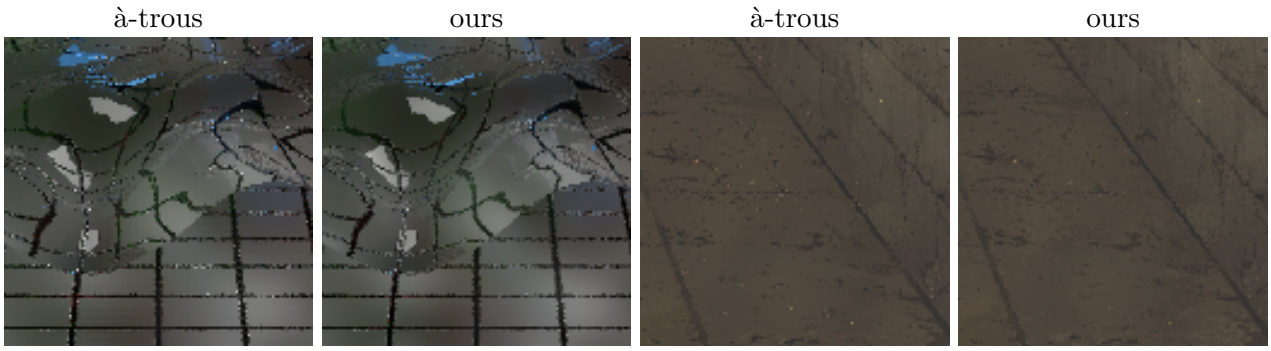

Figure 3.10: *Comparison of our filter with the À Trous filter (input was only the indirect component rendered with 4 spp). The section of the image shown is the same as in Figure 3.4. The last row gives the normalized RMSE (of the whole image) compared to a reference solution rendered with 4096 spp. The two images on the right have their brightness increased by 25% to make the difference visible in print.*

potential gain in speed. Also, we compute the three iterations with three separate kernel-launches and pay the launch-overhead three times. As a result, the À-Trous filter was "only" about four times as fast as our bilateral filter. An optimized version should yield even higher gains. The overhead for generating the edge information is similar in both approaches. For our method as a whole, we prefer the slightly better filtering performance of the classic cross bilateral filter to the faster running time of the À-Trous filter. Mainly because the running time of the filtering step is not a highly critical factor (because of the progression scheme the filter is applied very economically). Therefore, we kept the cross bilateral filter as our main *high-quality* filtering algorithm. However, the À-Trous filter can be used as a *fast* filtering algorithm in our framework, if the user prefers speed over quality (Sec. 3.6.1).

3.5.1.3 Comparison with Guided Image Filter

We have not yet had the opportunity to implement the method of Bauszat et al. [7] and directly compare it with ours. This comparison is based solely on reading the paper and should be taken with a grain of salt. The computation times given in the paper are similar to the À-Trous technique and much faster

than our straightforward implementation of the cross bilateral filter. The cost of generating the edge information should be similar. The paper also includes a clever mechanism to fight the aliasing problems that other approaches based on geometry buffers have and geometric edges seem to be very well preserved. However, how the method handles complex refracting/reflecting objects and high-frequency textures if irradiance filtering cannot be used is not clearly shown in the paper. Overall, we believe our filter to be superior in respecting edges that are present in textures, while their method seems to preserve geometric discontinuities better. The guided image filtering technique could replace the cross bilateral filter in our filtering step.

3.5.1.4 Comparison with Adaptive Kernel Widths

Our blending approach can be interpreted as a procedure to modify the (spatial) filter kernel by assigning a larger weight to the central pixel without modifying the weights of the other pixels relative to each other. In our opinion, this is a better way of balancing noise and bias than variable-width kernels, for the following two reasons. First, the non-linear nature of edge-preserving filters makes it difficult to find an adequate kernel width. The problem is that the weights a_i adapt to the signal (i.e. the pixels), which makes it hard to estimate the variance reduction with Equation 3.5 a priori. Finding a numerical solution in a single frame requires multiple evaluations of the filter and has an unacceptable performance overhead. Interleaving over multiple frames alleviates the performance issue, but then the methods tend to become unstable since the function they are working on is changing between evaluations and not very smooth. The second reason is that with adaptive widths, the only way to reduce bias (blurring) is to reduce the kernel width. Doing this will quickly lead to a splotchy image (Fig. 3.11 right). In contrast, our method blends in the *original* samples that are not blurred at all and that usually contain high-frequency noise. The eye is less sensitive to this noise than to splotches. So, whenever the user prefers a small amount of noise to an image with low-frequency filtering artifacts, the blending approach is a better choice. However, it can make sense to combine adaptive filter widths with our blending operator as an optimization, as outlined in Section 3.6.3.

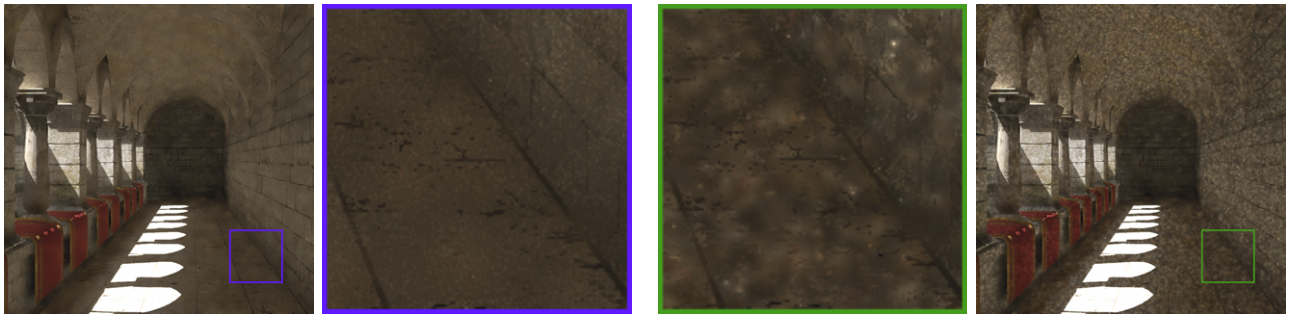

Figure 3.11: *Our blending approach (left) vs. adaptive kernel widths (right). The blending operator used $t = 0.1$ and left some high-frequency noise in the image. For the right image, we tried to reach the same variance reduction by iteratively adapting the kernel width with a bisection method. The filter removed high-frequency noise, but this resulted in a splotchy image, because the widths necessary to reach $t = 0.1$ are (by definition) too narrow to smooth out the splotches. The figure shows the same enlarged portion of the Sponza scene as Fig. 3.4, after 4 spp.*

3.5.2 General Observations

In summary, we have made the following observations.

Filtering performance. In general, the quality of our filtering step is superior to current state-of-the-art approaches in interactive rendering, especially in difficult scenes with high-frequency textures, much indirect illumination, and perfect specular objects that cause reflection, refraction, and caustics. The price we pay for this enhanced quality is a higher running time, but thanks to the progressive filtering scheme this is acceptable. Nonetheless, the filter can produce artifacts in some cases. Extreme intensity spikes can lead to splotches if they are spread out by the filter. On antialiased edges the filter may not find enough similar pixels and may not be able to reduce noise to acceptable levels. (But our method handles antialiased edges better than most approaches that build solely on geometry information for edge detection.) Finally, color bleeding can occur over edges that are not detected. Usually the blending operator can hide these artifacts by leaving a small amount of noise in the image.

Blending operator. The blending operator works well in general. However,

very early in the rendering process the variance estimates may be unreliable. In this case, the operator may not be able to satisfy the user-defined threshold.

Configuring parameters. Pixel filtering requires four parameters to be adjusted: t, variance of perceived noise that should remain in the image; σ_s, standard deviation for the Gaussian kernel in the spatial domain (controls maximum smoothing); and σ_c and σ_d, standard deviations for the range domain kernels (colors and depths – controls edge preservation). The parameters are relatively intuitive and default values of $t = 0.01$, $\sigma_s = 8$ pixels, $\sigma_c = 0.02$, and $\sigma_d = 2m$ usually produce acceptable results. However, in order to achieve optimal results careful manual adjustments may be necessary. Reducing the set of parameters or setting some of them automatically would improve the usability of our method.

Compatibility. In principle, pixel filtering should be compatible with any renderer based on Monte-Carlo path sampling. General optimizations like stratification, importance sampling, and adaptive sampling are orthogonal to pixel filtering. With stratified sampling the algorithm may not correctly estimate the variance reduction in the filtering step (Eq. 3.5), but since this estimation is a relatively rough approximation anyway this is usually not a problem in practice. We have not yet tested our method with advanced sampling techniques that produce correlated pixels and their own characteristic noise pattern (such as Metropolis Light Transport or other Markov chain Monte Carlo methods). It would be interesting to see how the correlation affects our algorithm. The renderer has to provide the range image, the separate high-variance radiance buffer, and the corresponding per-pixel variance values. Apart from that, pixel filtering does not interfere with the rendering process and should be relatively easy to integrate into existing pipelines. However, it is not a pure post-processing.

Overhead. The runtime overhead of pixel filtering consists mainly of the filtering step and range image generation (equivalent to 1-2 spp in our implementation). Due to the progressive filtering scheme, this overhead

quickly approaches zero as the number of samples grows. The blending is essentially for free. These numbers assume that the renderer does already keep track of variances (our renderer does). If that is not the case, this overhead has to be taken into account, too.

In terms of memory requirements, our methods adds 15 floats per pixel (2×3 for the low- and high-variance buffers, 3 for the blurred high-variance buffer, 4 for the range buffer, and 2×1 for the variances). This overhead is comparable to similar approaches and should be acceptable in general.

No bias estimate. Our method tries to limit bias by using bilateral filtering. We have not derived a theoretical bound on the bias or included an estimate of bias into the algorithm. So, in theory, the bias could eat up the benefits gained from variance reduction and the overall rendering error could in fact increase. However, the RMS-error in all of our test cases was significantly reduced by our algorithm, which indicates that this does not happen in practice.

It is also important to note that our algorithm is based on an estimate of variance. The bound on variance could be made more rigorous by using confidence intervals, but since the primary objective of our algorithm is to provide visual previews, we have not deemed this necessary.

Distribution effects. Effects like motion blur and depth of field are problematic, because the algorithm relies on clear per-pixel edges in the range buffer. In theory, it is possible to generate a range buffer that includes these effects, but in order to keep the range buffer free of noise an unacceptable overhead would have to be paid.

3.6 Optimizations

This section briefly discusses some additional optimizations to improve the speed and quality of pixel filtering. There are four notable contributions:

Faster filter. We replace the cross bilateral filter by a fast approximation that allows the filtering step to run more frequently at the price of a small loss in quality.

Adaptive filter width. We introduce a heuristic to adapt the width of the spatial kernel per pixel, which reduces runtime and blurring artifacts.

Spike noise removal. We present a way to withhold spike noise from the final image, which improves the perceived smoothness of the resulting image.

Antialiasing recovery. We extent the filtering step with a procedure that allows us to treat antialiased pixels more correctly.

Our primary objective is to extent the original method, but we would like to point out that many of the ideas and techniques presented in this chapter are applicable to similar approaches.

Note that only the images in this section use these optimizations, the other images are rendered with the original method.

3.6.1 Faster Filter

The original method uses a brute-force implementation of the cross bilateral filter. This produces a filtered buffer of very high quality, but the filter is very expensive, although the progression scheme (filtering only every 2^ith ($i > 1$) frame) can hide the costs well. Nonetheless, in highly dynamic scenes it can be beneficial to have a fast path available and to be able to apply the filter more often early in the image formation process – if the loss in quality is acceptable.

We have investigated three methods: the "separable" bilateral filter by Pham and van Vliet [102], the bilateral grid by Chen et al. [22], and the À-Trous scheme by Dammertz et al. [28].

The bilateral grid is very fast but has some serious weaknesses. First, the range kernel can only operate on luminance values, which can lead to color bleeding across isoluminant edges. Also, it cannot use the depth buffer, which can lead to a huge loss in quality for some scenes. (Gaussian kd-trees [1] could be used, too, but from the figures in the paper they seem to lose a lot of their performance gain with the filter sizes we use, and we have not yet tested them.) Second, it requires additional memory for the three-dimensional grid structure. For the kernel sizes we use, the grid structure takes up approximately as much memory as the frame buffer, which should be acceptable in most cases, but it is a noticeable overhead. Third, it was not obvious to us how the variance of the filtered result can be recovered from the grid (i.e. the sum of the squares of the weights). Fourth, the bilateral grid does not allow varying kernel widths, so the adaptive kernels described in the following section cannot be used.

The À-Trous scheme performs very well (with a few simple modifications). The filtering quality is only slightly worse than that of the original cross bilateral filter and it is quite fast. However, it is not as fast as Pham's "separable" filter, and for our fast path we wanted to make speed the priority.

So, in the end, we have chosen Pham's "separable" filter. This approach simply treats the bilateral filter as if it was separable and splits the filtering process in two one-dimensional (horizontal and vertical) passes. Naively applying the scheme results in objectionable streaking artifacts (which is the reason why we discarded the approach at first). However, Gastal and Oliveira [45] mention a nice trick that can remove these artifacts almost completely: multiple passes with a shrinking spatial kernel are applied iteratively.

They base their scheme on the observation that streaking artifacts are only present along the last filtered dimension. Since the next iteration will start with a one-dimensional pass along the other dimension, it can remove the streaks of the last iteration. The standard deviation $\sigma_s^{(k)}$ for the kernel of the

k-th iteration is given as [45]

$$\sigma_s^{(k)} = \sigma_s \sqrt{3} \frac{2^{(N-k)}}{\sqrt{4^N - 1}}, \tag{3.14}$$

where σ_s is the standard deviation of the desired (combined) kernel and N is the number of iterations. We use three iterations as default.

Figure 3.12 shows a comparison of different filters. The cross bilateral filter used in the original approach provides the best result (in terms of PSNR), but it is also the slowest. The modified separable approximation we use is about nine times faster. Early in the image formation process the separable approximation produces clearly results of lower quality (Fig. 3.12 upper block). However, as more samples are gathered and the input improves, the difference in quality quickly vanishes (Fig. 3.12 lower block). We have also included the À-Trous scheme [28] in the comparison. It lies in between the original filter and the separable filter. However, as already mentioned, speed was the priority for us and we chose the separable filter.

Figure 3.13 shows a comparison between a naive application of the separable filter and the iterative version we use. The naive filter produces vertical streaks; the iterative version effectively removes these artifacts.

3.6.2 Extended Range Buffer

Orthogonal to speed considerations, an improved range buffer with additional edge-stopping functions can improve the filtering quality. This subsection outlines some possible extensions.

A normal buffer can provide some benefit in cases where normal information is not adequately captured by shading with a single head light. Also, if direct illumination is to be filtered, it can make sense to augment the original range buffer with shadowed direct lighting (especially if the lights cast hard shadows). Depending on the number and type of light sources this may increase the shading costs for the range buffer significantly.

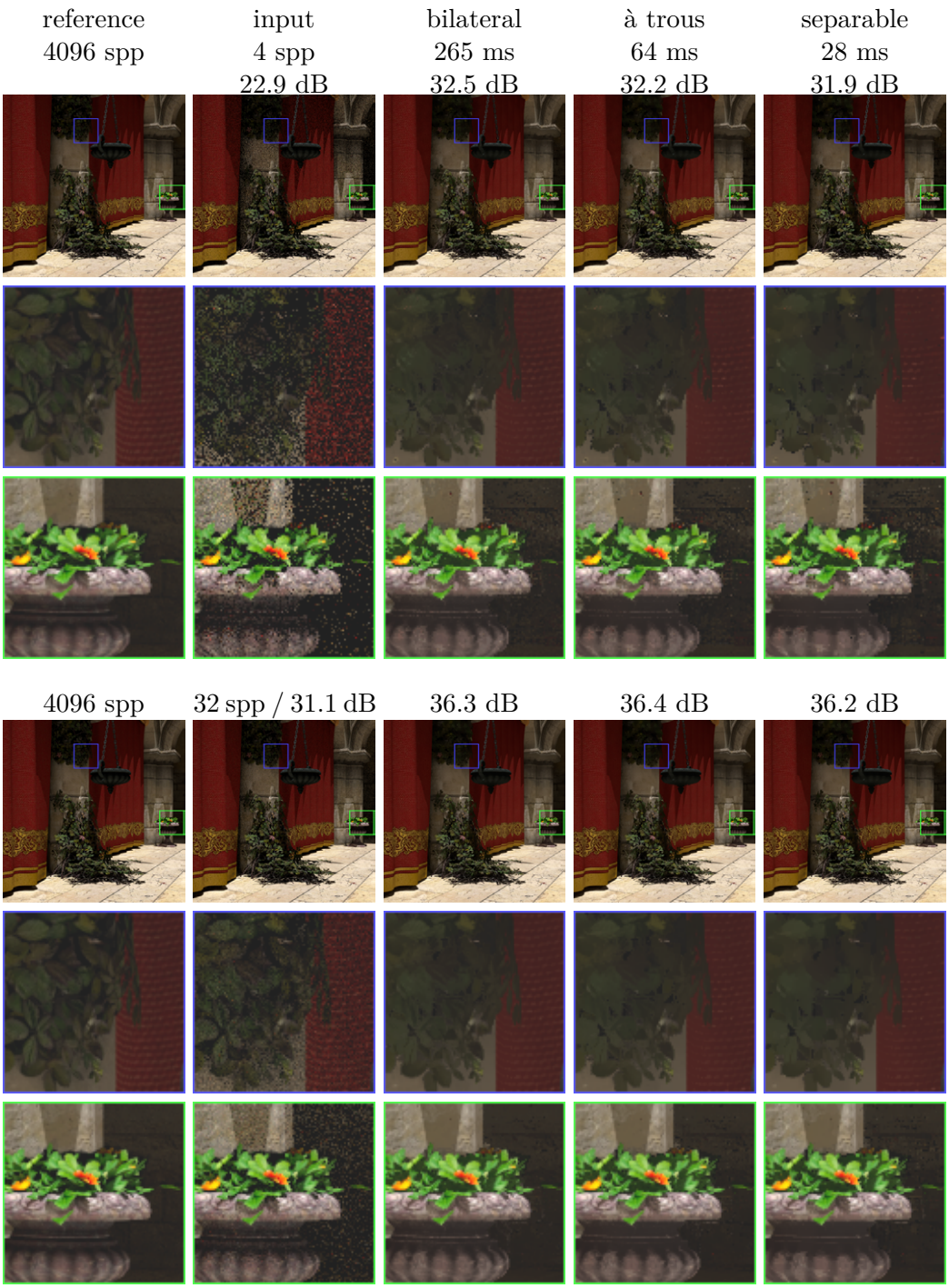

Figure 3.12: *A comparison of filtering steps. "bilateral" is the cross bilateral filter used in the original algorithm, "à trous" is a variant of the À-Trous scheme [28], and "separable" is our variant of the separable bilateral filter [102]. Below the identifiers are the number of samples per pixel (spp), filtering time (in ms), and the PSNR (in dB). The blow-ups had their brightness increased for print.*

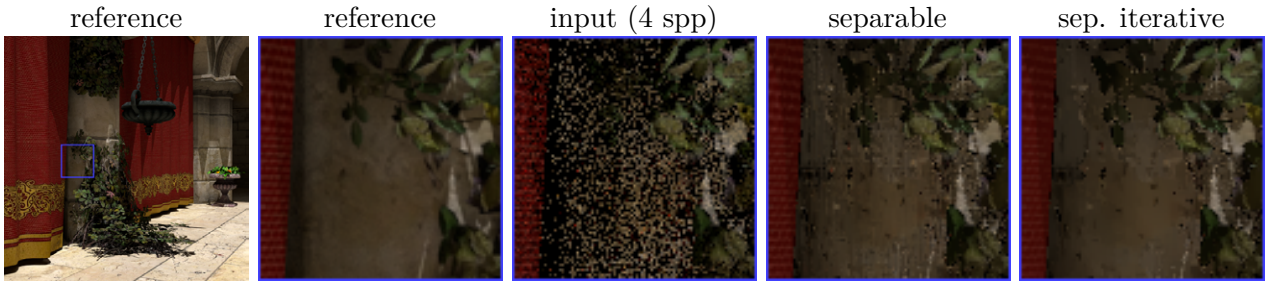

Figure 3.13: *A naive application of the separable bilateral filter produces vertical streaking artifacts. Iterating the filter as described in Section 3.6.1 removes these artifacts.*

Aliasing is a huge problem when storing geometry information. When storing depths, we only store the nearest sample (with respect to the camera), which produces artifacts (jagged edges) in the filtered result. Using a geometry-aware look-up as recently introduced by Bauszat et al. [7] may improve the situation, but we have not tried that yet.

Another modification can improve filter quality in the presence of complex specular objects in front of featureless backgrounds. Then the specular object appears only as a faint "ghost" in the range buffer, and the filter (whose σ_c has been adjusted to the diffuse surfaces) will smooth across large parts of the specular object, resulting in a slightly painted look. This artifact can be reduced by shading specular surfaces as if they also had a diffuse component in the range buffer. This clearly brings out the normal deviations and gives the object a structured surface in the range buffer.

The metric used to measure color differences for the range kernel can also be modified. Currently, we use the Euclidian distance in linear sRGB space per default. Since our renderer works in that space, it is a reasonable compromise between performance and accuracy. However, this color space is perceptually non-uniform and may lead to suboptimal results. Other color spaces or metrics like CIELAB [87] may improve filtering results and the performance loss may be acceptable, but so far we have only implemented the Euclidian distance in sRGB.

3.6.3 Adaptive Filter Widths

The original algorithm uses a constant spatial kernel for the whole image. An adaptive kernel size can improve the filtered result in two ways: First, it can improve performance if large parts of the image have low variance and a small kernel is sufficient to filter out the noise. Second, it can improve the filtered result if the edge preservation fails, that is if the filter blurs across relevant edges. In this case, an adapted smaller spatial kernel will reduce the blurring artifacts that occur.

Unfortunately, the non-linear nature of the bilateral filter poses a problem, as described in Section 3.5.1.4. The variance of the filtered buffer is reduced by a factor of $r = \sum_i a_i^2$ compared to the unfiltered buffer, where the a_i are the filter weights. The problem is that for the bilateral filter the weights a_i are not constant, but depend on the filtered pixels themselves. This makes it hard to estimate a kernel width that reduces the variance to certain amount a priori.

We have experimented with several numerical root-finding algorithms. However, these methods tend to be unstable, because the function for which they are searching the root is not constant, but changes continually (radiances and variances are updated each rendered frame). A simple heuristic that we have found to work quite well is to calculate what extent a simple box filter would have to have to reach a given variance reduction and then translate this extent into a σ_s for the spatial kernel width of the bilateral filter.

The weights of the box filter are simply the inverse of the number of pixels inside the support, so the extent e of a box filter needed to reduce the variance of the filtered buffer to some target f is

$$u\frac{1}{(2e+1)^2} = f \Rightarrow e = \frac{1}{2}\sqrt{\frac{u}{f}} - \frac{1}{2}, \tag{3.15}$$

where u is the variance of the unfiltered buffer. We then simply map e to a σ_s by $\sigma_s = 1.5e$, where 1.5 is a constant determined by experimentation.

A problem with this approach is that the value for u is estimated from a blurred version of the variance buffer. This can lead to a value for σ_s that deviates significantly from the optimal value, especially during the early frames, when the estimates are still unreliable. If σ_s is chosen too small for pixels containing intensity spikes, the energy is not distributed enough and remains in a small neighborhood around the pixel. This can create objectionable splotches in the image. To alleviate this issue, we do not allow a completely free adaption of the kernel width, but restrict it to an interval around a guide line of decreasing values:

$$\sigma_s' = \text{clamp}(c^{-1}\overline{\sigma_s}(n), c\overline{\sigma_s}(n), \sigma_s), \tag{3.16}$$

where c is a constant defining the width of the interval (we use $c = 4$ as default) and $\overline{\sigma_s}(n)$ is a sequence of values decreasing with the number of samples n:

$$\overline{\sigma_s}(n) = n^{-\frac{1}{2}}\overline{\sigma_s}(0), \tag{3.17}$$

where $\overline{\sigma_s}(0)$ is the user-defined base value for the standard deviation and the factor $n^{-\frac{1}{2}}$ stems from the expected convergence of the default Monte Carlo method. It is possible to adjust the exponent in this factor to accommodate for different sampling strategies (e.g. stratification), but this is usually not necessary since the $\overline{\sigma_s}(n)$ only provide a guide line for the σ_s that are used by the algorithm (and adjusted per pixel). This little trick works quite well in practice, but of course a spike noise removal mechanism in the filtering step, similar to the one for the blending step described in the following section, would be a better solution for the future.

Figure 3.14 shows a comparison of the original approach with a fixed kernel size and the heuristically adapted kernel described here. The adaptive kernel preserves the sharp caustic and the glossy reflection on the sphere much better.

Figure 3.15 shows a comparison of our method using a restricted adaptive kernel and a completely free adaptive kernel. An unrestricted kernel produces splotches from pixels with high intensity. The restricted kernel eliminates these artifacts.

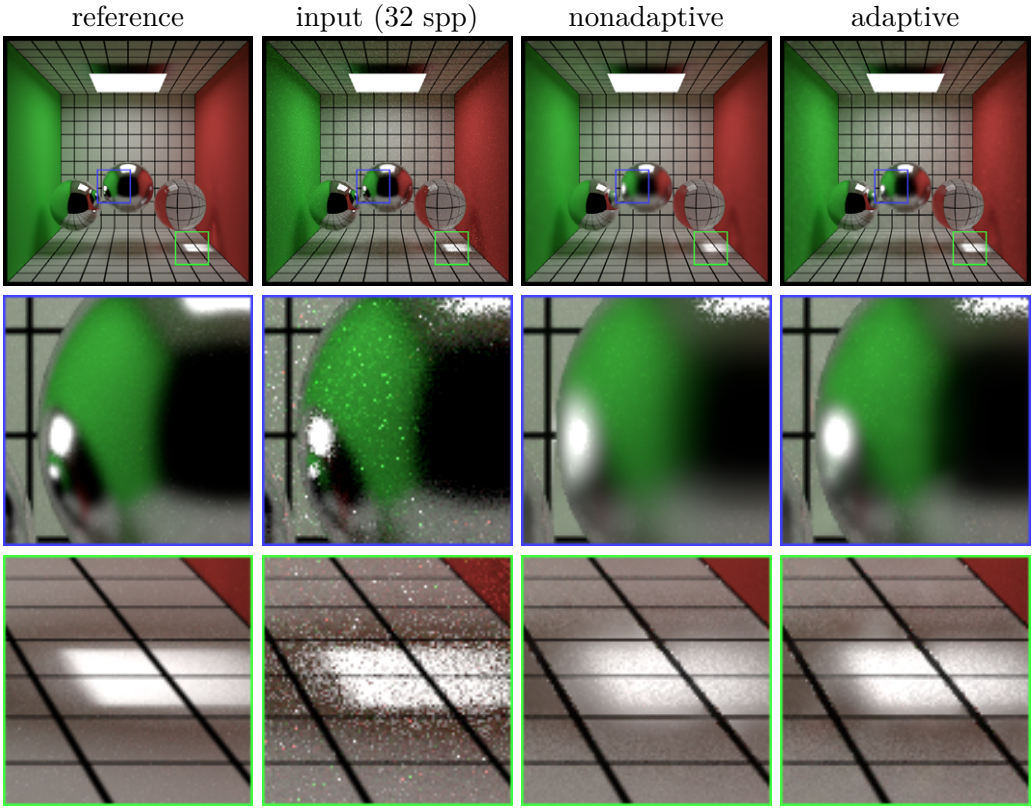

Figure 3.14: *Filtering with adaptive spatial kernel. The original approach with fixed kernel size blurs the glossy reflection and the sharp caustic too much. Using a kernel size that was adapted using the optimization from Section 3.6.3 produces sharper edges and still attenuates the noise below the user-defined threshold.*

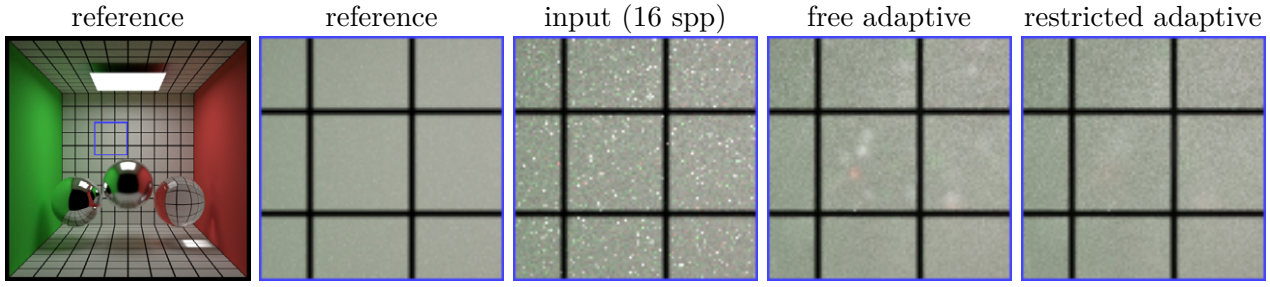

Figure 3.15: *Allowing a completely free adaptive kernel size may result in too narrow kernels. Restricting the kernel size reduces the problem.*

3.6.4 Spike Noise Removal

Spike noise is a huge problem for the original method. It is caused by pixels whose values deviate significantly from their expected value (usually due to bad sampling strategies). The first problem are the splotches mentioned in the previous section that can occur in the filtering step. If the value of the pixel is significantly larger than that of its neighbors or the spatial kernel of the filter is too small, the energy is not distributed in an area large enough and remains concentrated in a single bright splotch. The second problem is that the blending operator blends based on a blurred version of the pixel-variance. So, spike noise samples that are not yet present in the filtered buffer are not blended correctly (because they are also not in the blurred variance buffer). In addition, spike noise pixels that are present may be blended with too high a weight, because the blurred variance may be smaller than the actual variance of the pixel. They will be attenuated to some extent, but the spike will still be visible in the blended result.

We propose a simple mechanism to withhold spike noise from the blended result, this means we address only the second problem – the filtering step is unchanged. There exist sophisticated methods for spike noise removal [29, 96], but since we have a filtered version of the radiance buffer readily available, we can use a simpler and faster approach. Spike noise can be efficiently removed during the blending step by simply clamping the unfiltered pixel (L_u) that may contain spikes to an interval centered on the value of the filtered pixel (L_f):

$$L'_u = \text{clamp}(c^{-1}L_f, cL_f, L_u), \tag{3.18}$$

where c is a constant that defines the borders of the interval. We use $c = 2$ as a default, so the unfiltered contribution is always between half and two times the filtered value.

A potential problem is that very small bright image features that are spread out by the filtering pass may be (wrongly) identified as spike noise and may be removed. This happens if the filtering pass uses a large spatial kernel and fails to detect the relevant edges. However, in combination with the adaptive

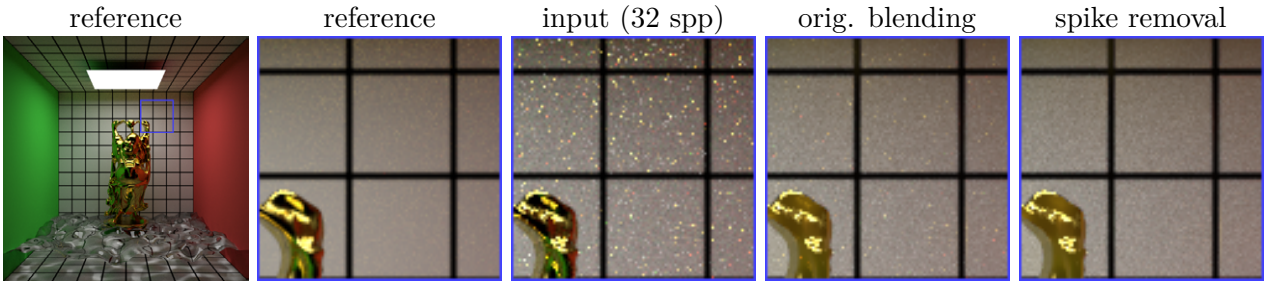

Figure 3.16: *Blending with spike noise removal. The glossy Buddha produces some intensity spikes on the wall. The original blending operator does not effectively withhold these spikes from the image. The optimized operator removes the intensity spikes and produces a cleaner image.*

kernel width we propose in Section 3.6.3, this problem disappears as more samples are averaged and the kernel size shrinks.

Figure 3.16 shows a comparison of the original blending operator with the optimized version that features spike noise removal. The glossy Buddha and the specular surface produce some intensity spikes. The original blending operator transfers these spikes into the final result. They are attenuated by the blending operator as described above, but remain visually objectionable. The optimized version can withhold spike noise from the blended image with practically no overhead. This can improve the perceived quality of the filtered result significantly.

3.6.5 Antialiasing Recovery

Antialiased edges cause problems in the filtering step of the original algorithm. Since the range buffer is supersampled stochastically, the bilateral filter often cannot find enough neighbors with similar values for pixels on antialiased edges. That is because their color is a combination of the regions adjacent to the edge and not present in the (undiscretized) signal itself. The result is that sometimes the filter cannot sufficiently smooth out noise for antialiased pixels. We present a simple edge model for the separable cross bilateral filter to address this issue. It is is based on recent work in antialiasing recovery by Yang et al. [150].

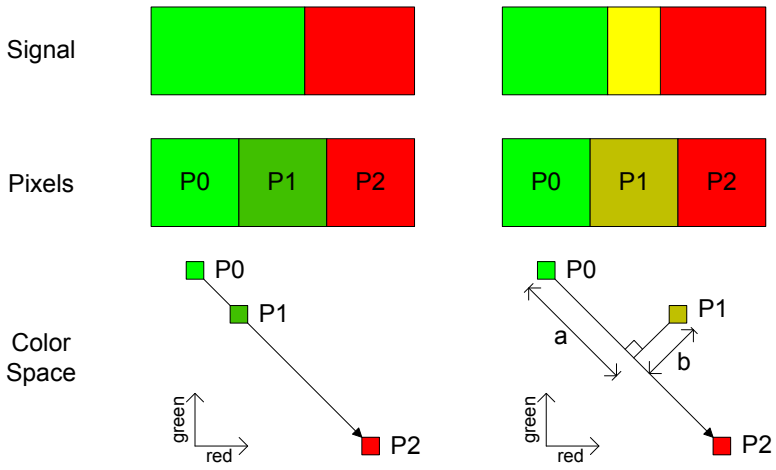

Figure 3.17: *Illustration of our edge model in a 2D red-green color space with two examples. Left: fully antialiased pixel; right: partially antialiased pixel. The pixel in question (here the central pixel) is projected onto the line segment defined by the gradient in color space. (In this illustration, we have only depicted two neighbors, but our implementation actually uses the Sobel operator to calculate the gradient, which takes a* 3×3 *neighborhood into account.) The distances a and b can be used to recover antialiased pixels during the filtering step.*

In each one-dimensional filtering pass, we try to detect and fit a simple edge model to the center pixel. Similar to Yang et al., our edge model assumes a pixel either covers an antialiased edge connecting exactly two regions, or it is not antialiased at all. First, we apply a one-dimensional Sobel filter in the current filtering dimension to get the corresponding component of the gradient. Then we project the color of the central pixel onto the line segment defined by the gradient in RGB color space (Fig. 3.17). This yields two distances: a and b. The distance b tells us how close the central pixel is to the line segment. A small distance means the central pixel is close to being a linear combination of its neighbors and thus we can assume with high confidence it is an antialiased pixel (it is a good match to our simplified edge model). The distance a tells us how much each neighbor contributes (i.e. the coverage).

To preserve antialiasing during the filtering step, we filter with three range values: the original antialiased range value and the range values of the neighbors (Fig. 3.18). This can be thought of as three filter windows in the color range domain. The two contributions corresponding to the neighbors (L_0, L_2) are blended based on the coverage (a) to get the contribution of the edge

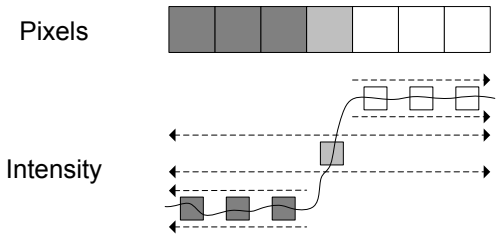

Figure 3.18: *Illustration of the filtering step with antialiasing recovery. Instead of filtering only with the range value for the antialiased pixel (which will not collect any samples but the central pixel itself), we also use two range values defined by the neighbors (which will collect samples). The three results are then blended based on the recovered antialiasing information (Fig. 3.17). The dashed lines illustrate the borders and search directions of the range filter windows: the window for the left (right) neighbor extents only to the left (right) and collects only samples similar the the left (right) range value, the window for the central pixel extents in both directions and collects samples similar to the central range value.*

model (L_e):

$$L_e = (1 - \hat{a})L_0 + \hat{a}L_2, \tag{3.19}$$

where $\hat{a} = a/||P_2 - P_0||$ is the normalized coverage. Then L_e is blended with the contribution of the antialiased range value (L_a) based on the "goodness" of the fit (b):

$$L = wL_e + (1 - w)L_a, \quad w = g_{\sigma_c}(b), \tag{3.20}$$

where $g_{\sigma_c}(b) = \exp\left(-\frac{1}{2}\frac{b^2}{\sigma_c^2}\right)$ is an unnormalized Gaussian with zero mean and standard deviation σ_c. It turns out σ_c, the standard deviation of the color range kernel, is a good value to weigh the "goodness" of the fit. Equation 3.20 states that if the edge model was a good fit, the result will be a coverage-weighted combination of the regions adjacent to the edge; if the model cannot be fitted well, the original filtered value is used.

A problem of our simple edge model in combination with stochastic supersampling and the separable filter is that thin lines that run along the first filter dimension may be thinned out. This is because the noise due to supersampling along these lines looks like many small edges to the algorithm (Fig. 3.19 bottom). However, as the noise in the range buffer disappears, the antialiasing recovery step can detect antialiased edges more reliably.

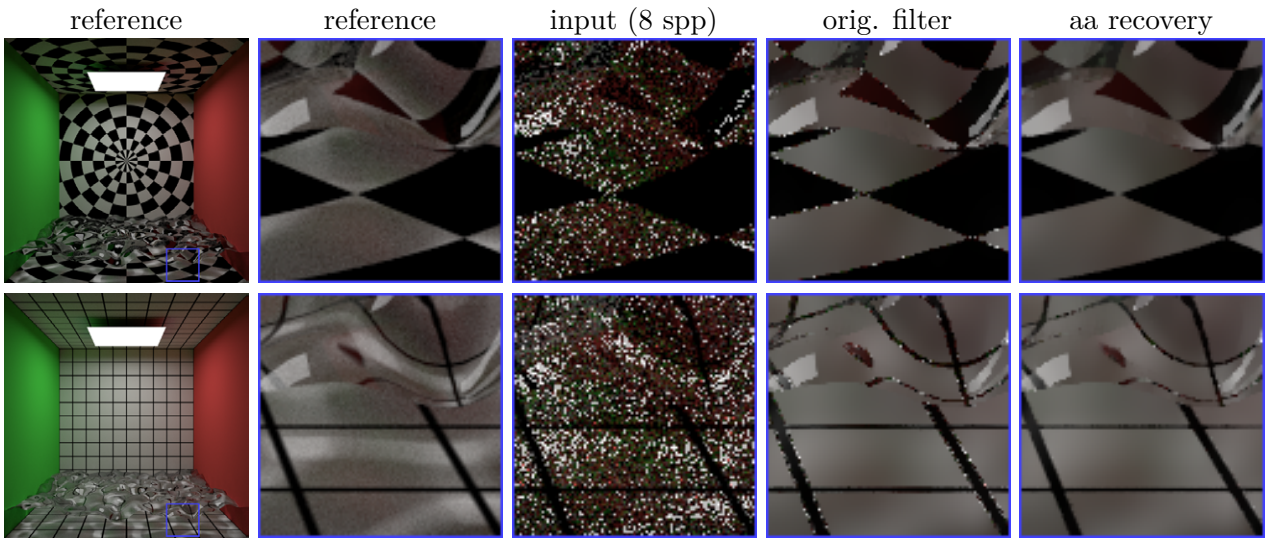

Figure 3.19: *Antialiasing recovery during the filtering step. The original algorithm has problems smoothing out noise on antialiased edges. Top: the extension for antialiasing recovery can smooth antialiased pixels and remove the intensity spikes on the edges caused by the specular surface. Bottom: under certain circumstances our simple edge model may let thin lines appear fainter than they actually are (details in text).*

Figure 3.19 compares the original algorithm with a version that uses our extension for antialiasing recovery. The specular surfaces cause some intensity spikes on the edges of the circular checkerboard pattern. The original filter cannot smooth out all of these spikes, because it cannot find enough neighbors with similar range buffer entries. The antialiasing recovery step reliably detects and recovers antialiased edges. The bottom row of Figure 3.19 illustrates the problem with thin lines that run along the first filter dimension (here horizontally). Horizontal lines appear slightly thinner in the filtered version than they are in the reference image.

3.7 Conclusions

We have described a combined filtering and blending approach to reduce noise in stochastic ray tracing. The method is especially tailored to progressive rendering and achieves strong noise reduction right from the beginning of the rendering process. Filtering performance reflects the target application

(progressive rendering of high-quality images). Our filter is slower than most related approaches for interactive rendering, but the quality of the filtered results is better, especially for more complex scenes featuring high-frequency textures on non-diffuse surfaces and reflective/refractive objects. The biggest innovation of our approach, however, is the blending operator, that allows a user to interactively balance noise versus bias as the image is rendered. Furthermore, it allows the method to use a progressive filtering scheme, which hides the comparatively high filtering costs. We have also described several optimizations that improve the performance of the original method in specialized cases.

The two most pressing issues that remain for future work are the process of finding suitable parameters and reducing filtering artifacts. The method presented in the following chapter addresses these shortcomings.

Appendix 3.A Theoretical Analysis of Blend Factor

This appendix analyzes the blend factor s (Eq. 3.13) in more detail.

First, let us state the argument for consistency given in Section 3.3.5 more formally. If the unfiltered image converges, we have $\operatorname{plim}_{n\to\infty} \hat{u} = 0$ for each pixel, with a consistent estimator $\hat{u}$ for u. This means there exists with probability one an n_0 after which $\hat{u} \leqslant t$. As is evident from the definition of s (Eq. 3.13), this means $s = 1$. In other words, only the unfiltered buffer is used after n_0 samples, which means the bias has vanished.

Furthermore, it is interesting to look at how s behaves for $u \geqslant t$, $t > 0$. It should be noted that the following is an idealized discussion of s as a function of u and f, ignoring the fact that in practice only estimates of u and f are available. Figure 3.20 shows s plotted against u for $t = 1$ and $f = 0.1u$. As u approaches t, s approaches 1. The curve has two interesting points: at point (a) s' is clamped to 1 ($u = t$, $s = 1$), at point (b) s' starts

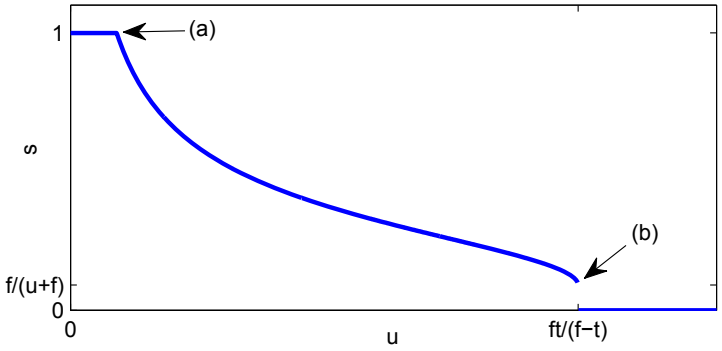

Figure 3.20: *Blend factor ($s(u)$) plotted against variance in unfiltered buffer (u). The plot depicts a rather low variance reduction in the filtered buffer ($f = 0.1u$, with threshold $t = 1$) to clearly show the discontinuity at (b). In practice, pixel filtering usually achieves higher variance reduction ratios and (b) moves closer to $s = 0$.*

to become a complex number with an imaginary part > 0 ($u = ft/(f - t)$, $s = f/(u + f)$).

We will show that $s(u)$ is continuous at (a), not continuous at (b), and strictly monotonic decreasing in the interval between (a) and (b). This means as the variance in the unfiltered buffer decreases, the blended result will contain less and less of the filtered buffer until case 2 is reached and only the unfiltered buffer is used.

At point (a), we have $u = t$. Cases 2 and 3 by themselves are continuous there, so s is continuous there if $s' = 1$, which is true:

$$\begin{aligned} s' &= \frac{f + \sqrt{tu + tf - uf}}{u + f} \\ &= \frac{f + \sqrt{uu + uf - uf}}{u + f} \\ &= \frac{f + u}{u + f} = 1. \end{aligned} \tag{3.21}$$

At point (b) s is not continuous. We have $tu + tf - uf = 0$ and case 3

evaluates to

$$\begin{aligned} s' &= \frac{f + \sqrt{tu + tf - uf}}{u + f} \\ &= \frac{f}{u + f}, \end{aligned} \tag{3.22}$$

which does not match case 1. At first, one would expect this point to coincide with $s = 0$, but it occurs at $s > 0$. This is because Equation 3.12 ignores covariance. This leads the blending operator to believe it can produce a better result by mixing L_f with a small amount of L_u than by taking only L_f. Therefore, it starts with an s slightly greater than 0 – because it is not aware that L_f has already a small amount of L_u mixed in. The theoretical worst case occurs if $f = u$, that is if the filter does not reduce variance at all. Then $s' = 1/2$, but (at least with reasonable filters) this means that the buffer was not filtered at all ($L_f = L_u$) and we would blend between the same values. In practice, the filter will usually reduce variance significantly, and this value will drop towards zero (for a 3×3 box filter it is already < 0.1).

To show that $s(u)$ is strictly monotonic decreasing between points (a) and (b) we show that the derivative of s' with respect to u exists and is negative in this interval. Assuming a linear variance reduction by the filter of $f = ru$, with $r \leqslant 1$, the derivative is

$$\frac{\mathrm{d}}{\mathrm{d}u} s' = -\frac{t}{2u\sqrt{u(rt - ru + t)}}, \tag{3.23}$$

which is well-defined for $t < u < ft/(f - t)$ (the interval between (a) and (b), excluding the points) and negative.

With adaptive kernels, we can assume the filter keeps f constant and independent of u, i.e.

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}u} s' &= \frac{\partial s'}{\partial u} + \frac{\partial s'}{\partial f}\frac{\mathrm{d}f}{\mathrm{d}u} = \frac{\partial s'}{\partial u} \\ &= \frac{-2f\sqrt{tu + tf - uf} + uf - f^2 - tu - tf}{2(u + f)^2\sqrt{tu + tf - uf}}, \end{aligned} \tag{3.24}$$

which is again well-defined for $t < u < ft/(f - t)$. The denominator is

positive, so the nominator dictates the sign; and the nominator is negative:

$$\begin{aligned} & -2f\sqrt{tu+tf-uf}-tu-tf+uf-f^2 \\ =& -2f\sqrt{tu+tf-uf}-(tu+tf-uf)-f^2 \\ <& -f^2 \leqslant 0, \end{aligned} \tag{3.25}$$

where we have used that $(tu+tf-uf) > 0$ (so the first two terms are negative and removing them will make the result larger).

4 Filtering Incident Radiance

This chapter presents a noise reduction method based on filtering incident radiance. Instead of filtering pixel values or irradiance, the incident radiance of neighboring pixels is reused in a filtering step with shrinking kernels. The filter's bandwidth is adapted to reach a user-defined target variance. This approach significantly reduces the variance in radiance estimates without blurring details in geometry or texture.

4.1 Introduction

In this chapter, we present another noise reduction method for interactive progressive path tracing, which we call "radiance filtering". As in Chapter 3, the idea is to exploit spatial coherence in the image and reuse information of neighboring pixels by filtering. However, in contrast to the method presented in Chapter 3 (and most other methods currently used in interactive rendering), radiance filtering does not simply filter pixel values or irradiance. Instead, it filters the incident (indirect) radiance. This approach significantly reduces the variance in the (indirect) illumination without blurring details in geometry or texture and typically leads to higher quality solutions than image filtering. In comparison to more sophisticated sample-based approaches such as light field reconstruction [83] and random parameter filtering [127], radiance filtering is faster and better suited for interactive progressive rendering, where keeping the system responsive is key.

The primary objective of radiance filtering is to provide fast, reliable previews of global illumination in dynamic scenes. The method is easy to integrate

into existing renderers and retains the conceptual simplicity of path tracing, which may be appealing to existing GPU-based implementations. It is also compatible with many common optimizations such as importance, adaptive, and stratified sampling. Furthermore, it is consistent in that any bias introduced during filtering vanishes as the number of samples approaches infinity.

The main contribution of this chapter is this novel approach for noise reduction in interactive progressive path tracing. We also provide a theoretical analysis with convergence rates for bias and variance as well as a practical evaluation. The primary applications we have in mind for our work are interactive design reviews and interactive previews in digital content creation.

4.2 Related Work

In this section, we will focus on approaches that aim to reduce noise in stochastic ray tracing by caching or filtering samples. The balancing act all these methods try to accomplish is to remove (or at least reduce) Monte Carlo noise while preserving genuine scene features. For other techniques geared towards interactive global illumination, we refer to the recent survey by Ritschel et al. [110]. Specifically for many-light methods, the survey by Dachsbacher et al. [27] is a good reference.

4.2.1 Adaptive Sampling and Reconstruction

The approaches for filtering with adaptive kernels by Rushmeier and Ward [117] as well as by Suykens and Willems [132] were already discussed in Chapter 3.

Kontkanen et al. [75] introduced irradiance filtering. Irradiance is integrated at a few sample locations that are determined by a coarse ray tracing pass. Then these noisy irradiance estimates are filtered by a filter that adapts lo-

cally to the irradiance signal. The method is similar in spirit (and name) to ours. One key difference is that our method works for non-diffuse surfaces. In that sense, radiance filtering offers the same improvement over irradiance filtering as radiance caching did over irradiance caching. Another difference is our focus on interactive progressive rendering. Irradiance filtering as proposed by Kontkanen et al. is not well-suited for interactive feedback in dynamic scenes because the irradiance samples have to be recalculated completely after each change. In that regard Kontkanen's algorithm is quite similar to irradiance caching. Compared to radiance filtering, there are also some differences in the computation of the kernel's bandwidth and weights.

Hachisuka et al. [54] combined anisotropic reconstruction with adaptive sampling in a method that works directly in the high-dimensional sample space. An initial set of probing samples is stored in a kd-tree, the set is then adaptively refined in regions with high variance. Finally, an anisotropic reconstruction of the radiance signal is performed, which exploits the smoothness of the signal in the high-dimensional sample space. The method provides good results in offline rendering, but the requirement to build complex auxiliary data structures and statistics hinder the adaption to interactive progressive rendering. A related factor is the relatively high memory consumption, which limit the technique to low-dimensional sample spaces (ca. 5 dimensions).

Adaptive wavelet rendering by Overbeck et al. [94] renders directly into an image-space wavelet basis and adaptively distributes samples according to the coefficients. The image is then reconstructed using soft thresholding on the coefficients in order to remove the remaining noise. The overhead is relatively low for an approach designed for offline rendering, but still an order of magnitude away from being practical for interactive rendering.

Rousselle et al. [114] recently proposed a method that iteratively adapts the bandwidths of pixel reconstruction filters and the locations of new samples in order to minimize relative MSE (consisting of variance and bias) for a given number of samples. Again, the method was not presented in the context of interactive progressive rendering, but in this case an adaption seems to be

relatively straightforward and would be interesting for a direct comparison. A year later, Rousselle et al. [115] extended the approach with Buades et al.'s [20] edge-aware non-local means filter.

Lehtinen et al. [83] used anisotropic reconstruction to approximate the indirect light field from a set of initial samples and then sample this reconstruction instead of the underlying scene to generate the image. The approach was developed in parallel to radiance filtering and achieves very high quality, but the reconstruction procedure is quite expensive and currently not feasible for interactive progressive rendering. In earlier work Lehtinen et al. [82] used a similar approach to reconstruct motion blur, depth of field, and soft shadows.

Another interesting line of work is the frequency-space analysis of light transport by Durand et al. [36], which was followed by several approaches to offline-rendering of distribution effects (e.g. [40, 39]) and, very recently, an interactive approach to rendering global illumination [88]. Incorporating some of these results into a kernel adaption scheme for radiance filtering is an interesting direction for future work.

4.2.2 Edge-aware Image Filtering

Another line of work that has recently gained popularity in interactive rendering is fast edge-aware filtering.

Notable approaches are McCool [86] (anisotropic diffusion), Dammertz et al. [28] (edge-avoiding À-Trous wavelet transform), and Bauszat et al. [7] (guided image filter). For details on these methods we refer to the discussion in Section 3.2.

Our approach from Chapter 3 [125] uses cross bilateral filtering [101, 41] in a similar fashion to Dammertz et al. The main innovation is to combine the filtering step with a subsequent blending step that tries to keep the variance in the final image below a user-defined threshold and makes the method

consistent in a progressive rendering setup. The method also handles specular surfaces better.

Sen and Darabi [127] introduced an interesting variation of cross bilateral filtering by applying random parameter filtering. In addition to the classic weighting according to distance, color and normal they added terms that reduce the contribution of samples based on their dependence on random parameters. The algorithm produces impressive results with low-dimensional sample spaces, where working with large numbers of samples is practical, but currently the overhead incurred by tracking sample statistics prevents the approach from being used in interactive systems.

4.2.3 Caching Sample Information

Krivanek et al. [78, 79] introduced radiance caching as an extension of irradiance caching [145, 25] to glossy reflections. The incident radiance at specific points is cached as a spherical harmonics representation; these samples are interpolated at each shading point and combined with BRDFs in the same representation. To our knowledge, the algorithm has not been used in interactive progressive rendering, although it seems possible to build the samples progressively and recent variants achieve near-interactive performance by making some limiting assumptions (e.g. Scherzer et al. [118]). Of course, the cache records will usually be generated on demand, and in this sense progressively, but each individual record will be constructed in one step (and not progressively). This leads to a huge initial overhead when rendering (re)starts and a blank cache has to be populated. Another important point with respect to our target scenario is that the method is not consistent. In the end, radiance filtering and radiance caching could be used in combination. For example, it may be possible to use radiance filtering to quickly fill a radiance cache. This is one of the items we have identified for future work (Sec. 4.7).

Walter et al. [144] decouple shading from image generation and lazily reconstruct an image by (re)projecting adaptively generated point samples. The algorithm is primarily designed to exploit temporal coherence for interactive

walkthroughs with slowly moving view points, not for variance reduction. It also breaks down on large, abrupt changes, which result in popping artifacts. The shading cache [133] alleviates some of these issues but the popping artifacts remain. Both methods are not consistent.

4.2.4 Progressive Photon Mapping

Hachisuka et al. [56, 55, 74] presented (stochastic) progressive photon mapping, which computes an image by progressive density estimation. The filtering step in radiance filtering is conceptually similar to the density estimation step in photon mapping, but there are a number of differences between the algorithms and their goals. We provide a detailed comparison of progressive photon mapping and radiance filtering in Section 4.4.2, after the complete description of radiance filtering.

Recently, there has been interest in an unification of classic Monte Carlo path sampling and photon mapping [57, 49, 48]. In a way, radiance filtering can be seen as an instance of such an algorithm, where the fuzzy connection is made between different eye-paths and only at the first non-specular surface the paths hit. It would be interesting to see if radiance filtering can be recast in the framework of Hachisuka et al. [57]. However, this is reserved for future work. Radiance filtering also bears a conceptual similarity to eye path reprojection [61] and similar path re-use techniques [11, 148], but we only reproject the first non-specular vertices of neighboring paths for each pixel.

4.2.5 Summary of Related Work

In summary, none of the reviewed methods provides a satisfactory solution to the problem at hand. That is to provide fast, reliable previews of global illumination in interactive progressive rendering with dynamic scenes, while being consistent. Edge-aware and adaptive filtering for interactive (or even real-time) applications are fast, but all of these methods blur textures and

$L_i(x, \omega_i)$	incident radiance at x from ω_i
$f_r(x, \omega_i, \omega_o)$	BRDF at x
$L_o(x, \omega_o)$	exitant radiance at x in ω_o
$\widehat{L_{ok}}(x, \omega_o)$	exitant radiance estimate of path tracing in frame k
$\widetilde{L_{ok}}(x, \omega_o)$	exitant radiance estimate of radiance filtering in frame k
T_u	user-defined threshold on variance
T_k	target variance in frame k
r_{sk}	image-space radius of filter kernel in frame k
M_k	target number of radiance samples in kernel in frame k
$\tilde{\epsilon}_k$	error with radiance filtering in frame k
$\bar{\tilde{\epsilon}}_N$	accumulated error with radiance filtering after N frames
$\hat{\epsilon}_k$	error with path tracing in frame k
$\bar{\hat{\epsilon}}_N$	accumulated error with path tracing after N frames

Table 4.1: *Quick reference for the most important symbols used in this chapter.*

other features to some extent or suffer from aliasing in their edge-detection. For diffuse surfaces, one can filter irradiance and postmultiply by the diffuse BRDF, but this does not work well for non-diffuse materials. Caching algorithms are not consistent or not well-suited for fast previews of interactive scenes, because they have to pay the initial overhead of reconstructing the cache each time the scene changes. Sophisticated adaptive sampling and reconstruction algorithms provide high-quality results, but are intended for offline rendering and not feasible for interactive applications. Progressive photon mapping is in parts similar to our method, but does not incorporate the focus on previews as much as we do. It is also a complete rendering algorithm on its own, while we regard radiance filtering merely as an extension to path tracing.

4.3 Method

We will explain radiance filtering for (unidirectional) path tracing, which is the only variant we have implemented and tested. However, adapting the idea to similar rendering algorithms based on random path sampling should easily be possible.

Table 4.1 explains the most important symbols we will use. In general, $\tilde{\cdot}$ refers

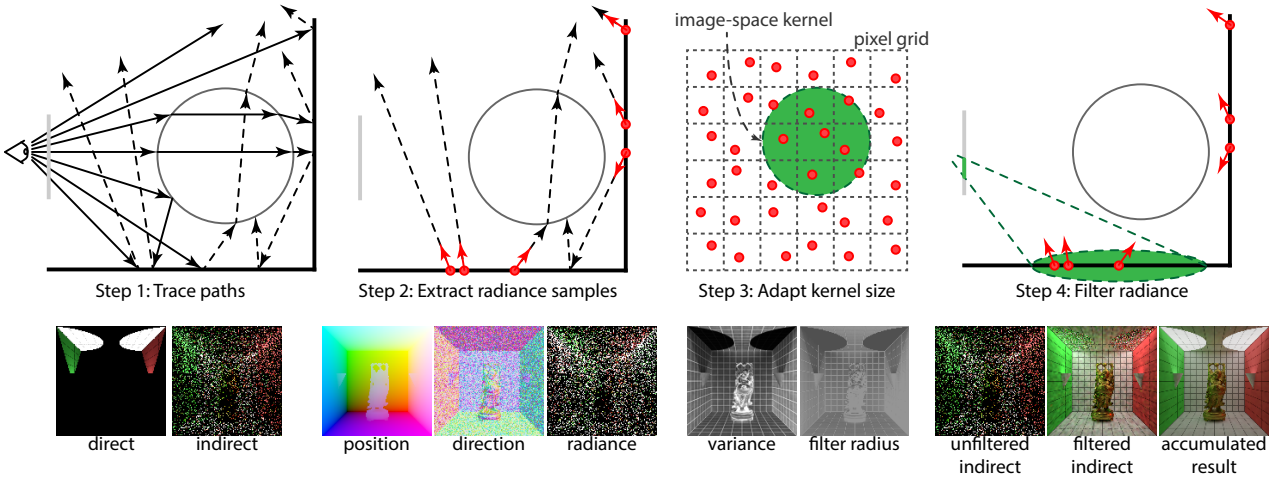

Figure 4.1: *A schematic overview of one frame of radiance filtering with some of the relevant buffers.*

to a filtered quantity, $\hat{\cdot}$ to the corresponding unfiltered path traced estimate, and $\bar{\cdot}$ to an average or accumulated quantity.

The objective of radiance filtering is to provide fast previews of global illumination for interactive scenes and to keep the rendering algorithm consistent at the same time. The method achieves fast previews by filtering the incident radiance with the goal to keep the variance of each pixel just below a user-defined threshold. Consistency is achieved by progressively shrinking the filter kernels. Figure 4.1 shows a schematic overview of the algorithm.

Section 4.3.1 describes a single frame of radiance filtering, Section 4.3.3 shows how a series of these filtering steps can be embedded into a progressive scheme that ensures the goals of fast previews and consistency. Sections 4.3.4, 4.3.5, and 4.3.6 further detail some aspects of the method.

4.3.1 Filtering Step

The basic idea behind the filtering step is to treat radiance samples of neighboring pixels as independent realizations of the same random variable. The variance of this variable (and thus the noise in the image) can be reduced by

taking a weighted average of independent realizations (i.e. by filtering). In general neighboring samples represent different variables, so this assumption is only approximately true and filtering means trading noise for bias.

The key observation is that the unwanted portion of the variance in the outgoing radiance is due to the incident radiance, not the other terms in the local reflectance integral (if proper BRDF importance sampling is performed).

The following paragraphs explain how radiance filtering is implemented on top of path tracing. The discussion is based on the steps outlined in Figure 4.1.

Step 1: Trace paths. We begin with standard path tracing, but we accumulate only direct lighting. The sampled indirect illumination for each path is stored in a radiance sample, which is used later in the filtering step to make the result of the path available to neighboring pixels. Perfect specular surfaces are traversed stochastically, similar to progressive photon mapping. So, the radiance sample for a path contains a sample of the incident indirect radiance at the first non-specular surface the path hits. The exact contributions depend on the implementation of the path tracer, but typically all explicit light connections (next event estimation) at subsequent path vertices are included as well as implicit connections (direct light hits).

Step 2: Extract radiance samples. The radiance samples are extracted and stored in the pixel through which the path started. There is no additional data structure. Typically the algorithm works with one sample per pixel, but it is possible to store multiple samples if more than one path is traced in one pass. A radiance sample consists of position, direction, and radiance. The collection of radiance samples can be thought of as an image-space photon map, generated by path tracing, not by a photon tracing pass. The reasoning behind storing the radiance samples in pixels is that we can quickly find relevant samples by searching image-space neighbors. This breaks down in the presence of geometric edges and complex specular objects, but usually there is some spatial

coherence to exploit.

Step 3: Adapt kernel size. The goal of the filtering step in each frame k is to reach a given target variance T_k. To do this, we need to average M_k samples:

$$T_k = \mathrm{Var}\left[\frac{1}{M_k}\sum_j^{M_k}\widehat{L}_{ij}\right] \overset{(a)}{\approx} \frac{1}{M_k}\mathrm{Var}[\widehat{L}_i]$$
$$\Rightarrow M_k = \frac{\mathrm{Var}[\widehat{L}_i]}{T_k}, \tag{4.1}$$

where the approximation (a) uses the assumption that neighboring pixels are realizations of the same random variable.

Since we have one sample per pixel, we expect approximately M_k samples in the kernel if it covers M_k pixels. This gives us the kernel radius in image-space. Assuming a circular footprint it is

$$r_{sk} = \sqrt{M_k}\frac{1}{2}\frac{4}{\pi} = \sqrt{\frac{\mathrm{Var}\left[\widehat{L}_i\right]}{T_k}}\frac{1}{2}\frac{4}{\pi} \tag{4.2}$$

pixels. For a quadratic footprint, the factor $4/\pi$ has to be omitted.

We also scale the radius by an ambient occlusion factor $\alpha \in [0, 1]$. This has an effect similar to the harmonic mean distance in irradiance caching and effectively and reduces the kernel size in highly occluded regions, which helps reducing light leaks and blurred shadows. The effect is relatively subtle, but since we can get a screen-space approximation of ambient occlusion essentially for free (we loop over the screen-space neighbors anyway), we include the scaling per default.

Step 4: Filter radiance. In order to avoid pulling in samples that are close in image-space, but far in world-space, we project the kernel into world-space and apply the filter there. The projection can be done with ray differentials or with a geometric construction using the surface normal of the shading point, as described in Section 4.3.5. After projecting the kernel, we loop over all pixels that the image-space kernel covers

and accumulate their radiance samples, weighing them according to the weights described in the following subsection. This estimate of indirect illumination is then added to the direct illumination to form the preview image.

4.3.2 The Filtered Radiance Estimate

The exact formula for the filter can be derived in the framework of multiple importance sampling [137, Ch. 9]. We want to combine m samples drawn with (potentially) different pdfs into one estimator while keeping the variance low. The exitant radiance can be estimated from the incident radiance samples as

$$\widetilde{L_o}(x, \omega_o) = \sum_{j=1}^{m} v_j \frac{F_j}{P_j}, \tag{4.3}$$

where

$$F_j = f_r(x, \omega_{i,j}, \omega_o)\breve{L}_i(x_j, \omega_{i,j})(n \cdot \omega_{i,j}) \tag{4.4}$$

is the jth radiance sample reflected at the current shading point x and

$$P_j = p_j(\omega_{i,j}) \tag{4.5}$$

is the pdf for sampling direction $\omega_{i,j}$ at point x_j. ($\breve{L}_i$ is the path tracing estimate $\widehat{L}_i$ without the division by the pdf.)

The balance heuristic defines the weights v_j as

$$v_j = \frac{p_j(\omega_{i,j})}{\sum_{k=1}^{m} p_k(\omega_{i,j})}. \tag{4.6}$$

However, m is usually quite large in the early frames and reevaluating all pdfs for each sample is at odds with our goal of interactivity. Therefore, we assume all p_k are equal to the pdf of the current shading point p and set

$$v_j = \frac{p_j(\omega_{i,j})}{mp(\omega_{i,j})}. \tag{4.7}$$

If we substitute these weights into Equation 4.3 the p_j cancel and we arrive at

$$\widetilde{L_o}(x, \omega_o) = \frac{1}{m} \sum_j \frac{f_r(x, \omega_{i,j}, \omega_o) \breve{L}_i(x_j, \omega_{i,j})(n \cdot \omega_{i,j})}{p(\omega_{i,j})}. \tag{4.8}$$

Note that there are two sources of bias in this estimate. The first is the obvious proximity bias that stems from taking $\breve{L}_i(x_j, \omega_{i,j})$ from neighboring points. The other source of bias is more subtle and stems from the approximation in Equation 4.7. The approximation has the effect of producing a very smooth result, because we always combine the BRDF of point x with the pdf of point x, which helps reducing intensity spikes that could occur if BRDF and pdf would not match each other. Unfortunately, it also introduces bias, because the pdf at x does not represent the true probability density with which the $\breve{L}_i$ were sampled. This "pdf bias" is usually negligible, but for highly glossy, curved surfaces it can be problematic (because the pdf can change quite dramatically for neighboring pixels).

To reduce proximity bias, we weigh the samples with a world-space spatial kernel $\sigma(x_j, x)$, as already described. To limit pdf bias, we use a simple heuristic based on the glossiness of the surface and the normal difference:

$$\nu(n_j, n) = \max(0, (n_j \cdot n))^e, \tag{4.9}$$

where e the is the exponent of the Blinn microfacet distribution. (It should easily be possible to derive similar heuristics for other shading models.) Note that for mildly glossy surfaces ($e \lessapprox 128$) this can be set to one, which can save the extra normal buffer that is otherwise not needed by our algorithm. For extremely glossy surfaces ($e \gtrapprox 4096$) it usually makes more sense to treat them as perfectly specular and continue tracing the rays.

The final estimate is thus

$$\widetilde{L_o}(x,\omega_o) = \frac{1}{W}\sum_j w_j \frac{f_r(x,\omega_{i,j},\omega_o)\breve{L}_i(x_j,\omega_{i,j})(n\cdot\omega_{i,j})}{p(\omega_{i,j})}, \tag{4.10}$$

with weights

$$w_j = \frac{1}{m}\sigma(x_j,x)\nu(n_j,n) \tag{4.11}$$

and normalization factor

$$\frac{1}{W} = \frac{1}{\sum_j w_j}. \tag{4.12}$$

Note that x, n, and ω_o in the estimate are taken from the current shading point, not the neighbors, which reduces blurring in the factors f_r and $(n\cdot\omega_{i,j})$. However, shadows and sharp glossy reflections are still blurred, as they are included in L_i. Usually we only filter indirect illumination, which is the primary source of noise and can be expected to be reasonably smooth, but the algorithm can also be applied to smooth direct illumination.

4.3.3 Progressive Setup

In a progressive rendering setup, we want the variance of the *accumulated* result to stay below a user-defined threshold T_u. At the same time we have to guarantee that both, variance and bias, vanish as the number of samples approaches infinity.

For the first frame ($k = 1$) we can use $T_k = T_u$ as target variance. We can derive the target variance $T_k = \mathrm{Var}\left[\widetilde{L}_k\right]$ for each frame $k > 1$ by induction. Here, $\widetilde{L}_k$ is the filtered radiance estimate of one pixel calculated in frame k. Assume $\mathrm{Var}\left[\widetilde{\overline{L}}_{k-1}\right] \leqslant T_u$, i.e. the accumulated filtered radiance up to frame $k-1$ fulfills the threshold. If we combine frame k with the already

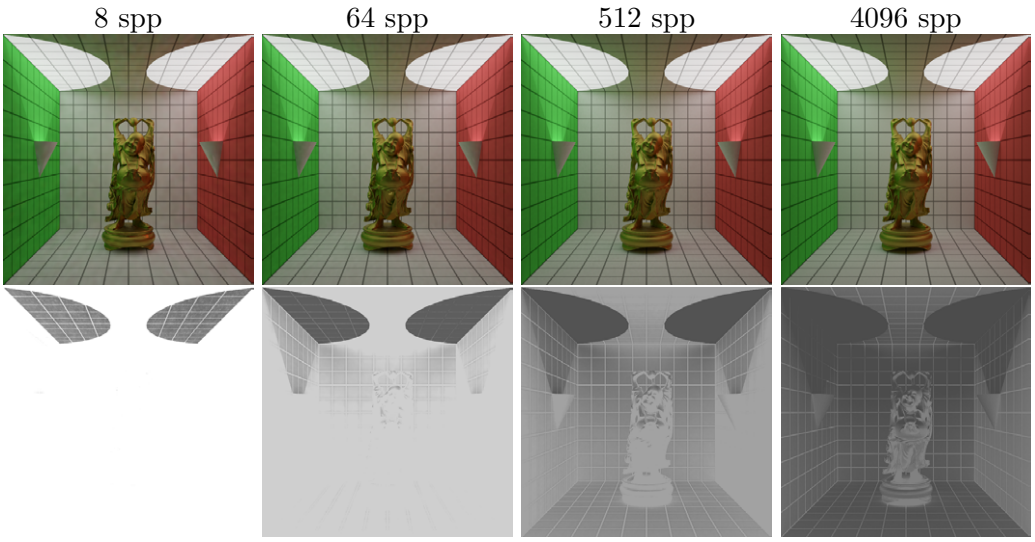

Figure 4.2: *A typical progressive rendering (top row) with corresponding image-space radii (bottom row).*

accumulated samples the variance will be

$$\begin{aligned}\mathrm{Var}\left[\overline{\widetilde{L}}_k\right] &= \mathrm{Var}\left[\frac{k-1}{k}\overline{\widetilde{L}}_{k-1} + \frac{1}{k}\widetilde{L}_k\right] \\ &= \left(\frac{(k-1)}{k}\right)^2 \mathrm{Var}\left[\overline{\widetilde{L}}_{k-1}\right] + \frac{1}{k^2}\mathrm{Var}\left[\widetilde{L}_k\right] \\ &\leqslant \left(\frac{(k-1)}{k}\right)^2 T_u + \frac{1}{k^2}T_k. \end{aligned} \tag{4.13}$$

Solving $\left(\frac{(k-1)}{k}\right)^2 T_u + \frac{1}{k^2}T_k \leqslant T_u$ for T_k gives

$$T_k \leqslant k^2\left(1 - \left(\frac{k-1}{k}\right)^2\right) T_u = (2k-1)T_u, \tag{4.14}$$

which allows us to find the target variance for filtering step k directly from the user-defined threshold. With Equation 4.2, T_k can be translated into an image-space kernel radius. Note that the target variance T_k is allowed to increase linearly with the number of frames, this means the (screen-space) radius of the kernel will decrease with the square root. Figure 4.2 shows this process in practice. Radii are high in regions with high variance and decrease with more samples.

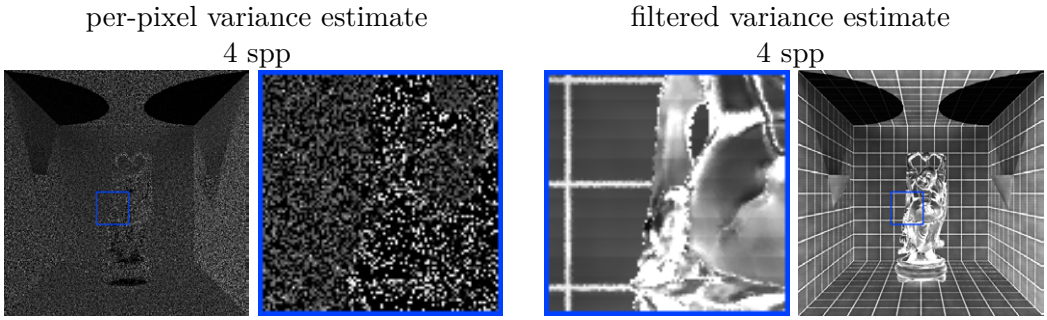

Figure 4.3: *The noisy and the filtered variance estimate. Note that this is the "perceived variance" as described in Section 3.3.1, which is why dark regions have high values.*

When the target variance for frame k is already larger than the variance of the path tracing estimator, the filtering step becomes unnecessary (the radius drops below the pixel size). Then we start accumulating unfiltered samples, which will not introduce additional bias (as they are raw path tracing samples). Since every pixel with finite variance will reach this point eventually, the accumulated bias will vanish as the number of samples continues to grow. This is an intuitive argument for the consistency of radiance filtering, a more thorough analysis in carried out in Section 4.4.

4.3.4 Remarks on the Variance Estimate

It is important to note that in practice the calculations are based on an *estimate* of the variance, since the real variance is not known. Therefore, unreliable estimates, especially in the first few frames, can lead to wrong kernel sizes. To deal with this issue we use a "warm-up" kernel radius for the first frames (usually 4-8) and then switch to the adapted version based on the per-pixel variance described in Section 4.3.1. The warm-up radius for frame k is

$$r_{wk} = r_{w1} k^{-\frac{1}{2}}, \quad k > 1, \tag{4.15}$$

where r_{w1} is a user-defined initial value; the factor $k^{-\frac{1}{2}}$ stems from the expected decrease of the kernel radius (Sec. 4.3.3).

A related issue is that the variance estimates themselves are contaminated by noise, this means the estimates may differ greatly from pixel to pixel, although the variances are in fact very similar. In order to obtain smooth variance estimates in image-space that still respect relevant geometric edges, we filter the variance estimates alongside the radiance values in the filtering step. This has little additional overhead and greatly improves the robustness of the algorithm, especially during the first few frames. Figure 4.3 shows the per-pixel variances and the filtered version.

Furthermore, we actually use the *perceived variance* as defined in Section 3.3.1. As a quick reminder: This is a simple measure of the perceived noise level of an image that incorporates Weber's law [16, 107]. It acknowledges the fact that the same absolute error triggers a larger response in the visual system when viewed against a low background radiance than it does when viewed against a high background radiance. For each pixel, this background radiance is fixed and the perceived variance is just a scaled version of the absolute variance relative to this fixed background intensity, so all equations remain valid. We currently use the sum of the direct and the filtered indirect radiance as background radiance, which is a smooth function that we can compute in the filtering step without much additional overhead.

It should also be noted that the filtering step can only remove variance *in the illumination*. Noise due to stochastic supersampling or other distribution effects in the image plane like depth of field is not reduced (see Section 4.5). However, many renderers keep track of the total pixel variance, including these effects as well as direct illumination. Such a variance estimate can still be used with radiance filtering. Doing this usually results in a slight overestimation of the variance, and consequently to slightly larger kernels, but in most cases this is not a problem in practice.

4.3.5 Projection of Kernel

After step 3 of the algorithm ("adapt kernel size" in Section 4.3.1), we have the footprint of the kernel as a rectangle in image-space (Fig. 4.4). In practice, we apply a scaling factor prior to the projection that makes the kernel

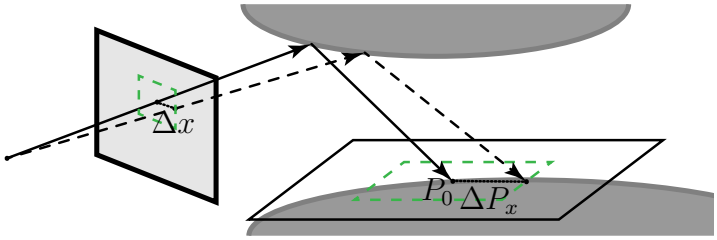

Figure 4.4: *The image-space kernel is projected into the tangent space of the shading point.*

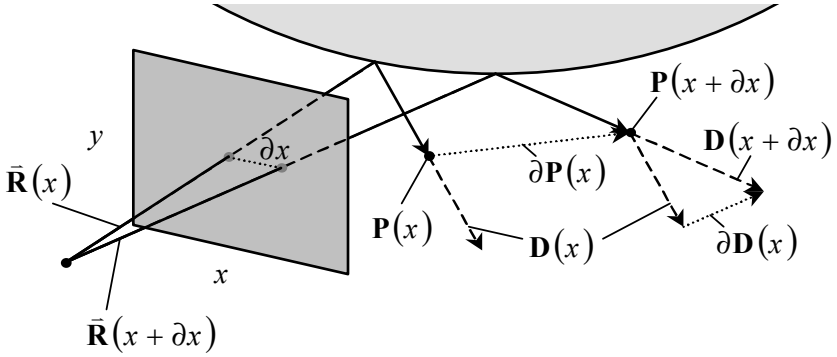

Figure 4.5: *A ray differential (after Igehy [63]).*

slightly larger. This is to compensate for invalid samples due to edges or reflection/refraction. Also, if no uniform kernel is used, we have to adjust the kernel size to account for different weights. Section 4.3.6 discusses some possible choices of kernels.

Let P_0 be the position where we want to apply the kernel in world-space. Note that due to refraction and reflection this point is not necessarily on the first surface a ray hits. A reasonable approximation is to project the image-space kernel footprint as a parallelogram into the tangent plane at P_0. This means we need to find

$$P_0 + ([-\Delta P_x, +\Delta P_x] \times [-\Delta P_y, +\Delta P_y]), \tag{4.16}$$

as illustrated in Figure 4.4.

With ray differentials [63], two differentially offset rays are traced in addition to the main ray (Fig. 4.5). Let $\partial P/\partial x$ be the positional component of the ray differential in x, propagated to the tangent plane at P_0 as described by Igehy [63]. Intuitively, this describes how the position in the tangent plane

changes if we move in x-direction in image-space. We can approximate ΔP_x to first-order as

$$\Delta P_x \approx \Delta x \frac{\partial P}{\partial x}, \tag{4.17}$$

where $\Delta x = r_{sk}$ is set to the image-space radius of frame k (Sec. 4.3.3). The formula for ΔP_y is analogous.

In practice, we have to extrude this parallelogram slightly along the normal N_0, to get a non-zero volume:

$$P_0 + \left([-\Delta P_x, +\Delta P_x] \times [-\Delta P_y, +\Delta P_y] \times [-\delta N_0, \delta N_0]\right). \tag{4.18}$$

An arbitrary kernel can then be embedded into this prism.

The advantage of using ray differentials is that they propagate correctly through perfect specular surfaces, taking the local curvature into account. If the renderer does not trace ray differentials, a simple geometric construction can be used, which is described in the following.

Let P_0 be the shading point, and N_0 its surface normal as above. We can construct a parallelogram with principal axes a_D and a_S in the tangent plane using the direction of the incident ray D:

$$\begin{aligned} a_D &= D + (D \cdot N_0) N_0 \\ a_S &= a_D \times N_0. \end{aligned} \tag{4.19}$$

If D and N_0 are (almost) parallel, a_D can be an arbitrary vector orthogonal to D, since the orientation in the tangent plane does not matter in this case.

Let $\widehat{a_D}$ and $\widehat{a_S}$ be the normalized versions of the vectors above. The extent in direction $\widehat{a_S}$ can simply be approximated using the intercept theorem:

$$r_S = \frac{d_{P_0}}{d_R} r_R, \tag{4.20}$$

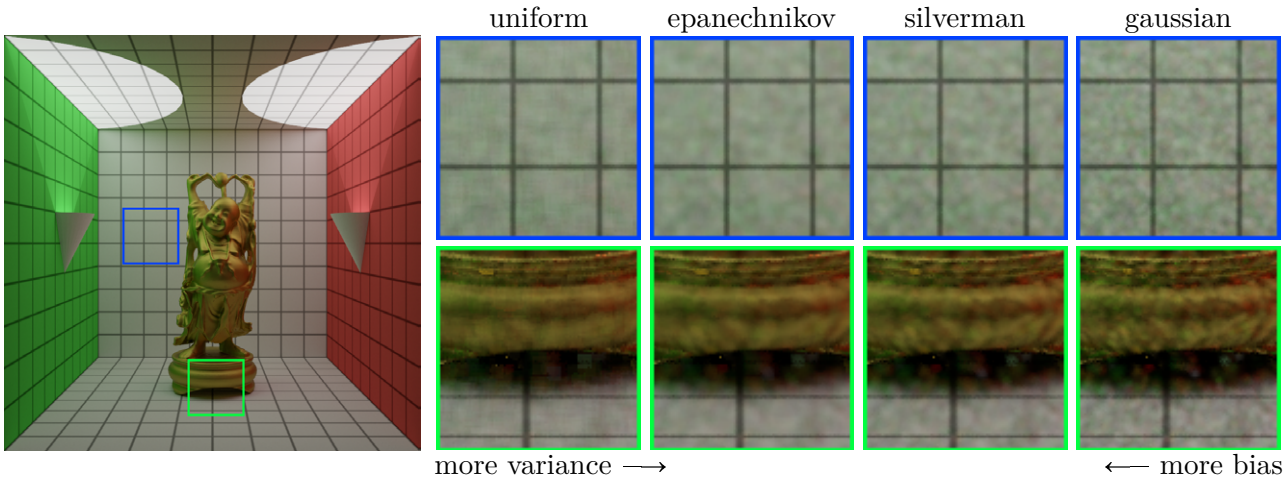

Figure 4.6: *Comparison of different kernels. Note that these images were intentionally rendered with too small a radius, in order to clearly show the kernel shapes.*

where d_{P_0} is the path length to P_0 (including refraction/reflection), d_R is the distance to some known reference plane (e.g. the near plane) and r_R is the extent on that plane. The extent in direction $\widehat{a_D}$ has to take surface slope into account, since we want an elongated parallelogram in case of a shallow angle of incidence. This elongation is inversely proportional to the cosine of the angle of incidence:

$$r_D = \frac{r_S}{\max(\epsilon, -(D \cdot N_0))}, \tag{4.21}$$

with a small $\epsilon > 0$ to prevent division by zero (and, in practice, excessive elongation).

The kernel can then be embedded into the prism

$$P_0 + ([-r_S\widehat{a_S}, +r_S\widehat{a_S}] \times [-r_D\widehat{a_D}, +r_D\widehat{a_D}] \times [-\delta N_0, \delta N_0])\,. \tag{4.22}$$

4.3.6 Choice of Kernel

The algorithm does not depend on a particular choice of kernel. (Although the derivations of the convergence rates in Appendix 4.A make some mild

assumptions.) Apart from the canonical uniform kernel, we have evaluated the Epanechnikov kernel as used by Walter [143], the Silverman kernel as used by Shirley et al. [128], and a simple windowed Gaussian kernel ($\sigma = 1/3$, cut-off at 3σ).

Figure 4.6 shows how the choice of kernel affects the result of radiance filtering. As one would expect, narrow kernels with strong fall-off (Silverman or Gaussian) introduce less bias, but also provide less variance reduction than wider kernels (Epanechnikov or uniform). The uniform kernel has the best variance reduction properties, but the reconstructed function is not continuous, which leads to blocky artifacts in practice. The Gaussian kernel does not reduce variance enough in practice, since many of the collected samples are "wasted" with a low weight. We typically use the Epanechnikov kernel or, if higher-order smoothness properties are necessary, the Silverman kernel.

4.4 Theoretical Analysis

Radiance filtering has two phases: a "preview" phase where the algorithm tries to keep variance at a constant, low level and bias is introduced; and a "correction" phase where the remaining variance and the bias from the preview phase are simultaneously reduced. Note that the phase is determined for each pixel independently and at any time some pixels may be in preview phase while others may already be in correction phase. It is also possible (but unlikely) for a pixel to drop out of the correction phase again, due to an unreliable variance estimate.

In the following subsection, we examine the asymptotic behavior of the overall error $\bar{\tilde{\epsilon}}_N = \frac{1}{N}\sum_{k=1}^{N} \tilde{\epsilon}_k$ after N frames, where $\tilde{\epsilon}_k = \widetilde{L_{ok}} - L_o$ is the error introduced in frame k. More specifically, we look at the bias $\mathrm{E}\left[\bar{\tilde{\epsilon}}_N\right]$ and the variance $\mathrm{Var}\left[\bar{\tilde{\epsilon}}_N\right]$ in each of the two phases of the algorithm.

4.4.1 Convergence Rates

In the preview phase, the variance is

$$\operatorname{Var}\left[\bar{\tilde{\epsilon}}_N\right] = O(1), \tag{4.23}$$

with $\operatorname{Var}\left[\bar{\tilde{\epsilon}}_N\right] \ll \operatorname{Var}\left[\bar{\hat{\epsilon}}_N\right]$ for smaller N, i.e. variance is approximately constant but usually much lower than with path tracing.

The accumulated bias is

$$\operatorname{E}\left[\bar{\tilde{\epsilon}}_N\right] = O\left(\frac{\log N}{N}\right). \tag{4.24}$$

We derive this result in Appendix 4.A.

In the correction phase, radiance filtering reduces to path tracing, so the variance is

$$\operatorname{Var}\left[\bar{\tilde{\epsilon}}_N\right] = O\left(\frac{1}{N}\right). \tag{4.25}$$

No additional bias is introduced in the correction phase, so the accumulated bias decreases with the number of samples:

$$\operatorname{E}\left[\bar{\tilde{\epsilon}}_N\right] = O\left(\frac{1}{N}\right), \tag{4.26}$$

for which we provide a short derivation in Appendix 4.B.

In fact, we can accumulate the path tracing estimators in a separate buffer and then switch to this buffer completely once a pixel enters the correction phase. With this little trick, all accumulated bias disappears instantaneously at the cost of an additional buffer. Unfortunately, the accumulated variance may also spike above the user-defined threshold after the switch, but this may be tolerable for some applications.

It is interesting to compare these convergence rates to progressive photon mapping and path tracing. Figure 4.7 shows plots for variance and bias which

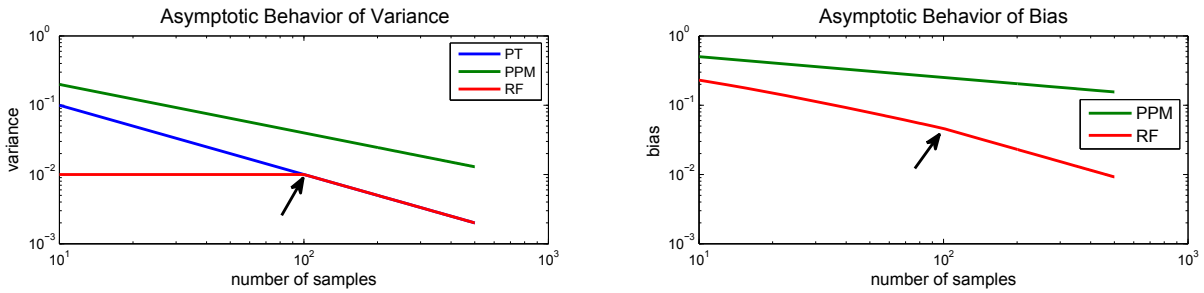

Figure 4.7: *Convergence rates for path tracing (PT), progressive photon mapping (PPM, $\alpha = 0.7$), and radiance filtering (RF) in a loglog-plot with arbitrary scale. (We use $\alpha = 0.7$ here, which was used in the original stochastic progressive photon mapping paper [55], although it was later shown that the optimal asymptotic convergence is reached with $\alpha = 2/3$ [70].) These plots illustrate the asymptotic behavior of the algorithms as described in Section 4.4. The arrows indicate the point where radiance filtering switches into correction phase ($N = 10^2$).*

reflect the different goals of the algorithms. Path Tracing in its ideal form is unbiased, but usually starts with high variance, which vanishes as $O(1/N)$. Progressive photon mapping lets variance and bias vanish simultaneously as $O(1/N^{\alpha})$ and $O(1/N^{1-\alpha})$, respectively, $\alpha \in (0, 1)$ [74]. The graph for radiance filtering clearly shows the focus on low-noise previews. Variance is constant but low at the beginning; bias is potentially high, but drops quickly. Note that $O(\log N/N)$ is asymptotically better than $O(1/N^{1-\alpha})$ for any fixed $\alpha \in (0, 1)$. Because in the preview phase radiance filtering does not reduce variance and bias *simultaneously*, it can shrink the kernel radius more aggressively than progressive photon mapping and consequently reduce bias faster. In the correction phase, the algorithm then removes the remaining bias and variance.

It ought to be mentioned that this discussion is primarily of theoretical interest. Specifically, although progressive photon mapping has the worst asymptotic convergence of the three algorithms, in practice it is usually the best algorithm and provides a visually converged image faster. We do not advertise radiance filtering as an alternative to progressive photon mapping or a fully-fledged rendering algorithm, but as an extension of path tracing that can provide a visually pleasing image very early in a progressive rendering setup. However, the many conceptual similarities to progressive photon mapping suggest a comparison. The similarities were already pointed

out in Section 4.3; in the following section we discuss the conceptual differences.

4.4.2 Comparison with Progressive Photon Mapping

Conceptually, the main difference to progressive photon mapping is that radiance filtering distributes radiance samples in image-space during path tracing, not flux samples in world space during a separate photon tracing path. The advantage is that radiance filtering can expect a sample density of one sample per pixel. This means we can adapt the kernel to try to collect as many samples as needed to reach a user-defined threshold on variance. This almost eliminates the splotches that are typical for early stages of progressive photon mapping. The assumption that neighboring pixels contain relevant radiance samples breaks down in the presence of geometric edges and complex perfect specular objects, but in most practical cases there will be at least some spatial coherence in the image. A disadvantage in comparison to progressive photon mapping is that radiance filtering inherits the weaknesses of path tracing with respect to SDS paths, where photon mapping is clearly the superior algorithm.

Also, radiance filtering does not perform a classic density estimation at the measurement points and is more akin to a simple filter. (There is no area in the radiance filtering estimate and the samples carry radiance, not flux.) As a consequence, it does not suffer from bias that is due to wrong area estimates. Specifically there is no boundary bias and no topological bias [111]. (Although the "pdf bias" is similar to topological bias in that it occurs on curved surfaces.) There is, however, proximity bias, and the typical artifacts resulting from it (e.g. light leaks and blurred caustics). This is especially a problem early in the rendering process, when kernel sizes are relatively large, see Section 4.5. Note that with radiance filtering no samples are deposited on invisible surfaces, so light does not leak from invisible regions (e.g. back walls) as in photon mapping. In that sense there is no occlusion bias.

4.4.3 Comparison with Image Filtering

Approaches based on (cross) bilateral filtering or similar edge-aware filters usually filter pixel values and try to limit bias by reducing the influence of strongly biasing pixels. The biggest problem of these approaches is that in most practical situations the filter will either be too wide and blur fine details in geometry or texture, or it will be too narrow and will not reduce variance enough. One can preserve textures by multiplying the diffuse BRDF by a filtered irradiance estimate, but this approach does not work well for non-diffuse surfaces. Another problem is that most filters do not handle sharp antialiased edges correctly, because the pixel value is a linear combination of the regions adjacent to the edge and not present in the undiscretized signal itself. Addressing these two issues was the primary motivation for our work.

Blurring is significantly reduced with our approach of just filtering the incident radiance instead of the pixel values. Also, radiance filtering averages before the pixel reconstruction filter is applied, so it handles antialiased pixels correctly, which is another advantage over image filtering. A disadvantage is the slightly higher overhead due to repeated BRDF evaluations.

4.5 Results

4.5.1 Comparison with Pixel Filtering

As the purpose of radiance filtering is to improve upon methods based on cross bilateral filtering, we first compare it to the approach introduced in Chapter 3 ("pixel filtering"). We use the progressive scheme described in Section 3.3.3 and the adaptive bandwidth selection described in Section 3.6.3).

Figure 4.8 shows a setting where radiance filtering performs best: Mostly diffuse surfaces predominantly lit by smooth light. Since the surfaces are

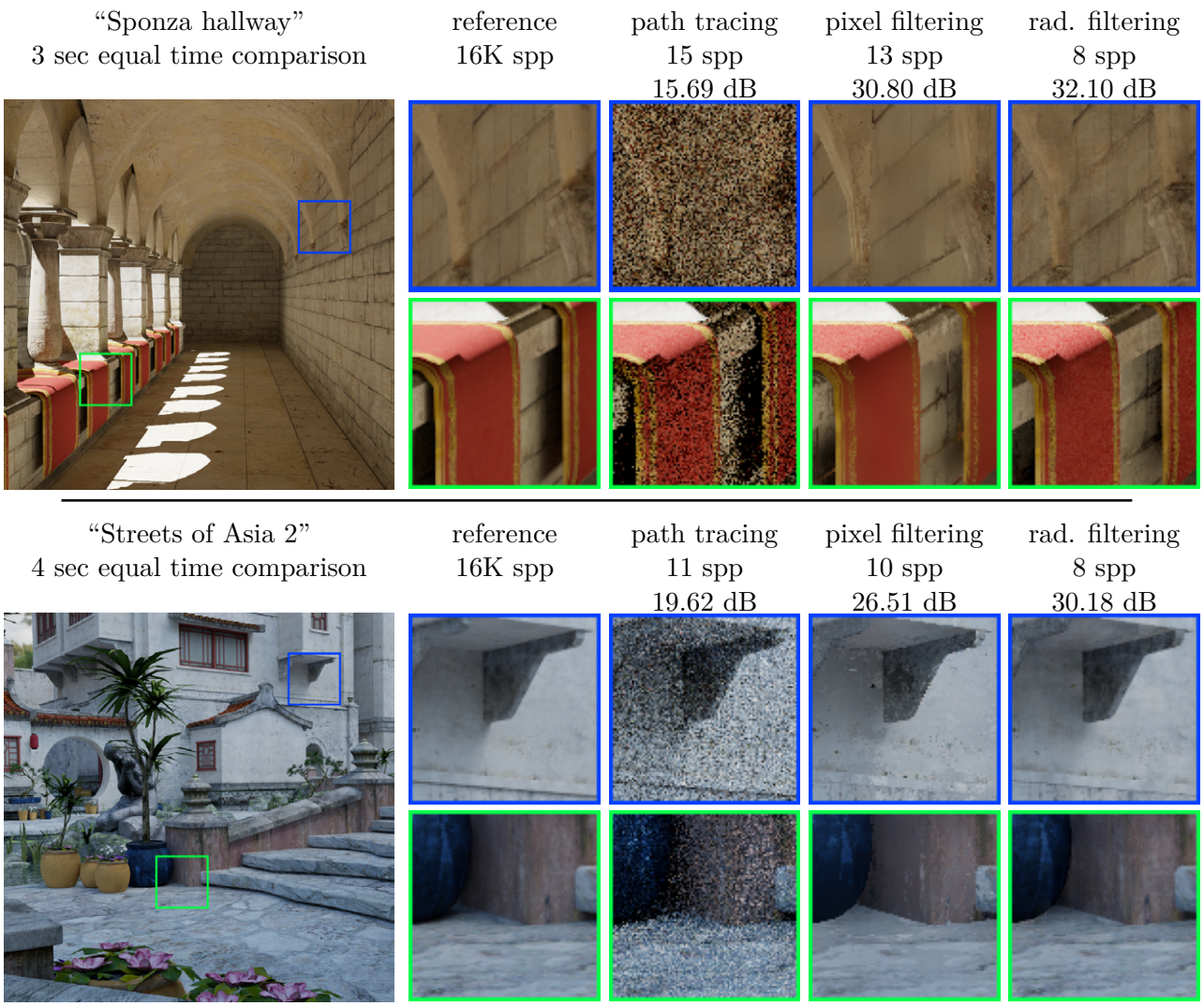

Figure 4.8: *Equal-time comparison of path tracing, pixel filtering, and radiance filtering. For "Sponza hallway", both filtering algorithms were applied only to the indirect illumination. For "Streets of Asia 2", both filtering algorithms were applied to direct and indirect illumination. For each algorithm, the number of samples per pixel (spp) and the PSNR relative to the reference solution (in dB) are given.*

scene	$\tilde{t}_8$ (seconds)	$\hat{t}_8$ (seconds)	$\tilde{t}_8 - \hat{t}_8$ (seconds)	relative overhead
"Sponza hallway"	0.282	0.171	0.111	39%
"Streets of Asia 2"	0.383	0.305	0.13	23%

scene	$\bar{\tilde{t}}_8$ (seconds)	$\bar{\hat{t}}_8$ (seconds)	$\bar{\tilde{t}}_8 - \bar{\hat{t}}_8$ (seconds)	relative overhead
"Sponza hallway"	2.761	1.365	1.396	51%
"Streets of Asia 2"	3.195	2.44	0.755	24%

Table 4.2: *Timings for the scenes in Figure 4.8. Top: a single frame; bottom: multiple frames. $\tilde{t}_8$ is the time frame 8 took to compute with radiance filtering, $\hat{t}_8$ is how long frame 8 took with path tracing (1 spp). $\bar{\tilde{t}}_8$ and $\bar{\hat{t}}_8$ are the accumulated timings for 8 frames, i.e. how long it took to compute an image with 8 spp. Relative overhead is the percentage of rendering time spent on the filtering procedure.*

mostly diffuse, we did not use the MIS-approximation and the ν-factor for these test scenes.

Although both approaches achieve a very good noise reduction, radiance filtering has a constantly higher peak signal to noise ratio (PSNR). The green blow-up of "Sponza hallway" also shows a typical artifact of (cross) bilateral filtering that does not occur with radiance filtering: color bleeding due to unrecognized edges. (Although for diffuse surfaces this can be addressed by filtering irradiance, see the following subsection.) A problem from which both approaches suffer is light leaks (note the missing shadows under the red carpet). This is a manifestation of proximity bias and vanishes as the rendering converges.

"Streets of Asia 2" in Figure 4.8 shows radiance filtering applied to direct illumination. The illumination is relatively smooth (cloudy day), but there is still some directionality in the light, as can be seen in the blue blow-up (soft shadow under the balcony). Both filtering techniques retain this feature. Another subtle global illumination effect that would not be easily possible with simplified lighting is the indirect light that falls from the glossy blue vase onto the pillar in the green blow-up. The image filter again has problems with fine texture details. The blow-ups also show its tendency to produce jagged edges.

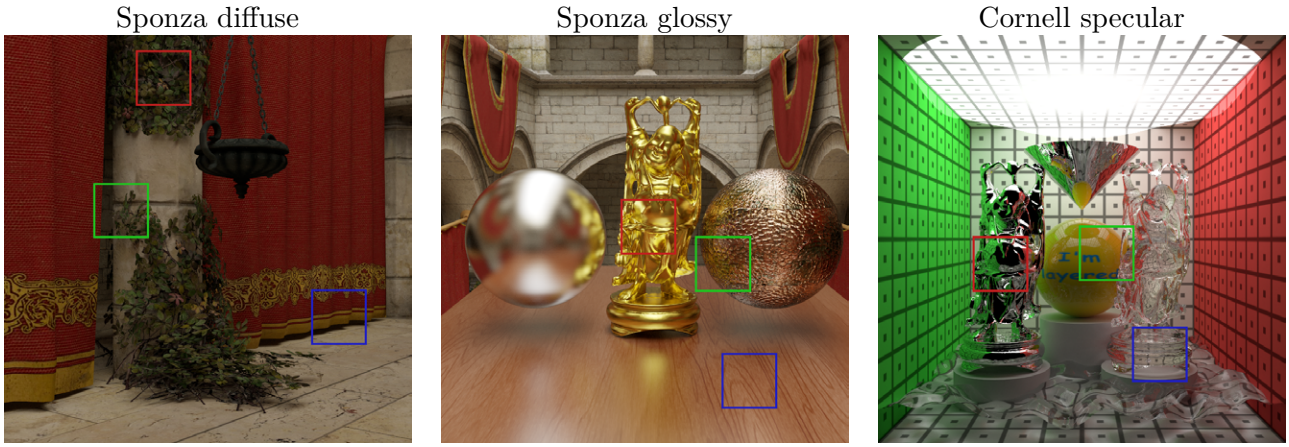

Figure 4.9: *Three test scenes (diffuse, glossy, specular). The marked regions correspond to the blow-ups in Figures 4.11, 4.12, 4.13.*

Table 4.2 shows timings and the relative overhead for the renderings in Figure 4.8. As one would expect, the overhead is quite substantial during the early frames, when the radii are still large – but it decreases as the rendering progresses. For all images the increased PSNR is worth the overhead, as the equal-time comparisons show.

4.5.2 Comparison using Three Distinct Scenes

In this subsection, we present results for the three scenes shown in Figure 4.9, one with diffuse, one with glossy and one with perfectly specular materials. We compare our method to Dammertz et al. [28] (*atrous*) and Schwenk et al. [125] (*headlight* – this is the method from Chapter 3, but now it is using postmultiplication by filtered irradiance). We have adapted *atrous* for progressive rendering using the bandwidth selection from Chapter 3. To limit the ringing/stippling artifacts of *atrous* and improve the overall quality, we used three iterations instead of five (which costs some performance). An option to reduce these artifacts without performance loss is jittering, but since this brings back some noise, we did not use it for our tests. We have also extended *atrous* with the range buffer used by *headlight* (the "virtual flash image") and modified it to work with filtered irradiance even for glossy surfaces (similar to Bauszat et al. [7]). (In practice it is actually not really the irradiance,

scene	total	pt	filter
"Sponza diffuse"	3.906	2.232	1.674 (43%)
"Sponza glossy"	3.915	2.088	1.827 (47%)
"Cornell specular"	5.036	3.346	1.689 (34%)

Figure 4.10: *Timings (in seconds) for radiance filtering with 8 spp applied to the scenes in Figure 4.9. With only 8 spp the filtering overhead is relatively high, but it decreases with more spp because the kernels shrink.*

but only the irradiance from a small cone, due to BRDF importance sampling – which is why the approach works surprisingly well in many cases.) This hybrid (*atrous2*) combines properties of the original *atrous* (fast filter, good preservation of geometric edges) with those of *headlight* (good preservation of specular reflections) and Bauszat et al. [7] (postmultiplication for glossy materials). For diffuse surfaces, the classic edge-aware filtering methods (*atrous*, *headlight*, *atrous2*) use postmultiplication by filtered irradiance. *headlight* uses the progressive filtering scheme described in Chapter 3, but not the blending operator. The other techniques filter every frame in order to avoid the aliasing problems with storing geometry information for multiple spp. Geometry-aware look-ups as used by Bauszat et al. [7] could help with this, but we did not implement these.

All images are 768×768 and were rendered with a custom GPU path tracer based on Optix [99]. The system was a Core i7 with 3.33GHz (6 cores, each with Hyperthreading) and a GeForce GTX580. Figure 4.10 shows timings for the three scenes in Figure 4.9.

As metrics we used the peak signal to noise ratio (PSNR) and the HDR Visual Difference Predictor 2 (HDR-VDP2) by Mantuik et al. [84]. For HDR-VDP2 we used the default viewing conditions and set the display to a 1080p CCFL-LCD viewed from 50 cm distance. Since the metrics give only a relatively rough indication of the perceived difference, we recommend that readers compare the original images in the supplementary material themselves.

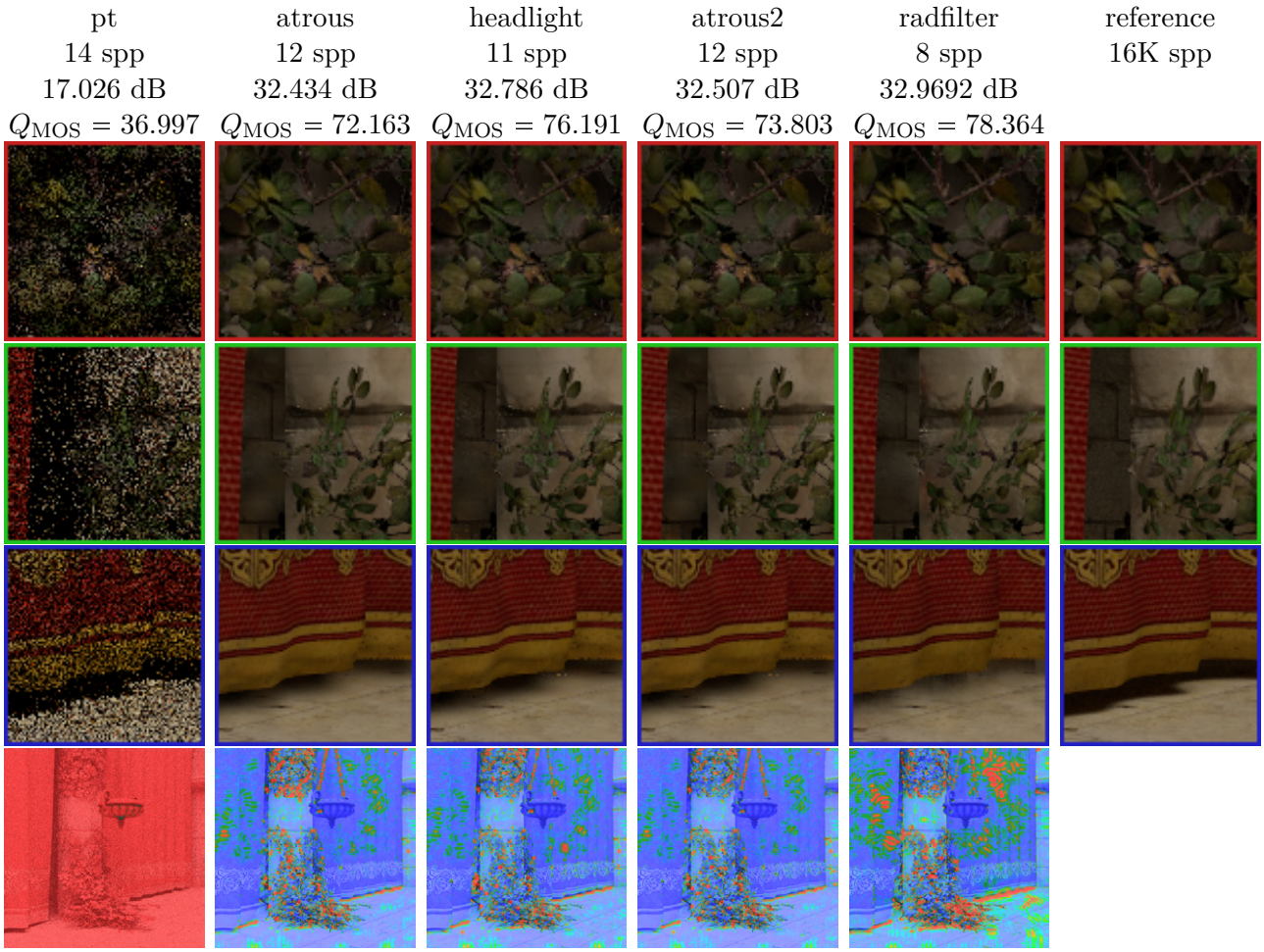

Figure 4.11: *Blow-ups of the "Sponza diffuse" scene. This is a 4 seconds equal-time comparison. We give the PSNR in dB and the Q_{MOS} (image quality mean-opinion-score) prediction of HDR-VDP2 (0–100). Higher is better in both cases. The last row shows the P_{map} visualization of HDR-VDP2 (probability of detection per pixel).*

4.5.2.1 Diffuse

For the diffuse scene (Fig. 4.11), all methods perform very well. Edge-aware filtering has slightly more problems with normal maps (pillar in green blow-up), while radiance filtering has more problems with light leaks (shadow in blue blow-up). One region where radiance filtering leaks less light than the other approaches is the pillar under the leaves in the red blow-up. Here, the projection into world-space prevents radiance filtering from collecting (too bright) samples on the leaves. Edge-aware filtering guards against this with a positional sigma – but if this is decreased splotches will appear on sloped surfaces (e.g. the ground plane). Apart from that all methods handle the geometric complexity of the leaves very well.

While quality-wise there is no significant advantage for diffuse scenes (if post-multiplication by irradiance is used), we would argue that radiance filtering is easier to use. First, there is no need to tune several sigmas specifically for a scene (or a view of a scene). Second, radiance filtering treats diffuse and glossy surfaces in the same way, so there is no need to implement a separate code path for glossy surfaces, where postmultiplication by irradiance cannot be used.

4.5.2.2 Glossy

In the glossy test scene (Fig. 4.12), we have a metallic sphere that demonstrates how glossy reflections are preserved and can be used to judge the overall glossiness of the objects (all use the same exponent). We also have a golden Buddha to demonstrate performance with high curvature normal variations, a copper sphere with a gloss and a normal map, and a wooden plate with a simple layered material (Ashikhmin-Shirley).

headlight blurs the reflections on the golden Buddha relatively strongly (red blow-up). This is because is relies only on the headlight to encode normal information. *atrous* and *atrous2* can preserve this feature better. On the other hand they have problems with sharp edges, where noise is not reduced enough.

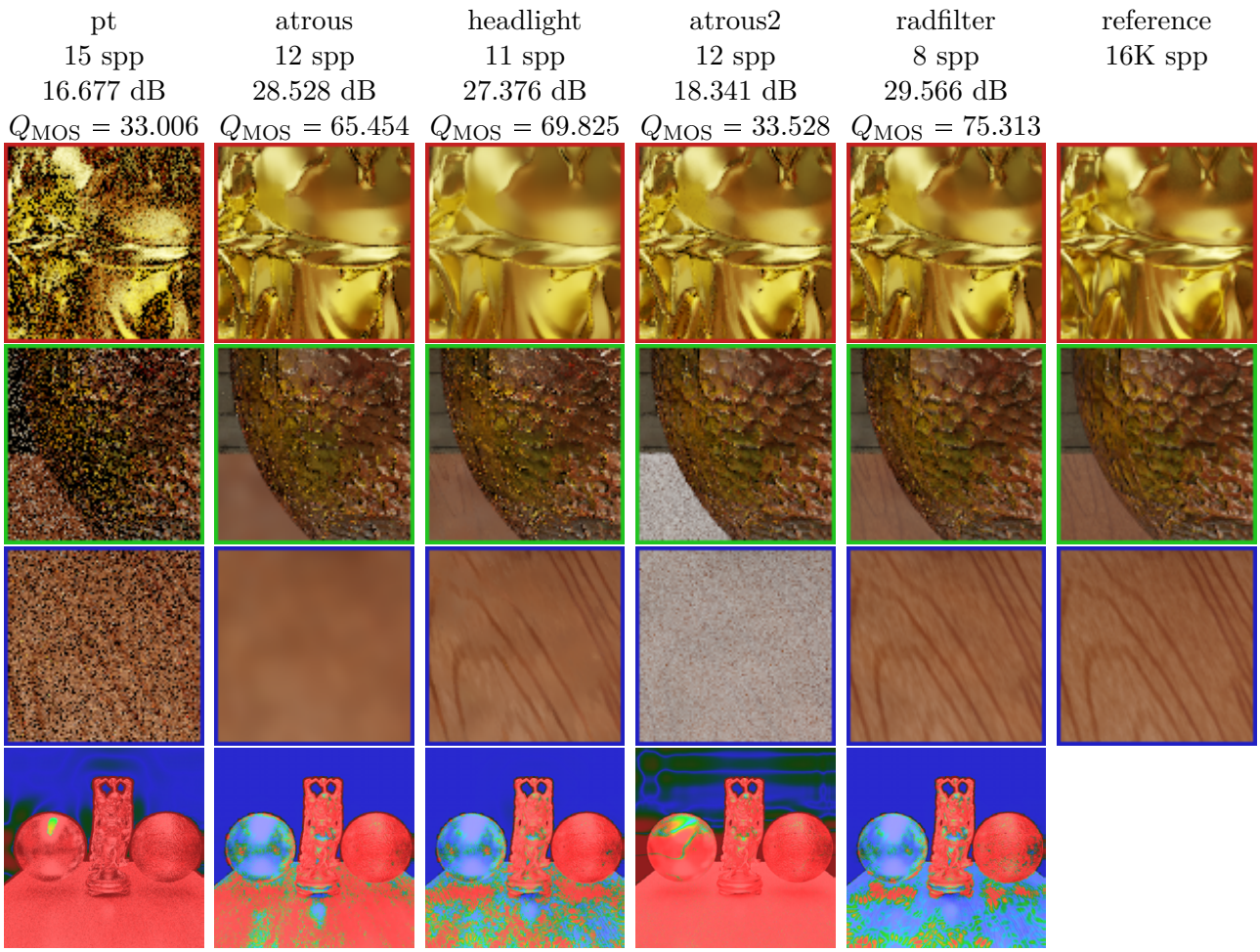

Figure 4.12: *Blow-ups of the "Sponza glossy" scene with PSNR, Q_{MOS}, and P_{map}. This is a 4 seconds equal-time comparison.*

radfilter offers a good compromise between smoothing and preservation of the reflections, although the difference is relatively subtle.

The copper sphere (green blow-up) offers a similar picture. However, *atrous* and *headlight* leave a higher noise level here. This is because they have to work with one global sigma for normals (and textures in the case of *headlight*) and thus have to compromise globally between preserving features and reducing noise. A way to automatically adapt these sigmas locally, similar to what Kalantari and Sen recently proposed [69], would greatly improve these approaches. *radfilter* produces the best results here, *atrous2* is very close.

On the wooden plate, *atrous* loses the texture details, because it respects features in textures only via the "rt" buffer, which is too noisy in this test case. Despite that, *atrous* scores surprisingly high in the metrics. *headlight* preserves the texture relatively well, but fine details are lost, too. *atrous2* is the worst method here, because the strategy of filtering irradiance breaks down. The glossy layer "sees" the irradiance from above (sampled by the diffuse layer), which leads to an overestimation of glossy reflections. *radfilter* is the only method that preserves the texture and the glossy reflection well.

4.5.2.3 Specular

First of all, it has to be mentioned that the specular scene (Fig. 4.13) contains light paths that cannot be sampled by path tracing because they have zero probability (paths hitting the point light from inside the specular cone). But since these paths are not available to any method in the test, the comparison is still valid.

On the chrome Buddha, *atrous* loses the texture pattern in the reflection (red blow-up). *atrous* could preserve this to some extent via the "rt" buffer, but it was not possible to find a sigma that preserves the texture and still allows enough smoothing in the rest of the image (specifically for the caustics). *headlight* and *atrous2* can preserve the texture. *radfilter* preserves the texture too, and also copes a little better with the variation in the normals.

The yellow sphere (green blow-up) is a simple layered material (diffuse under specular coating). *atrous* loses the texture and the reflection. *headlight* and *atrous2* can preserve the texture, but lose the reflection. In theory they could also preserve the reflection, but once again the sharper sigma needed to do that would degrade image quality in other regions (in this case on the glass surfaces). *radfilter* preserves the texture best, but the reflection remains noisy. Unlike the layered material in the glossy scene, this material has a perfect specular component. So, the radiance samples for neighboring pixels can

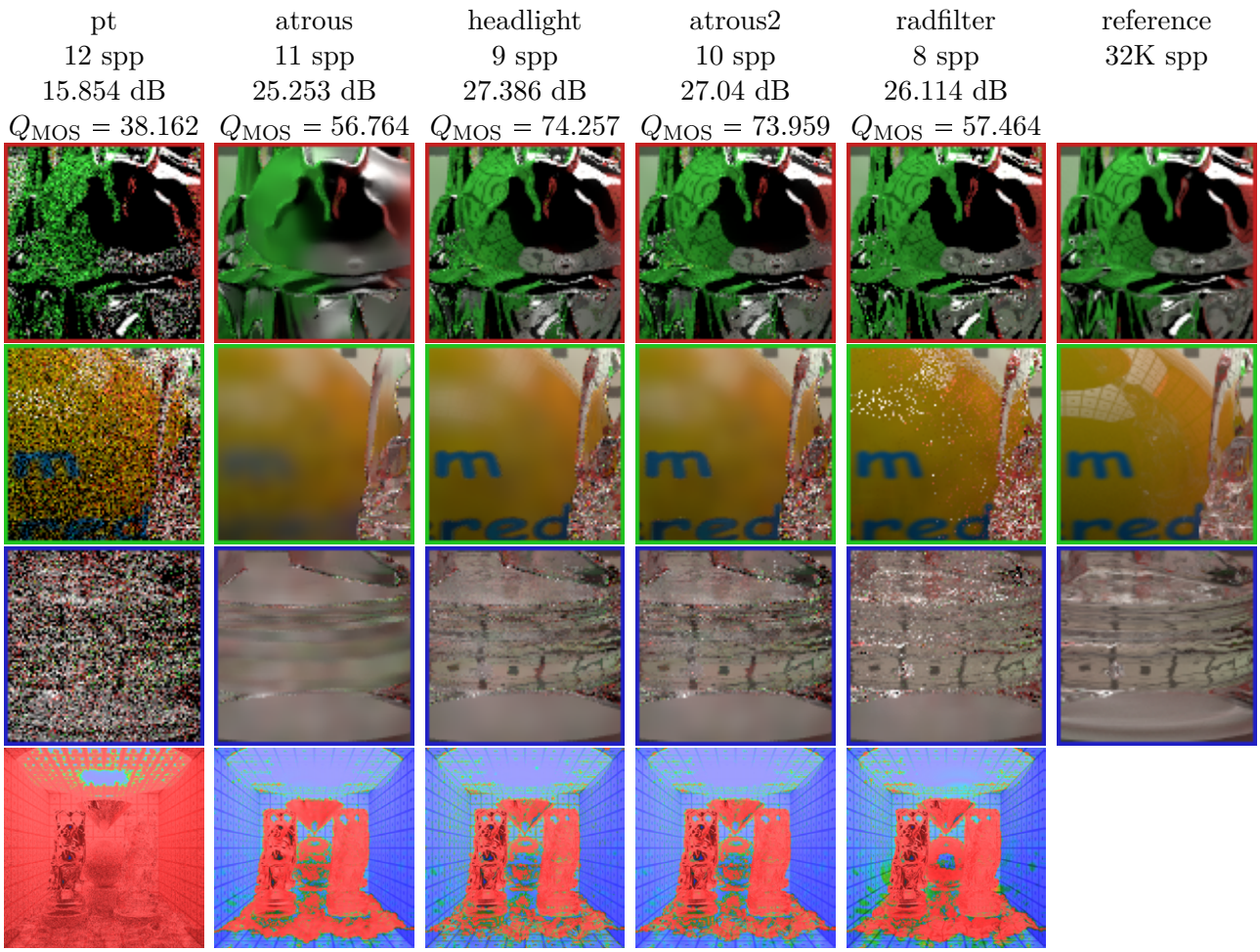

Figure 4.13: *Blow-ups of the "Cornell specular" scene with PSNR, Q_{MOS}, and P_{map}. This is a 5 seconds equal-time comparison.*

end up in completely different world-space locations, which prevents radiance filtering from being effective.

The problem is worse for the glass Buddha (blue blow-up), where neighboring paths take diverging routes recursively. (But even there there is at least some noise reduction). *headlight* and *atrous2* handle the glass very well. *atrous* loses the texture pattern in the reflection/refraction. All approaches blur the caustic under the Buddha.

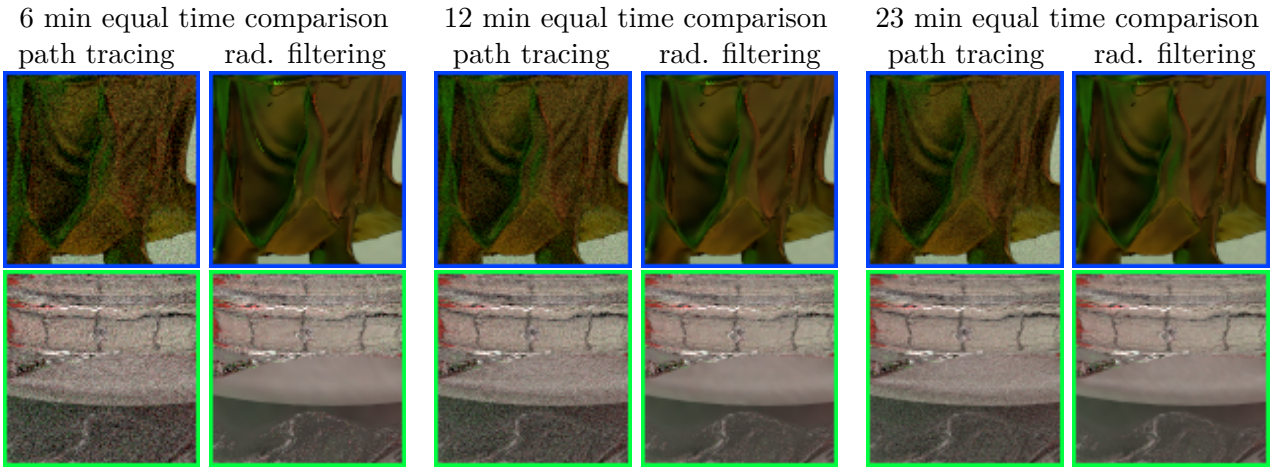

Figure 4.14: *Blow-ups from a modified "Cornell specular" scene after longer rendering times to illustrate the convergence behavior for difficult scene features. Note how the caustic under the glass Buddha gradually becomes sharper while the algorithm maintains a low noise level throughout the rendering process.*

4.5.3 Additional Results

Figure 4.14 shows that radiance filtering can also be used to get final renderings quickly, although the focus is on fast previews. We demonstrate this with "Cornell specular", which is the most difficult scene for radiance filtering in the test set. The green blow-up shows how the caustic starts to form over time while the noise-level is constantly low. After 23 min the image rendered with radiance filtering is very close to the reference, although some bias still remains.

Figure 4.16 analyzes the variance and bias for two pixels. The blue pixel is under almost uniform incident radiance, and we expect radiance filtering to introduce little bias in this case. The green pixel is in a penumbra region, and therefore should receive a relatively high bias during radiance filtering. The graphs confirm this: The blue pixel has little bias and the algorithm succeeds in keeping the variance constant and low. The green pixel starts with a high bias, which quickly drops, as stated in the theoretical analysis. It is also near a geometric discontinuity, which means that the filter cannot find enough valid neighboring samples to reach the threshold on variance (but the variance is still much lower than with path tracing).

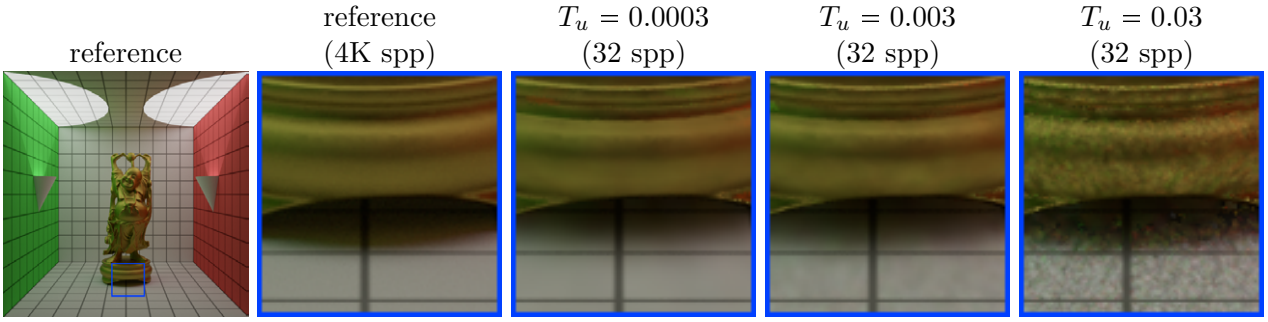

Figure 4.15: *Renderings with three different thresholds on variance. A higher threshold allows more variance and means less bias. However, the remaining variance manifests itself as low-frequency noise.*

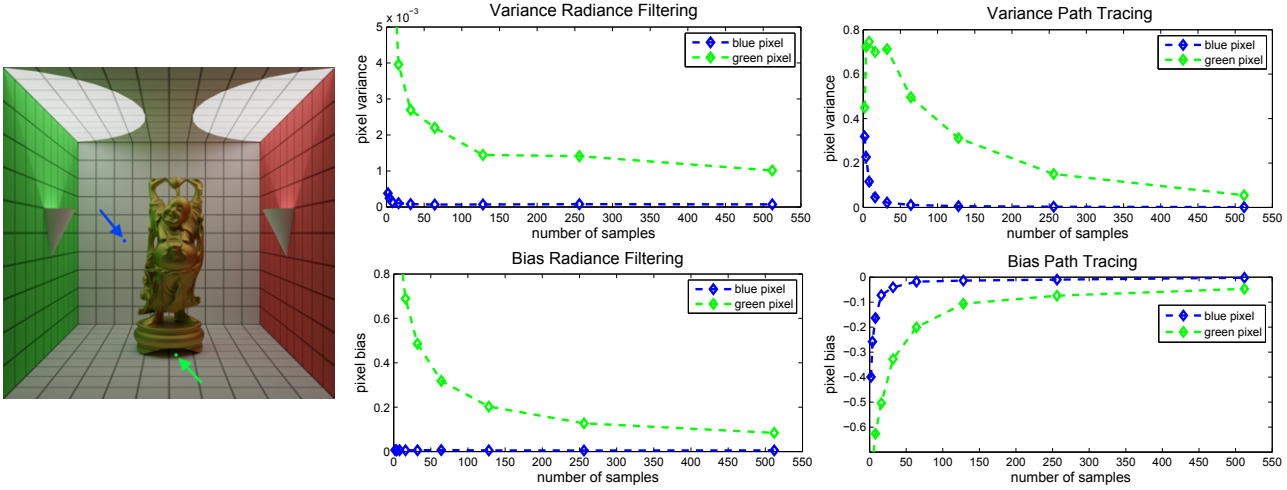

Figure 4.16: *Bias and variance for two pixels. For the blue pixel, radiance filtering introduces little bias. For the green pixel, much bias is introduced in the early frames (but it vanishes rather quickly). Bias and variance were estimated by rendering 128 sequences of images with different seeds and comparing them to the reference solution. Bias and variance are given relative to the reference solution and are computed using the actual pixel values after tone mapping. This is why path tracing has bias in the graphs, too. Note that the variance plots use different scales – the variances for radiance filtering are actually two orders of magnitude lower.*

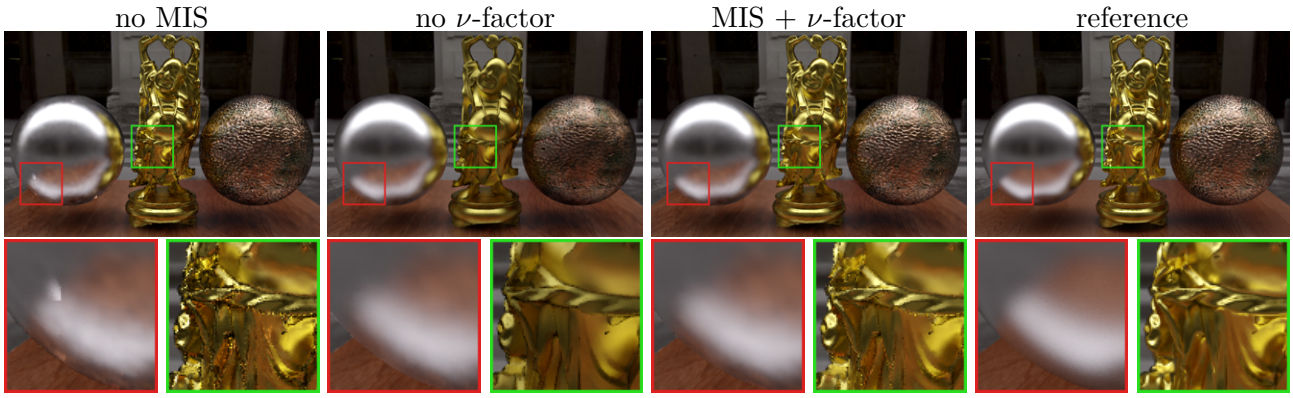

Figure 4.17: *The effects of our MIS approximation and the ν-factor (with 4 spp).*

Figure 4.17 illustrates how the approximate MIS weights and the ν-factor play together for glossy objects. Without MIS, spike noise can result in splotches. With our MIS approximation, the result is overly smooth and blurs the sharp glossy reflection. The ν-factor has the effect of sharpening the result again for highly glossy, curved surfaces while retaining the smoothness of the MIS approximation. For planar surfaces, the reflections remain blurred, but get sharper over time.

Figure 4.18 shows the Buddha from Fig. 4.17 with varying glossiness. For high glossiness, the sharp ν-factor reduces smoothing on sharp edges, but overall, radiance filtering achieves a reasonable compromise between smoothing and preservation of sharp glossy reflections. With an exponent of approximately 4096 the Buddha clearly appears darker and artifacts at sharp normal transitions become quite objectionable (see blow-ups). Also, on planar surfaces, the reflections tend to get blurred too much. Therefore we usually do not apply the filter for exponents greater than 4096, but continue tracing the rays (so the rightmost column in Figure 4.18 is there merely for illustration). However, in principle the filter remains usable for very high exponents – just not for very fast previews, because one has to invest more samples to get good results.

Figure 4.19 shows how radiance filtering deals with distribution effects like depth of field or motion blur. The method still works in the presence of these effects. However, since radiance filtering only filters illumination, it cannot

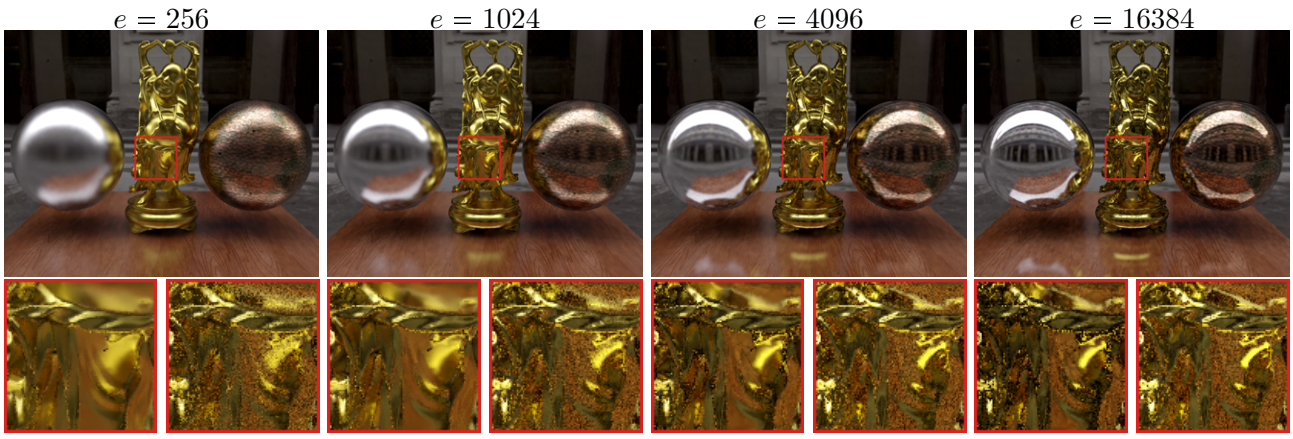

Figure 4.18: *The scene from Fig. 4.17 with varying glossiness (Blinn-exponent from 256 to 16384). The blow-ups show radiance filtering (rad) vs. path tracing (pt). This is not an equal-time comparison (pt is actually favored), but conveys a rough idea of the noise reduction.*

reduce noise that results from an increased variance in the hit points of primary rays. The effects can be seen in the blow-ups: there is a significant noise reduction, but texture and geometry still have to be sampled at high rates to get a high-quality image. Note that while the same is true for stochastic supersampling of pixels, much lower sampling rates are needed in that case and the problem usually is much less severe in practice.

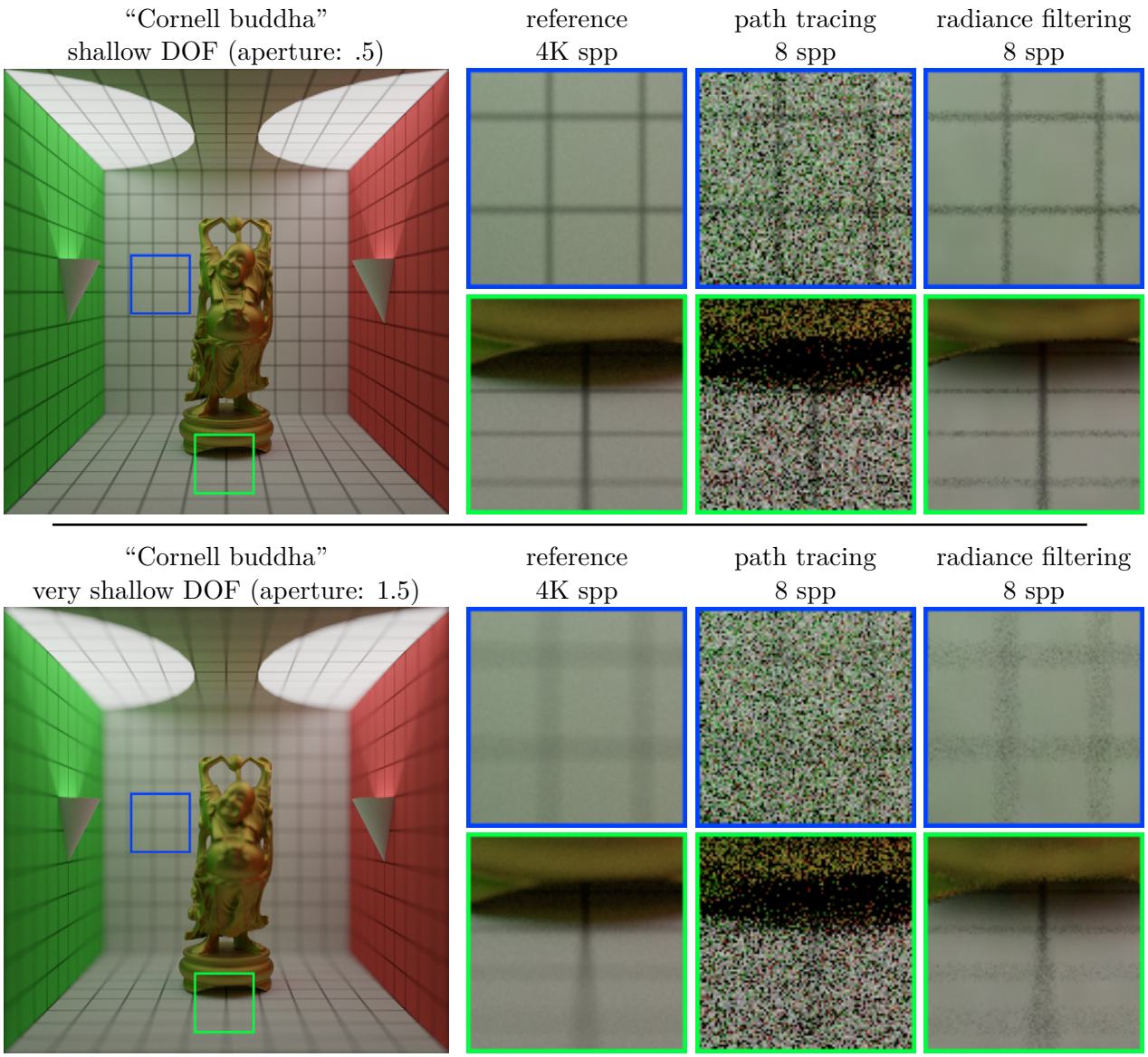

Figure 4.19: *Radiance filtering applied to renderings with depth of field. The blue region is completely out of focus, while the green region is (at least partially) in focus. Note that this is a comparison with equal sample counts, not equal rendering time.*

4.6 Optimizations

This section presents some optimizations for the original approach. The first optimization improves performance with perfect specular reflecting/refracting surfaces. The second optimization reduces blurring in the incident illumination. The third optimization adapts the blending operator introduced in Chapter 3 to radiance filtering.

4.6.1 Hybrid Approach for Perfect Specular Surfaces

Probably the biggest weakness of the original radiance filtering algorithm is the reduced efficiency with perfect specular surfaces that reflect and refract light at the same time. Reflection and refraction lead to diverging rays and the subsequent filtering step can have problems finding valid radiance samples.

There are a number of brute-force solutions that immediately come to mind. For example, one can force deterministic behavior at reflecting/refracting surfaces and only trace one path at a time, effectively decomposing the scene into layers. This solves the problem of diverging rays and leads to much better noise reduction with radiance filtering, but we regard the resulting popping artifacts as worse than the noise that would remain otherwise. Another approach is to trace all possible paths at a time (at least to a certain depth) and store multiple radiance samples. A third possibility is to (adaptively) supersample reflecting/refracting surfaces.

We present a fourth option here: a hybrid filtering approach that combines radiance filtering with the pixel filtering method described in Chapter 3. The basic idea is to apply each algorithm where it is strong: pixel filtering to reflecting/refracting surfaces and radiance filtering to all remaining surfaces. To do this, we mark all pixels with paths that hit a reflecting/refracting surface before they can deposit a radiance sample in the path tracing pass (Fig. 4.20 middle). Note that this includes more than just directly visible surfaces, for example it includes reflections of reflecting/refracting surfaces in a perfect

Figure 4.20: *Left: the "Cornell specular 2" scene contains many reflecting/refracting surfaces. Middle: mask for the hybrid filter; pixels that are reflecting* and *refracting at the same time are green, pixels that are* either *reflecting* or *refracting are red. Pixel filtering is only applied to green pixels, radiance filtering to the rest. Right: range buffer for pixel filtering.*

mirror. Then a range buffer is computed for these pixels (Fig. 4.20 right) and the filter is applied as described in Chapter 3. To all remaining pixels, radiance filtering is applied as described in this chapter.

Figure 4.21 compares the hybrid approach to pixel filtering and radiance filtering for a modified "Cornell specular" scene. The hybrid filter performs better than each of the components alone. Note that it is also slightly faster than radiance filtering (in that it can compute more samples in the same time). This is because the relatively expensive radiance filter is not applied to the masked pixels.

4.6.2 Bilateral Kernels

The original radiance filtering can blur sharp features in the illumination, such as sharp (indirect) shadows and sharp caustics. This is because the (linear) kernels do not depend on the radiance signal and therefore cannot respect "edges" (sharp changes) in the illumination.

A simple way to remedy this is to include a range component into the kernel, effectively turning it into a bilateral filter. However, using the incident radiance in a range kernel is impractical, because of its high dimensionality (two dimensions for the position on the scene's surfaces plus two for the incident

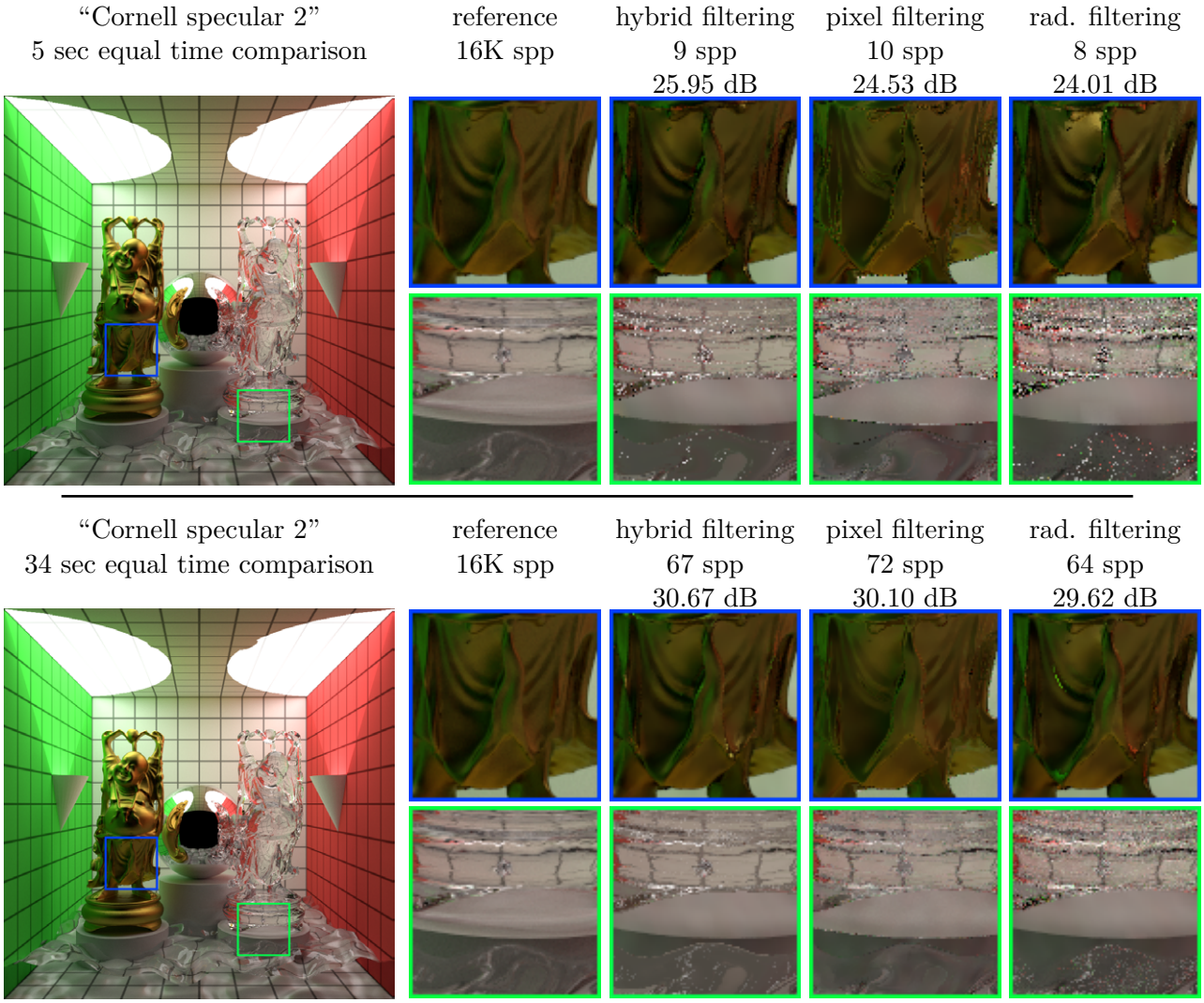

Figure 4.21: *Equal-time comparison of hybrid filtering, pixel filtering, and radiance filtering. The hybrid approach is better than pixel filtering or radiance filtering alone.*

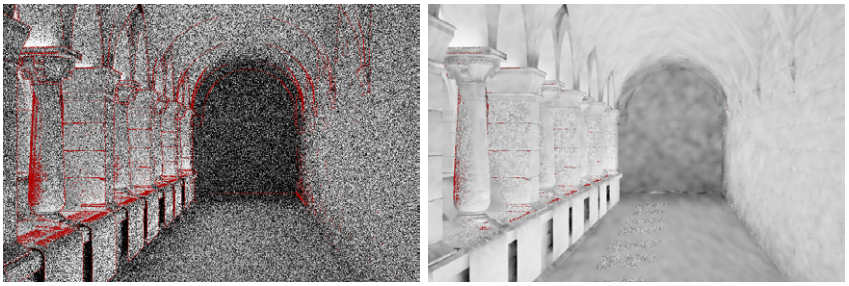

Figure 4.22: *The per-pixel irradiance is not reliable with only a few samples per pixel (left). Filtering it alongside the radiance values provides a smooth approximation (right).*

direction). A better choice is the irradiance, which also captures illumination changes, but is only two-dimensional and can be saved as one additional 3-vector per pixel.

Unfortunately, the irradiance estimates are unreliable at the beginning of rendering, with only a few samples per pixel. Therefore, we use the same procedure as in Section 4.3.4 and filter the irradiance alongside the radiance values. This produces a smooth irradiance estimate, see Figure 4.22. However, filtering also smoothes the sharp features in the illumination we want to detect with the irradiance estimate. A consequence is that the approach described here needs a short warm-up phase to work properly. Experience shows that sharp features are detectable after 16-32 spp in typical scenes, which is when the modified filter starts to pay off.

The modified weights to plug a bilateral kernel into Equation 4.10 are

$$w_j = \sigma(x_j, x)\ \nu(n_j, n)\ \rho\left(\tilde{\hat{E}}(x_j), \tilde{\hat{E}}(x)\right), \tag{4.27}$$

where ρ is the range kernel that works on the filtered irradiance estimates $\tilde{\hat{E}}(x_j)$. We use a Gaussian with a standard deviation of 0.1 $\tilde{\hat{E}}(x)$ and mean $\tilde{\hat{E}}(x)$. This relative standard deviation produces good results in HDR environments. It also prevents an additional free parameter. Note that the filtered irradiance is filtered with the *unmodified* kernel, not the bilateral kernel, to prevent feedback effects (amplification of wrongly detected edges).

Figure 4.23 shows a comparison of the original approach and the extended version with a bilateral kernel. The subtle shadow under the carpet is pre-

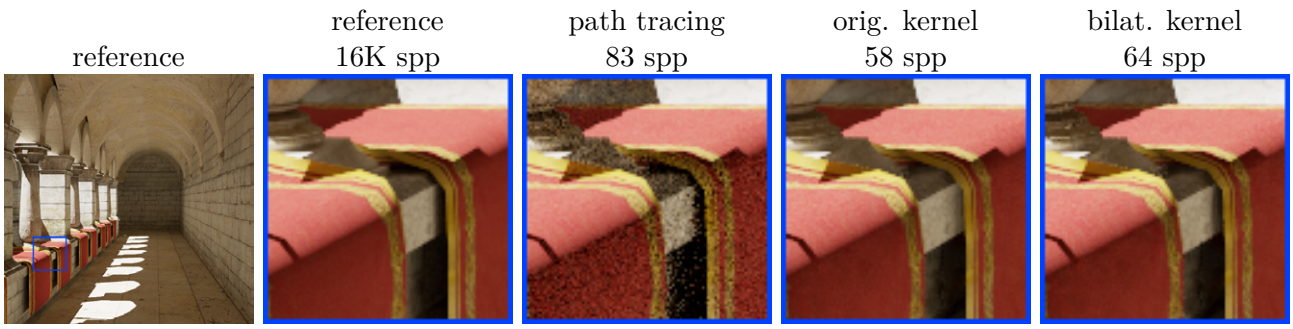

Figure 4.23: *Bilateral kernels can mitigate the light leaks from which the original approach suffers. The original kernel removes the subtle shadow under the red carpet almost completely. The bilateral kernel preserves the shadow much better.*

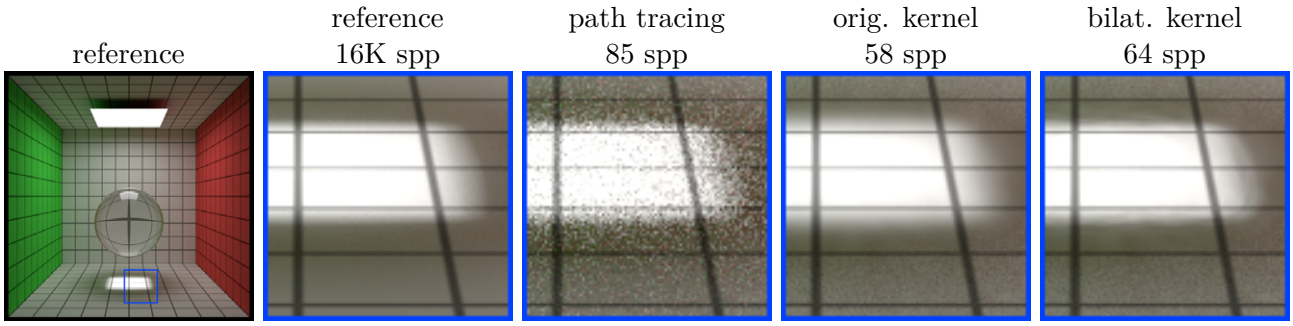

Figure 4.24: *Bilateral kernels can also reduce the blurring of caustics. Compared to the original kernel, the bilateral kernel renders the boundary of the caustic more sharply.*

served much better by the bilateral kernel. Figure 4.24 shows that caustics are preserved better, too.

The overhead of this extension consists of the evaluation of ρ and the filtering (and storage) of the irradiance estimate. This amounts to approximately 10 percent of running time in our implementation and one additional buffer in rendering resolution (containing 3 floating point values per pixel).

Note that the bilateral kernel is not equivalent to a smaller kernel radius. A smaller kernel radius would reduce blurring, but would also introduce low-frequency noise in the rest of the image. The bilateral kernel effectively only reduces the kernel radius in high-gradient regions. This helps to preserve

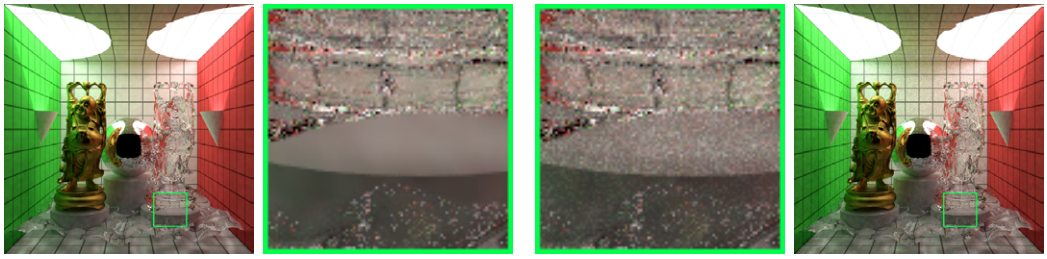

Figure 4.25: *The original blending approach (left) vs. filtering with blending (right). The blending operator leaves some high-frequency noise in the image, which can hint at features the filter smoothed out, e.g. the subtle caustic under the glass Buddha. This is a psychological effect, the numerical error is often higher with blending. Rendered with 32 spp.*

sharp changes in the illumination, while leaving the filtering performance for (roughly) homogeneous regions untouched. Also note the difference to the classic bilateral filter as used in Chapter 3: here, we filter incident radiance with a world-space kernel, not pixel values with an image-space kernel.

We have also tried the trilateral filter [24], which includes the gradient into the range kernel. It smoothes towards a piecewise linear solution, not a piecewise constant solution. This can prevent the "shock waves" that the bilateral filter tends to produce in high-gradient regions. However, we have found that in general the improvement is not worth the additional overhead and use the bilateral kernel as default.

4.6.3 Blending Operator

In the spirit of the hybrid filtering approach presented above, we can also adapt the blending operator from Chapter 3 to radiance filtering. This is a straightforward extension: in addition to the target variance, the user also specifies what percentage of the variance should be "filled up" with the original, unfiltered samples. This opens up another degree of freedom for reaching the threshold on variance. We can adapt the bandwidth *or* the blend factor.

The filter effectively converts high-frequency noise to low-frequency noise by

blurring the incident radiance. The blending operator blends in the original, unblurred samples that are contaminated by high-frequency noise. With the original approach, we could only reduce blurring by reducing the filter's bandwidth. The resulting low-frequency noise would produce a splotchy image. With the blending operator, we can also reduce blurring by blending in some high-frequency noise.

Figure 4.25 shows this extension in action. This particular example allowed a relatively high amount of noise in the final image. The numerical error is lower for the purely filtered image, but the image with blending hints at the subtle caustic under the Buddha, which is almost completely lost in the filtered image. Attenuating high-frequency noise in this way, instead of trying to remove it by converting it into low-frequency noise can be beneficial for some previews.

4.7 Additional Remarks and Future Work

The dominant artifacts of radiance filtering are light leaks and blurred caustics. Whenever the assumption of smooth (indirect) illumination is violated, filtering the illumination will introduce significant bias.

A possible approach to mitigate this problem is to incorporate an estimate of bias into the algorithm. Currently, the only objective is to reduce variance. If an estimate of bias was available at rendering time, we could try to limit the *overall* rendering error and allow more variance in regions where the filter introduces much bias. So far, we have not been able to develop a fast and reliable method to estimate bias for radiance filtering. Bias depends on second-order derivatives of L_i, which are hard to estimate reliably in an undersampled image, and fast approximations proved to be not accurate enough. However, related work exists for photon density estimation [53], and it should be possible to incorporate these results into radiance filtering with future work.

Another artifact that can occur, especially during the early frames, is splotches

due to spike noise. If the radiance of a sample is much higher than the expected value, the energy is not distributed in a large enough area by the filter and a bright splotch is visible (Fig. 4.26). A method to withhold such "outliers" from the image, similar to the one proposed by DeCoro et al. [29], could reduce these artifacts.

We would like to mention that the artifacts of radiance filtering are in general visually less objectionable than the artifacts introduced by cross bilateral filtering. Most viewers will regard fading or blurred shadows and blurred caustics as less severe as remaining noise or blurred textures. Therefore, radiance filtering often produces visually more pleasing results, even if objective metrics (like PSNR) are similar. We compare the typical artifacts in Figure 4.26.

It is also worth mentioning that our method is compatible with many commonly used optimizations such as stratified, adaptive and importance sampling. For adaptive sampling, the assumption that there is one sample per pixel (per frame) may not be true anymore. This can lead to wrongly estimated kernel sizes, but is usually not a problem in practice. Stratification can lead to wrongly estimated kernel sizes, too, since combining well-stratified samples may reduce the variance better than completely independent samples. Again, this is usually not a problem in practice as it can be counterbalanced by raising the target variance. If no proper importance sampling is performed, radiance filtering has to cope with the increased variance. This too is usually not a big problem in practice, however, if the lack of importance sampling leads to significantly more spike noise, the splotches described in the preceding paragraph will occur.

Finally, it may make sense to use radiance filtering to quickly fill a per-pixel radiance cache and use this for importance sampling. The result would be similar to the recently presented importance caching algorithm [47].

Figure 4.26: *Comparison of artifacts. Top: reference solution; middle: radiance filtering; bottom: pixel filtering. The undersampled insets show the path traced input to illustrate the noise reduction achieved by the filtering algorithms.*

4.8 Conclusions

We have presented a novel approach for noise reduction that is especially tailored to interactive progressive path tracing. Other than popular image filtering techniques, radiance filtering does not simply filter pixel values or irradiance, but the incident (indirect) radiance. This way, radiance filtering can significantly reduce the variance of a rendered image without blurring details in geometry or texture while still being applicable to non-diffuse surfaces. The primary use case is to provide fast, reliable previews of global illumination. It is also consistent, retains the conceptual simplicity of path tracing, is compatible with importance and stratified sampling, and is easy to integrate into existing renderers. The method works best for smooth indirect illumination, but smooth direct illumination can be handled, too.

The most pressing issues that have to be addressed by future work are the performance of the method with reflecting/refracting objects and sharp features in the indirect illumination that are currently blurred (sharp shadows and caustics). The bilateral kernels we have proposed mitigate the problem, but a more efficient solution is necessary. Another interesting avenue for future work is to limit the overall error, not just the variance, by incorporating an estimate of bias into the algorithm.

Appendix 4.A Bias in Preview Phase

In the following, we assume the radiance signal over the image plane is finite and bounded everywhere. We assume M radiance samples have been distributed over the image plane (typically one per pixel). We also assume that the weight of radiance samples outside the current image-space radius is zero, that all weights are non-negative, and that the maximum weight assigned to a single radiance sample is one.

The bias introduced in frame k is

$$\begin{aligned}\mathrm{E}[\tilde{\epsilon}_k] &= \mathrm{E}\left[\widetilde{L_{ok}} - L_o\right] \\ &= \mathrm{E}\left[\sum_{j=1}^{M} w_k(X_j)B(X_j)L_i(X_j) - L_o\right] \\ &= \sum_{j=1}^{M} \mathrm{E}\left[w_k(X_j)B(X_j)L_i(X_j)\right] - L_o, \end{aligned} \tag{4.28}$$

where we have compactly written

$$B(X_j) = f_r(x, \Omega_{i,j}, \omega_o)(n \cdot \Omega_{i,j}) \quad \text{and} \tag{4.29}$$

$$L_i(X_j) = \breve{L}_i(X_j, \Omega_{i,j})/p_\omega(\Omega_{i,j}). \tag{4.30}$$

We have interpreted the position of the j-th radiance sample as a random variable X_j with some pdf p_{x_j}; the associated $\Omega_{i,j}$ are interpreted analogously.

$\mathrm{E}\left[w_k(X_j)B(X_j)L_i(X_j)\right]$ is the expected value of the j-th radiance sample in frame k when weighted relatively to the shading point at x. Let C be the most biasing radiance sample on the image plane, weighed with 1 (the maximum weight). This maximum exists, because the radiance over the image plane is assumed to be bounded.

We can then estimate the worst case by substituting this maximum for all radiance samples with $w_j > 0$. At the same time, we can discard all samples with $w_j = 0$ and reindex the remaining samples with $l = 1, \ldots, M_k$, which then enumerates all samples that contribute in frame k. With this, we arrive at

$$\begin{aligned}\mathrm{E}[\tilde{\epsilon}_k] &\leqslant \sum_{l=1}^{M_k} C - L_o \\ &= M_k C - L_o = (2r_{sk} - 1)^2 C - L_o \\ &= O\left(r_{sk}^2\right). \end{aligned} \tag{4.31}$$

By substituting Eq. 4.14 into Eq. 4.2, we obtain a formula for the image-space

kernel radius in frame k:

$$r_{sk} = \frac{1}{2}\sqrt{\frac{\mathrm{Var}[\widehat{L}_i]}{(2k-1)T_u}}\frac{4}{\pi} = \Theta\left(\frac{1}{\sqrt{k}}\right). \tag{4.32}$$

The world-space radius shrinks proportionally to that and with Eq. 4.31 we can express the bias introduced in frame k directly in terms of the frame number k:

$$\mathrm{E}[\tilde{\epsilon}_k] = O\left(\frac{1}{k}\right). \tag{4.33}$$

For the accumulated bias, we have

$$\mathrm{E}\left[\bar{\tilde{\epsilon}}_N\right] = \frac{1}{N}\sum_{k=1}^{N}\mathrm{E}[\tilde{\epsilon}_k] = \frac{1}{N}\sum_{k=1}^{N}O\left(\frac{1}{k}\right) = O\left(\frac{\log N}{N}\right). \tag{4.34}$$

Because the asymptotic behavior of the harmonic series is $O(\log N)$:

$$\sum_{k=1}^{N}\frac{1}{k} = \gamma + \log N + O\left(\frac{1}{N}\right) = O\left(\log N\right), \tag{4.35}$$

where γ is the Euler-Mascheroni constant.

To gain some intuition, we can look at a fixed pixel grid without supersampling and reflecting/refracting objects and an image-space box filter. With radius one (one pixel) we have only one sample inside the kernel: the correct one. Bias is zero. With radius two (3×3 pixels) the influence of the correct sample is 1/9, the influence of potentially biasing samples is 8/9. With radius three (5×5 pixels) the influences are 1/25 and 24/25. The exact bias will vary depending on the samples, but even if we assume each biasing sample has the value of the sample in the image that throws the estimate off most, the influence of biasing samples will only grow quadratically.

Appendix 4.B Bias in Correction Phase

We begin by observing that each pixel with finite variance (which means that the pixel converges with pure path tracing) reaches the correction phase with a probability of one. Let $\widetilde{V}_N \approx \mathrm{Var}[L_i]$ be the variance estimate that drives the algorithm. With a probability of one we have

$$\lim_{N\to\infty} \widetilde{V}_N \in \mathbb{R}, \tag{4.36}$$

and therefore, because the target variance grows as $T_N = (2N-1)T_u$ (Eq. 4.14), there exists a N_0 after which T_N is greater than $\widetilde{V}_N$:

$$\exists N_0 \forall N > N_0 : \widetilde{V}_N < T_N = (2N-1)T_u. \tag{4.37}$$

In other words, after N_0 frames the algorithm is and stays in correction phase with a probability of one.

$\mathrm{E}\left[\bar{\tilde{\epsilon}}_{N_0}\right]$ is the bias accumulated up to this point (which is asymptotically bounded by $O(\log N/N)$ as shown in the previous Section). For $N > N_0$ we have

$$\begin{aligned}\mathrm{E}\left[\bar{\tilde{\epsilon}}_N\right] &= \frac{N_0}{N}\,\mathrm{E}\left[\bar{\tilde{\epsilon}}_{N_0}\right] + \underbrace{\frac{1}{N}\sum_{k=N_0+1}^{N} \mathrm{E}\left[\tilde{\epsilon}_k\right]}_{0} \\ &= O\left(\frac{1}{N}\right),\end{aligned} \tag{4.38}$$

because after entering correction phase, no additional bias is added (all $\mathrm{E}[\tilde{\epsilon}_k]$ are zero for $k > N_0$).

5 System Architecture

This chapter presents a practical and non-intrusive approach to using multiple, potentially very different rendering back-ends in a distributed rendering system. The approach allows back-ends to be plugged into the OpenSG infrastructure, without impairing their strengths and without burdening the back-ends or the application with details of the cluster environment.

5.1 Introduction

The problem we address in this chapter seems simple at first sight: how to support different renderers for the same application layer. It also seems to have been solved many times already. Many systems in computer graphics have support for different rendering back-ends. In interactive graphics, it is quite common to support different rasterization APIs (e.g. OpenGL and Direct3D). In offline rendering, many systems support several rendering algorithms, each with their individual advantages and disadvantages. One route that is often taken, is to find a common abstract imperative interface and to implement this interface for each renderer. Another approach, mainly used in offline rendering systems, is to define a common scene representation and let each back-end work on this representation. However, the problem we had to solve for our system goes much deeper.

Our system, InstantReality [43], is a generic framework for interactive visualization and VR/AR-applications. It has to support a wide range of applications (from visualization of large CAD models over game-like virtual

environments to photorealistic rendering) on a wide range of platforms (from CAVEs to Laptops, Fig 5.2). No single rendering architecture meets the requirements of all our customers, so we decided to support specialized, potentially quite different, rendering back-ends (from GPU-rasterization to progressive ray tracing). Yet, we still wanted to use the same scene description and application layer for all scenarios, mainly in order to ease application development with existing tool chains. This problem statement implies two important issues that, in combination, are not sufficiently addressed by previous approaches.

The first issue is that the back-ends can be based on extremely different architectures and paradigms. Furthermore, they often come packaged as closed source, third party libraries that define their own API and scene description. For example, a cloud-based ray tracer and a GPU-rasterizer may have very different interfaces. This makes it difficult to abstract rendering back-ends with a single imperative interface. From this issue we derived the first central design decision: we should abstract the scene, not the renderer. On the other hand, it does not make sense to include high-level, application-specific features into the abstraction, so the scene abstraction should only consist of a few low-level concepts.

The second issue is that we support features such as stereoscopic rendering and rendering in a cluster (e.g. for multiprojector environments). We want to use these features with different rendering back-ends, even if the back-ends themselves do not support them. It seems reasonable to implement these features in a layer on top of the rendering back-ends. On the other hand, the application layer should be mostly agnostic to which hardware-configuration the application runs on. This brought us to the second central design decision: we should hide the computing environment (hardware configuration) from both, the application layer and the rendering back-end.

In order to implement these two design decisions, we use a mediator layer between application layer and rendering back-ends. The mediator translates a common scene representation into the back-end's specific representation, keeps both of them in sync with incremental updates, and triggers the back-end's renderer whenever an image is needed. Our mediators are based on

OpenSG [32], an open source scene graph library with sophisticated support for clustering and multithreading.

Of course, the insights above and the idea of a mediator layer are not new – but we are not aware of a system that has implemented them with such consequence as ours. The contribution we make with this chapter is therefore the description of a pragmatic, practice-proven approach to use different rendering back-ends with a common application layer in distributed systems. In particular, we show an interesting way to propagate changes and how to support clustering based on OpenSG's multithreading concept. We regard our work primarily as valuable addition to the OpenSG infrastructure, but we also think that the ideas and design principles described here are applicable to similar systems.

5.2 Related Work

Many approaches to using exchangeable rendering back-ends with a common application layer have been described. Also, a lot of literature exists on rendering in clusters. But surprisingly few publications deal with the combined problem of how to efficiently support different rendering back-ends for distributed systems.

5.2.1 Rendering in a Cluster

Techniques for cluster-based rendering can be broadly categorized on the basis of where the distribution happens. We restrict ourselves to three popular examples here, for more techniques and details we refer to the recent survey by Staadt et al. [130].

Humphreys et al. [62] developed the Chromium library, a low-level approach that streams commands of the underlying graphics API. This approach is very generic (in theory any OpenGL application can be clustered), but typically consumes a rather high bandwidth. Bierbaum et al. [13] implemented the

other extreme in VR Juggler. There, clustering happens at the application layer, which is very flexible, but requires manual adaption of each application to its intended runtime environment. Reiners et al. [32] occupy a middle ground with OpenSG. They distribute the scene, not the entire application, and stream only scene updates, not the entire graphics command stream. This is the architecture we based our work on.

In addition to these rendering-centric approaches, there are systems whose primary concern is to distribute arbitrary computing tasks in a cluster. They contain the problem we address in this chapter as a sub-problem, but the top-level architecture usually just defers it to black-box rendering nodes. PaTraCo [44], KoDaVis [37], and FlowVR [2] are recent examples.

5.2.2 Exchangeable Rendering Back-ends

Approaches for exchangeable rendering back-ends range from low-level abstractions of imperative graphics APIs to high-level abstractions that work with a common understanding of a "scene".

OpenGL [147] is probably the most popular graphics API. In theory, anybody can develop an OpenGL interface to their rendering back-ends, and it will work with OpenGL-compatible applications. In practice, however, since the API strongly reflects the underlying rasterization pipeline, it has only been used for rasterization (as far as we know). Although Dietrich et al. [30] developed OpenRT, an API similar to OpenGL for ray tracing, with the intention to ease porting of OpenGL applications to ray tracing [31]. While OpenGL is centered around feeding primitives to a rasterization pipeline, OpenRT already offers a (very primitive) scene abstraction that is better suited to ray tracing.

Some ray tracing frameworks take the scene abstraction to the extreme and rely on scenes that basically just have to provide an "intersect" method. For example Pharr et al. [103] do this with PBRT, and Georgiev et al. [50] follow a similar approach with RTfact. While this works well for ray tracing, other

back-ends may require scenes that expose more internal structure in order to process them efficiently.

Many scene graphs allow custom renderers as plug-ins (e.g. [33, 131, 95, 108, 32], to list only a few). The typical mechanisms are custom traversals and extensible nodes (via callbacks), usually a combination of both. If the scene-graph is well-designed, one can come up with an adapter for most rendering back-ends with acceptable time and effort. Even if the adapter has to translate the whole scene graph into the back-end's internal structures. Döllner and Hinrichs' [34] Virtual Rendering System, used for example by Steinicke et al. [131], stands out because it was specifically designed to work well with different back-ends and regards adapter components as part of the core architecture. Döllner and Hinrichs [34] also contains a survey of previous approaches, which we will not re-iterate here.

A recent approach that is very closely related to our work is Rubinstein et al. [116] and their Real-time Scene Graph (RTSG). RTSG is a scene graph whose core is based on a stripped-down version of X3D/VRML. It allows the application to attach different rendering back-ends and provides an efficient way of propagating changes via callbacks.

Berthelot et al. [12] introduced the Scene Graph Adapter, an architecture for mixing different scene graphs in one application at runtime, without an offline conversion step. The approach consists of two standardized interfaces, Format Wrapper and Renderer Wrapper, that have to be implemented for each 3D format and renderer, respectively. A central Scene Graph Adapter instance then mediates between several Format and Renderer Wrapper instances by mapping nodes and calls.

5.2.3 Comparison with Related Work

The most important aspect that sets our work apart from related approaches is the "fat" mediator layer based on OpenSG. In fact, OpenSG can be regarded as the front-end towards the application layer, providing a common interface for the mediator and the back-end.

Compared to approaches based on imperative APIs [147, 30, 31] our approach provides a higher abstraction of the rendering back-end behind a declarative interface (a stripped-down scene graph). The loose coupling allows more generic back-ends; the scene graph adaption allows each back-end to work with a suitable scene representation for high performance.

Approaches that treat the scene as a black box (or at least a very, very opaque box) [103, 50] have the complementary problem: they cannot query enough information to efficiently map a scene. In contrast to them, our scene-graph-based approach allows for querying relevant data and supports an efficient mapping of the scene.

Approaches based on scene graphs that use custom callbacks during traversal (e.g. [33, 95, 108, 32], but especially Döllner and Hinrichs [34]) are usually quite close to a sweet spot as far as scene abstraction is concerned. However, purely relying on this approach leaves the concerns of multithreading and clustering to the application layer or the rendering back-end. Another issue not sufficiently addressed by the traversal approach is that the adapter has no efficient way of detecting and propagating changes (without a full traversal). RTSG [116] offers a solution to this problem that seems to be extensible to clustering (RTSG was used in the URay framework [109] for distributed rendering, but it is not explicitly mentioned how changes are handled). Compared to RTSG, our approach provides a cleaner separation of application layer, mediator layer, and back-end and uses another mechanism to handle incremental updates. The Scene Graph Adapter by Berthelot et al. [12] leaves the multithreading and clustering issues unattended and focuses solely on scene mapping. We use an adapter that is similar in spirit in our mediators, but it is based on a low-level OpenSG scene. This results in a cleaner interface than the bloated wrappers the Scene Graph Adapter has to offer and takes the burden of multithreading and clustering off the shoulders of the application and the back-end. The downside is that we have to pay the memory overhead of the OpenSG scene as an intermediate representation.

5.3 Our Approach

In this section, we first lay down the most important requirements that guided our design, then we give an overview of the architecture itself, and finally we discuss two particularly interesting aspects of the system in more depth: incremental changes and clustering support. Our approach is based on OpenSG [32] and we assume a basic familiarity with concepts like Fields, FieldContainers, Aspects, and ChangeLists [140].

5.3.1 Requirements

Our most important requirements were:

Extensibility and Generality. The system should be able to integrate new rendering back-ends relatively painlessly. It should also be general enough to handle back-ends coming from very different application areas and following different design paradigms.

Non-intrusiveness. Neither the application layer nor the rendering back-end should need any changes or extensions in order to work together. (Back-ends may extend the application layer to expose specialized functionality, as described below, but basic functionality should be possible without touching both.) This is important because we want to support commercial libraries as back-ends that usually come as black boxes with an unalterable interface.

Clustering and stereo. The system should provide (at least basic) support for rendering in a computer-cluster (tile-based and cooperative) and stereoscopic rendering, even if a back-end itself does not support it.

Rendering performance. Of course, the system should allow each renderer to play out its strength. After all, that is why we want multiple, specialized rendering back-ends. Integrating a back-end into the system should hamper the rendering performance as little as possible.

Fast incremental updates. Not only the raw rendering performance is important, but also how updates are propagated from the application layer to the rendering back-end (and sometimes the other way around). The system should provide an efficient (in terms of runtime and usability) solution to this problem.

Ability to extend application layer. While the system should rely as much as possible on a common low-level abstraction of a scene, sometimes it is practical to expose attributes that are specific to a certain back-end in the application layer. An example are extended material attributes for a ray tracer. The system should provide a mechanism to pass on such data.

Mixed (hybrid) rendering. The system should provide (at least basic) support for mixing different renderers during the generation of one image. For example, it should be possible to render large static geometry with a back-end optimized for that purpose and to render dynamic 3D GUI elements in the same scene with another back-end.

5.3.2 Basic Design

Figure 5.1 shows a schematic overview of our design. The application layer (in our case an X3D-browser [8]) manages the high-level application logic and mirrors the state of its scene in a low-level OpenSG scene graph [32, 106]. The OpenSG-layer (and all layers below) are only concerned with the current state of components (Transforms, Materials, Geometries, etc.) and not with procedures that change this state (animation, physics, I/O, etc.).

The mediator layer is the main subject of this chapter. It has to be implemented for each rendering back-end (although parts can be reused as shown in Section 5.4.1). It usually consists of a viewport, a scene adapter, and a context. With the viewport, the mediator can hook itself into OpenSG's rendering infrastructure. It is the only part of a mediator that is mandatory. The scene adapter translates the OpenSG scene into the renderers

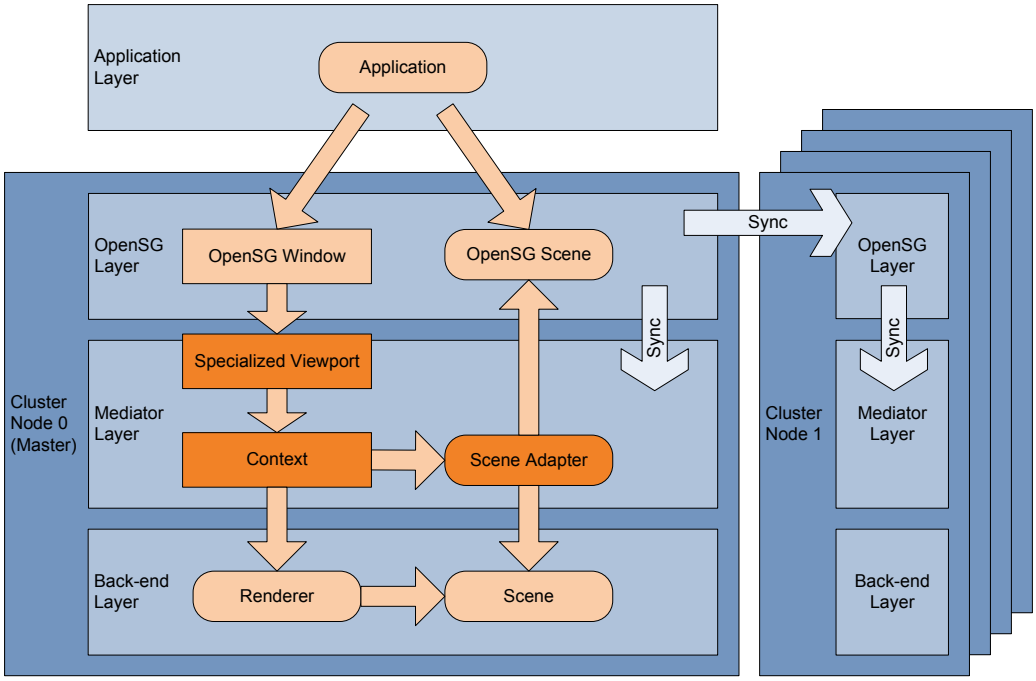

Figure 5.1: *Schematic overview of our architecture.*

internal representation and keeps it up-to-date. The context manages instances of the back-end and allows multiple viewports to share these instances.

Note that there is no direct dependency from the application/OpenSG layer into the mediator layer (only via the default OpenSG Viewport interface) and no dependency from the mediator into the application layer (only into the OpenSG layer, a mediator potentially works with all OpenSG applications). Also, the mediator depends on the back-end, but not the other way around. This takes care of our requirements of **extensibility and generality** and **non-intrusiveness**. Since the back-end is fed with its own scene representation and simply asked to fill a viewport, it usually can maintain near-optimal **rendering performance**. Relying on viewports also enables a simple (but usually sufficient) way to do **mixed (hybrid) rendering**. Viewports can be layered on top of each other in order to combine the images of different back-ends using z-buffering and alpha-blending. The **ability to extend the application layer** is provided by OpenSG's attachment mechanism [32], which allows to attach arbitrary data to nodes. The application layer only needs to pack data for extensions into attachments, which are then interpreted by

mediators that understand the extensions and ignored by others. How the system meets the requirements **fast incremental updates** and **clustering and stereo** is particularly interesting and described in Sections 5.3.3 and 5.3.4. For the remainder of this section, however, we focus on the viewport and scene adapter mechanisms.

5.3.2.1 The OpenSG Scene

The OpenSG scene in the scheme in Figure 5.1 is usually a stripped down scene graph with only a few node types. We intentionally did not strictly define which nodes a mediator has to understand, unknown nodes can simply be ignored by the scene adapter. Node types that are supported by all our renderers are Transform, Geometry, two types of Materials, Lights, Camera, and Background. This seems to be the minimal set a renderer needs to produce meaningful pictures.

Transforms are simple 4×4 matrix transforms with n children (i.e. they are also Group nodes). Geometry will usually be an indexed triangle mesh, but since OpenSG provides the TriangleIterator interface it is almost always possible to convert an arbitrary geometry into a triangle soup, which can be handled by most renderers. This includes parametric surface patches. (Of course, a back-end can also choose to interpret them directly, if they are supported.) As far as materials go, we take a pragmatic approach and provide two pre-defined declarative materials, a simple one based on the OpenGL 1.1 material model and a more complex one (CommonSurfaceShader, see Appendix 5.A) based on a recent X3D extension proposal [123]. The CommonSurfaceShader is quite powerful and supports features like perfect specular reflection/refraction and bump mapping. However, it cannot describe every appearance adequately, so a back-end can also interpret ShaderChunks that may contain explicit shader code in addition to the declarative material nodes. This mechanism offers greater flexibility, but will incur a loss of portability. Every mediator we implemented so far supports at least point, spot, and directional lights. The ray tracers also support area lights and lightprobes, see Section 5.4.1. Currently all back-ends use a simple pinhole camera, which can be easily translated into a view/projection matrix (for rasterization) or a view

frustum (for ray tracing). Furthermore, a simple mono-colored background is supported, as well as a skydome.

5.3.2.2 Viewport

Each mediator exposes a specialized OpenSG-Viewport which internally maps to the underlying renderer. So, every time OpenSG (on behalf of the application) wants a viewport to be rendered, the back-end is invoked to render its (sub-)scene. The target is usually the OpenGL back buffer, but it can also be another render target. We could even implement a viewport that renders an image to disk or streams a video to a website.

The back-ends are invoked exclusively through this specialized Viewport class. If the application wants to use a certain back-end, it just creates the corresponding viewport, assigns camera, root-node, and background to this viewport and attaches it to an OpenSG-Window. The viewport then creates the entire infrastructure necessary to convert the scene and instantiates the underlying renderer. Usually this is done lazily upon the first render-request, but other patterns are possible. The viewport also provides the interface to set parameters of the back-end that are not tied to geometry, materials, or other objects (e.g. antialiasing options or maximum ray depth).

Using a viewport in such a way also allows us to elegantly use (some may say: abuse) OpenSG's stereo and clustering capabilities. Stereo rendering is possible by simply using two viewports, one for each eye (layered on top of each other, side-by-side, or even on different machines). In order to prevent wasting resources in such a setup, the viewports usually share the underlying converted scenes and other resources via ref-counted contexts. Arbitrary clustering setups are described in Section 5.3.4. Since viewports can be deactivated and activated on-the-fly, layered viewports can also be used to quickly switch from one back-end to another. For example, one can use rasterization for navigation and then seamlessly switch to ray tracing once an interesting viewpoint has been reached. Also, common post processing effects can

be attached to the viewport (the anaglyph encoding in Fig. 5.9 is a simple example).

5.3.2.3 Scene Adapter

The scene adapter is responsible for mapping the OpenSG scene to a representation the back-end can use. This is usually where the bulk of the work has to be done when implementing a new mediator layer. In some cases, the renderer may be able to use the OpenSG scene directly, or at least parts of it (e.g. an OpenGL-based forward or deferred renderer), but often a conversion of the scene will be necessary. In this case, the adapter will usually traverse the whole OpenSG scene graph once in the first render call and build a shadow scene by converting objects such as geometries, materials, and lights into suitable representations. Note that this does not have to be (and usually is not) a one-to-one mapping, the case studies in Section 5.4 show examples. However, the handling of incremental updates (described in the following subsection) requires to quickly identify representatives that need updating as a result of a change. Therefore, usually several maps are build during the initial conversion, which serve as a scene dictionary.

5.3.3 Handling Changes

Probably the most obvious choice to handle incremental changes is to use callbacks that either the representatives themselves or some handlers register at the objects of the OpenSG scene. However, OpenSG has already a sophisticated system to handle incremental changes, which we can readily use (or, again: abuse).

OpenSG automatically keeps track of all changes that are made to the attributes of an object (the Fields of a FieldContainer) in so-called ChangeLists [140, 112]. A ChangleList's primary purpose is to allow synchronization between threads and over the network in OpenSG's multi-buffered threading model. A side effect of this mechanism is that at each render-call on the viewport, we have a ChangeList available that contains all changes made to

the scene in this frame. Syncing the OpenSG scene with the back-end scene is now almost trivial: In each call to render, the viewport invokes the scene adapter's sync-method, which parses through the current ChangeList and updates the representatives for all relevant changes (Fig. 5.5). ChangeLists also contain entries for newly created and deleted objects, so these events can be handled as well.

This approach has several advantages over registered listeners. First of all, it processes a changed field only once. If, for example, a transform is changed three times in one frame, only one update propagates to the back-end, namely the last. Second, the mechanism (in conjunction with OpenSG's Aspect concept) works well in multithreaded and clustered environments. The back-end can access a consistent OpenSG-state for the current frame, while the application already updates the OpenSG scene for the next frame. Similarly, it is absolutely irrelevant for the mediator whether the ChangeList came from the same machine or over the network. This way, the mediator is completely agnostic towards the clustering setup it runs in. Finally, we believe the ChangeList concept is just simpler and less error-prone. For a reasonably complex scene thousands of listeners may have to registered, and since there is not always a one-to-one mapping, things can get out-of-hand quickly. However, parsing the ChangeLists gives rise to one pathological case: if the ChangeList contains a myriad of changes but most of these changes are irrelevant to the back-end, the parsing may incur a performance penalty. But the number of irrelevant entries would have to be very large in practice, since ChangeLists can be parsed very efficiently. They are basically just a list of object ids and associated bitmasks that mark the changed fields.

5.3.4 Multithreading and Clustering

Multithreading and clustering have been mentioned before, and the mediator design allows us again to use large parts of what OpenSG has to offer here. The details and inner workings of OpenSG's multithreading and clustering concept are described by Voß et al. [140], we will only describe how our approach integrates into this framework.

In our system, the application layer exists only once on a single host, but the OpenSG scene may be mirrored on multiple machines. The copies of the scene are kept in sync by ChangeLists sent over the network. As already mentioned, the fact that the mediators only work on the OpenSG scene and ChangeLists allows them to function properly in cluster/multithreading setups without knowing anything about the application layer.

Rendering in a cluster is managed by the ClusterWindow class, which can be envisioned as a virtual window that exists on a remote host; or, in the case of tiled rendering and load balancing, multiple hosts. Since a mediator only interacts with OpenSG via instances of its specialized Viewport class, we just have to add the specialized viewports to a ClusterWindow like ordinary OpenSG Viewports to use them in any OpenSG-compatible cluster setup. Cameras and viewports are automatically adapted by OpenSG and each machine renders with its own instances of specialized renderers without them even knowing that they are part of a clustering setup and cooperating. Supporting classic setups such as tiled rendering for large display walls and cooperative rendering with load-balancing is important, and our design does this quite well. However, the possibility to selectively use different back-ends on different machines opens up interesting new possibilities. For example, we could use a fast, low quality GPU-based renderer on a touch-table or mobile device to navigate through a scene, while a powerful cluster renders the same scene on a large tiled display wall using load-balancing with a progressive, photorealistic algorithm such as path tracing.

5.4 Case Studies

In this section, we discuss two case studies: an Optix-based ray tracing back-end and a visibility guided renderer for very large data sets. We also describe a concrete example application.

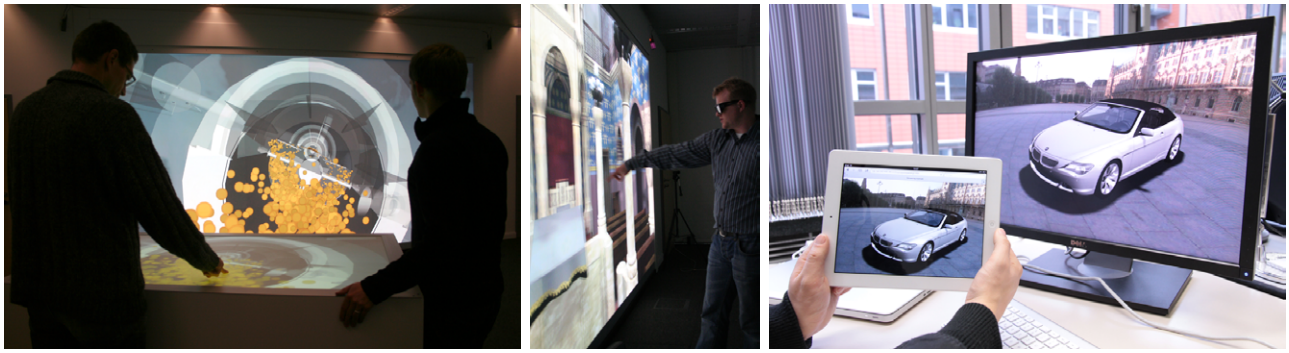

Figure 5.2: *Some use cases. Left: an application running on a touch table and a large tiled display wall (6 × 4 tiles, 8640 × 4200 pixels). Middle: stereoscopic rendering on the same display wall. Right: video stream from desktop to web page, viewed in a browser on a tablet.*

5.4.1 Optix Back-end

Optix [99] is a ray tracing engine based on CUDA [91]. Figure 5.3 shows the static structure of our Optix mediator. One interesting aspect is that the same mediator supports three renderers: a simple Whitted-style real-time ray tracer, an interactive progressive path tracer, and a special coverage renderer (described in Section 5.4.3). The renderers differ mainly in some CUDA programs (e.g. camera and material programs), while most other parts (e.g. intersection programs, most of the scene adapters) are identical. Therefore, we implemented the key components of the mediator layer, OptixScene and OptixViewport, as "fat" base classes that contain most of the functionality and specialized them where needed.

The most interesting component of most mediators is the scene adapter, this is also the case here (OptixScene). One important point is that the scene adapter does not really mirror the OpenSG scene as an Optix scene. The OpenSG scene graph is usually quite deep (as it mirrors the X3D/VRML-graph of the application layer), and the Optix scene adapter collapses it into a flat graph that has at most one transform above each geometry instance (Fig. 5.4). A mapping from the original transforms to the collapsed ones is established, so we can quickly find the representatives that need updating for a given ChangeList entry (Fig. 5.5).

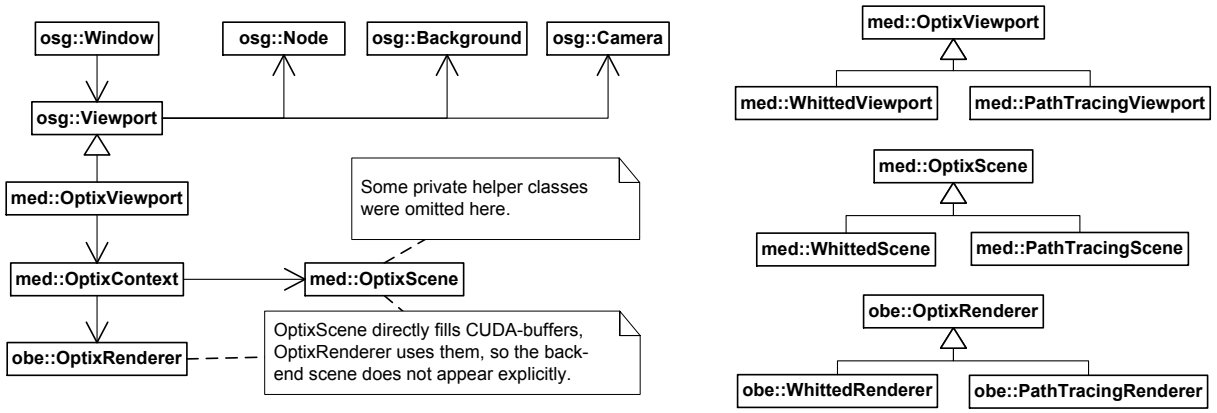

Figure 5.3: *Static structure of the Optix mediator. The namespace-tags refer to the layers in Fig. 5.1: osg = OpenSG layer, med = mediator layer, obe = Optix back-end. Some small helper classes and the coverage renderer (Sec. 5.4.3) were omitted.*

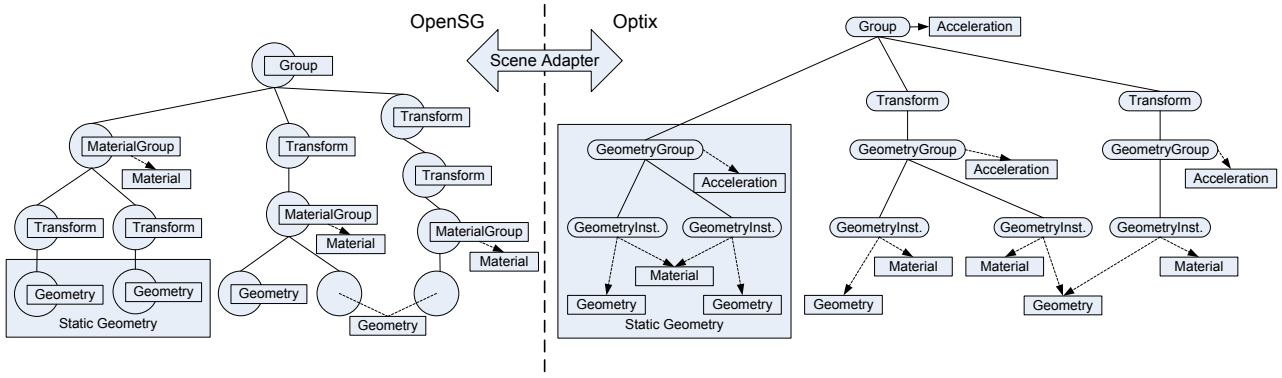

Figure 5.4: *The scene adapter maps the OpenSG scene to an Optix scene. In order to optimize for ray tracing, this is not a one-to-one mapping: GeometryGroups that provide Acceleration structures are introduced, Transforms are collapsed, static geometry is placed under one common Acceleration. The figure also illustrates sharing of Materials and Geometries (instancing).*

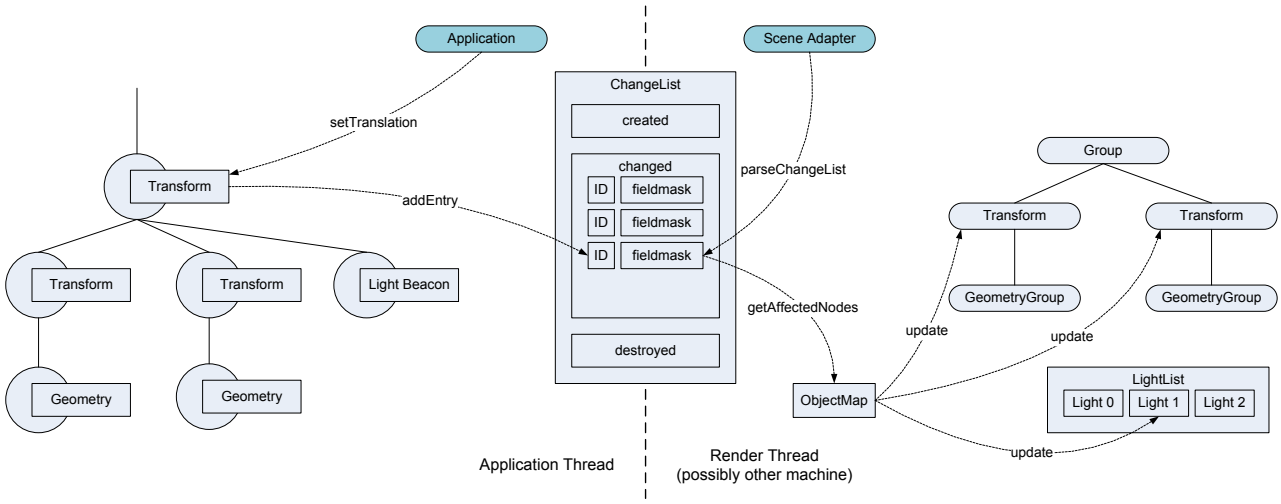

Figure 5.5: *Snooping on ChangeLists is our way to propagate incremental changes. When the application changes a Field, a corresponding entry is created (or updated) in the current ChangeList. When the mediator syncs the back-end (usually before rendering), it parses through the ChangeList and carries out the necessary updates for each entry in the back-end scene. Because of OpenSG's Aspect mechanism, this sync is thread-safe and works between cluster nodes.*

Geometry instances below the same transform node are assumed not to move independently and are combined into the same acceleration structure. Geometry that has been explicitly flagged as static is also combined into one large static geometry chunk. Geometry can also be flagged as animated, in which case a special acceleration structure is used that can be quickly updated but has slightly lower rendering performance.

These optimizations are necessary to retain good ray tracing performance while still allowing fast updates for frequently occurring changes (changing transforms, lights, animated meshes, etc.). Changes to the graph structure become more expensive, though, but we assume these to occur relatively seldom. As the build times for the acceleration structures can be quite high, they can be cached on disk, so we do not have to pay the costs every time a scene is loaded. Geometry data is directly written into CUDA-buffers, as are image textures, in order to prevent duplication of large data. (Textures can also be shared with an OpenGL context as texture objects.)

Area lights are another interesting aspect of this mediator. They are a simple example of how a mediator can freely interpret the scene in order to play out

Figure 5.6: *The same scene rendered with three different renderers by switching through viewports. Left to right: rasterization, whitted-style ray tracing, path tracing.*

Figure 5.7: *Participating media specified by CommonVolumeShaders.*

the strengths of its back-end without exposing extensions in the application layer. Instead of exposing a specialized node in the application layer, area lights are simply geometries with a non-zero emission component in their material. The mediator detects these geometries and converts them into area lights for the ray tracers. This way, the mediator can easily support area lights of arbitrary shape (an example is shown in Figure 5.9). For back-ends that do not support them, they remain emissive objects but do not shine light onto other geometry.

All ray tracing back-ends support the full feature set of CommonSurface-Shader [123, 124], an effort by the InstantReality team to provide X3D with a modern and portable material description. Figure 5.6 shows the same materials with three different rendering back-ends. Since the CommonSurface-Shader material has an "emission texture" slot, it allows authors to specify textured area lights.

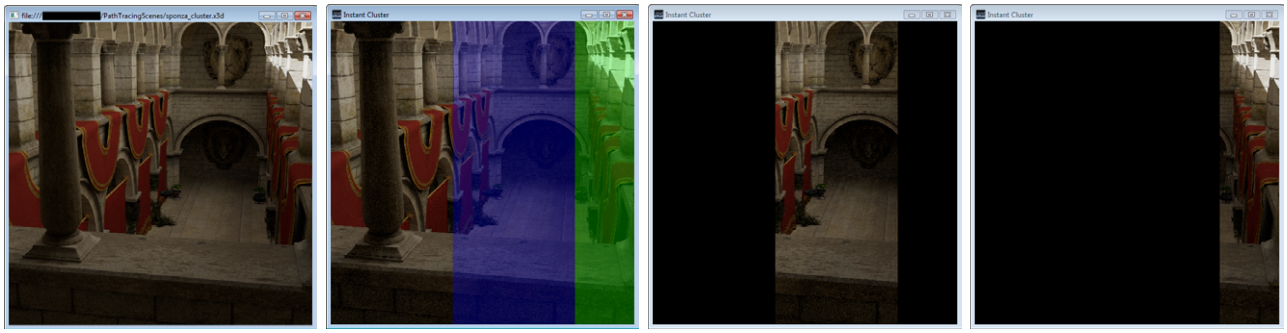

Figure 5.8: *Cooperative path tracing with load balancing (localhost with two supporting machines). The three images to the right show the workload of all servers.*

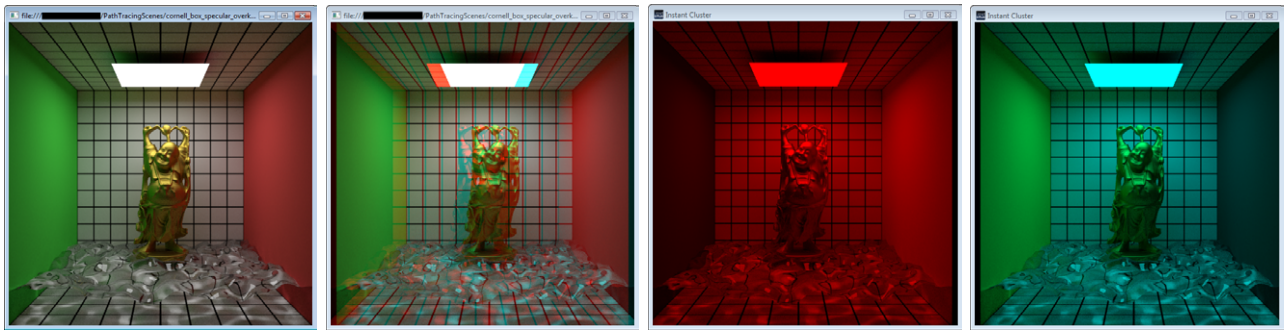

Figure 5.9: *Path tracing in a stereo setup (anaglyph with one client and two servers, one for each eye).*

The ray tracing back-ends also support the CommonVolumeShader [120], a realistic material description for volumetric effects such as fog or plasma. Figure 5.7 shows a simple single-scattering fog. This is an example of a shading feature that is exposed via an extension of the application layer (in this case a specialized ShaderChunk).

CommonSurfaceShader and CommonVolumeShader are reviewed in Appendices 5.A and 5.B.

Figures 5.8 and 5.9 show the Optix path tracing back-end in cluster setups (cooperative and stereo). It is important to note that the back-end is not even aware it is running in a cluster or stereo setup, it just fills its viewport.

5.4.2 VGR Back-end

We have successfully integrated 3D Interactive's Interviews3D platform [19] into our system, a visibility guided renderer (VGR) for large datasets. Besides the default renderer, the VGR mediator also supports a coverage renderer (Sec. 5.4.3).

Optix is a relatively low-level API (often advertised as OpenGL for ray tracing) and constitutes a framework on top of which users have to implement their own rendering algorithms. In contrast to Optix, VGR is a closed package that gives users much less freedom. VGR has a very strictly defined scene authoring interface centered on a scene database. This allows the library to play out its strengths when it comes to out-of-core rendering of huge data. The scene adapter for this mediator converts the OpenSG scene into such a database. Here we had to compromise and provide two options. The first option converts the OpenSG scene and keeps it intact and in memory. This is the default. This option is best for interactive use, since the application layer is kept intact and everything that relies on the OpenSG scene still works (navigation, picking, animation). It also works automatically in clustered environments. The downside is that it hampers VGR's out-of-core abilities, because the whole OpenSG scene is kept in memory. Therefore, we also provide the option to use a (possibly pre-converted) database that does not need the OpenSG scene in memory. This is most useful for visualizing very large static scenes without much interaction (the ChangeList mechanism will only work on elements that are present in the OpenSG layer). For some applications, however, this may not be a problem. An example is the web-based visualization application described in the following section, where all the interaction takes place on the client-side. If a pre-converted database is to be used in a cluster setup, it currently has to be manually distributed to all cluster nodes that need it.

Another problem we encountered is the fact that VGR relies even more on static geometry than the ray tracers. In general, updates to geometry are very slow, even changes to transforms. But geometry can be moved to special "transform groups" that allow relatively efficient updates, as long as there are few transform groups. We use the static flag again to classify geometry

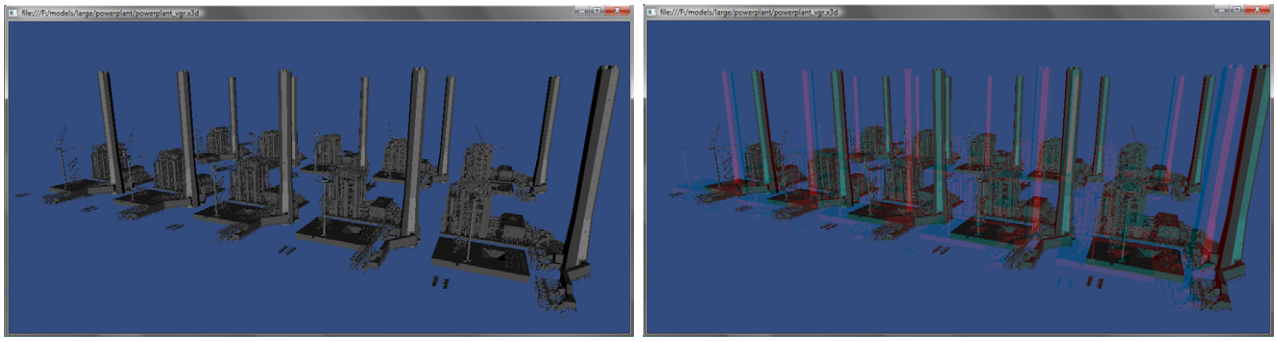

Figure 5.10: *10 powerplant models as anaglyph rendered with our VGR back-end (not instanced but fully replicated, ca. 140 million triangles, left and right eye rendered on separate machines).*

as strictly static. Each geometry flagged as animated is moved into its own transform group until all available groups are used. Geometry not flagged in any special way is treated as static, but is moved into a transform group as soon as an update occurs. This way, we may have to live with a small stutter on the first change, but subsequent changes are relatively painless, unless the geometry is pushed out of its transform group slot by another geometry.

With VGR mixed rendering became very important, because the VGR renderer is quite limited when it comes to (3D-)GUI-elements. A lot of our applications need to display huge models efficiently, but also need the possibility to combine them with dynamic annotations and markers. Here, the viewport interface of the mediator and the thoughtful design of the VGR system help a lot. Even though the system is closed (it even creates its own OpenGL context internally), VGR's output is basically a rendered frame buffer (color+z-buffer), which can be combined with other frame buffers (rendered by other viewports with other back-ends).

Figure 5.10 shows the VGR back-end in the same anaglyph setup as Figure 5.9. Again, the back-end knows nothing about stereo or the clustering setup. Figure 5.11 shows a Boeing 777 CAD model rendered in real-time with the VGR back-end.

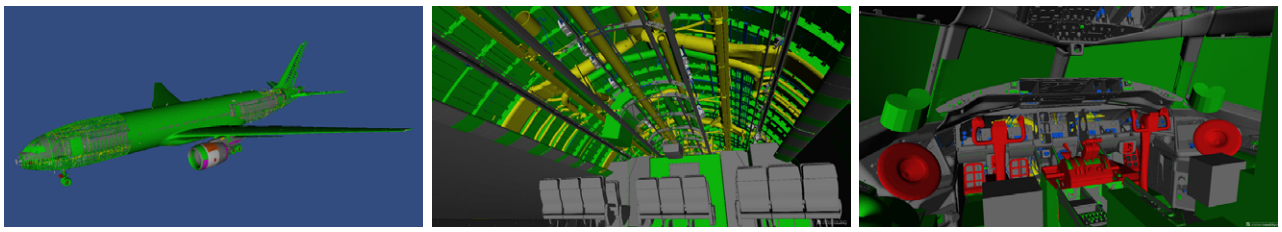

Figure 5.11: *Boeing 777 CAD model rendered with VGR back-end. The model consists of over 300 million polygons and is rendered at 30 Hz on a GeForce GTX 470.*

5.4.3 Application: Large CAD-Models in Web Browser

In this subsection, we describe a concrete application built on our system architecture and the mediators explained above. It is a distributed visualization application for large models.

The front-end is simply a WebGL-enabled web browser, rendering a HTML5-page with X3DOM [10]. However, current web-technology is not capable of handling large models efficiently. ("Large models" here means large for web applications, in the order of tens or hundreds of millions of polygons.) Therefore, we use a novel out-of-core approach to minimize the workload in the browser. The key idea is to use an asynchronous, remote culling service. Figure 5.12 shows the basic data flow. The browser (actually the X3DOM runtime) sends its current view frustum to the culling service, which determines the objects with the largest screen coverage and sends back a list of IDs for these objects. The browser then only fetches these "most important" objects from the asset server. This keeps memory consumption and rendering time manageable on weak devices, which would otherwise not be able to render such complex models. On the other hand, the approach consumes less bandwidth between culling-service and browser than pure server-side rendering with video streaming. This allows us to maintain high quality and interactivity even in weaker networks, where streaming does not work well.

The culling service is an InstantReality instance running a special rendering back-end (CoverageRenderer). This back-end does not render a traditional

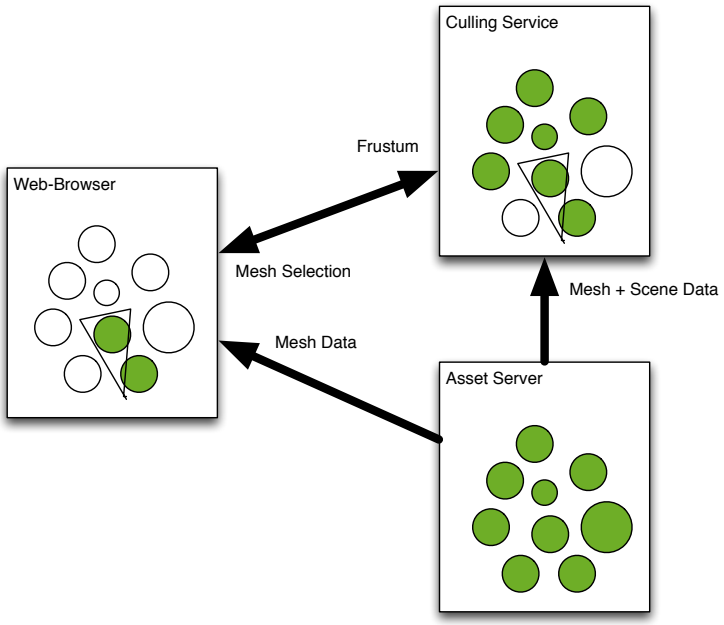

Figure 5.12: *Schematic data flow between browser, culling server, and asset server.*

image, but calculates which objects have which coverage in the final rendering (including occlusion). From this information the sorted list of object-IDs is generated, which allows the browser to prioritize important objects. We have implemented the culling service as an Optix-based back-end (as a ray tracer) and as an VGR-based back-end (as a rasterizer). Both cases use a minimalistic scene adapter that basically only converts geometry and establishes a mapping of IDs to objects. The geometry conversion is shared with the other renderers in the Optix/VGR mediator. Material information (apart from transparency) is not necessary.

Figure 5.13 shows both implementations, the Optix-based CoverageRenderer and the one based on VGR. In both cases the navigation is smooth in the browser.[1] We believe this is a show case that demonstrates nicely how the freedom obtained by our approach for flexible rendering back-ends can be used to build innovative distributed applications.

[1] `http://www.youtube.com/watch?v=zIHV3yC3IYo`
`http://www.youtube.com/watch?v=h0SUWqJfQsE`

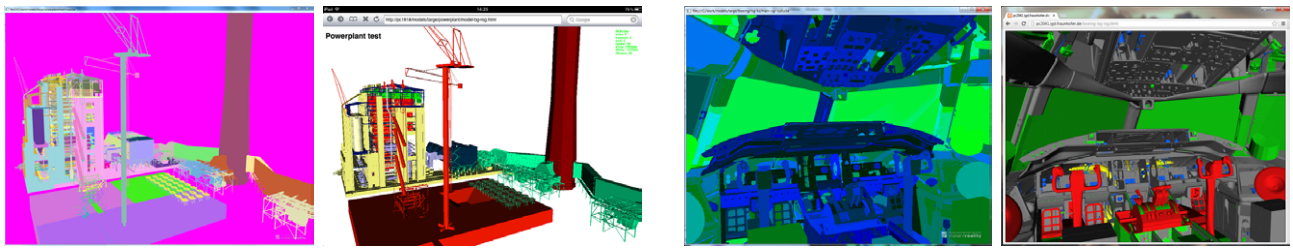

Figure 5.13: *The browser application with Optix back-end (left) and VGR back-end (right). The culling service identifies large objects (in terms of screen coverage). The web application only loads and renders the most important objects. The full Powerplant model has 14 million triangles, the web-application only renders 1.8. Note that the left image is only a debug visualization, the culling service does not have to generate an image. The full Boeing has over 300 million polygons, the web application only renders 4.2 million.*

5.5 Discussion

Apart from the points discussed in the case studies, we have made the following observations:

Scene adapter. Converting a stripped-down OpenSG scene into the back-ends preferred representation works very well in general. The mediator design allows us to easily extend OpenSG's well-designed and practice-proven support for incremental updates, multithreading, and clustering to back-ends that never were designed to work with OpenSG or in cluster setups. The fact that a scene adapter can (and usually will) change the structure of the scene graph can make it hard to track which changes imply updates to which representatives, but that is the price one has to pay if one wants to feed the back-end with an optimized representation.

Conversion speed. Another issue related to the scene adapter is that the conversion of the scene can be slow if complex operations are necessary (e.g. changing a texture format or converting surface patches into triangles). Performing parts of the conversion only once and caching the result can alleviate this problem. The cached acceleration structures and pre-converted databases described in Section 5.4 are examples of this approach.

Viewport interface. The fact that a rendering back-end only shows a specialized viewport to OpenSG and the application layer is a mixed blessing. On the one hand, it is a very slim interface that allows us to plug in the back-ends at the most important places. On the other hand, it can be limiting for advanced use cases, because it fails to separate three concerns: *what* to render (scene, camera), *how* to render it (the back-end), and *where* to render it to (render target). For example, a viewport that streams to a website cannot be freely combined with each back-end, but would have to be implemented multiple times. Of course, there can still be code-reuse, but a design with clear separation of concerns, as sketched in Section 5.6, would be preferable.

Memory consumption. Building the mediators on OpenSG scenes seems like a waste of memory at first sight. In the worst case, the scene can be represented three times: in the application layer, the OpenSG layer, and in the mediator or back-end. While this can be a problem sometimes (e.g. in the VGR-case), most of the time memory consumption is not excessive and acceptable. The reason for this is that the scene adapter usually does not duplicate large data (e.g. vertex buffers and images) in main-memory, but translates them directly into the back-ends representation (e.g. CUDA-buffers and OpenGL textures) – an operation that has to occur anyway. Also, the application layer can usually directly use OpenSG data structures, which removes the duplication between application layer and OpenSG layer. This leaves only the OpenSG scene as the central scene representation. Even this copy can be eliminated by feeding the mediator directly from the application layer, an option we provide for the VGR back-end. But this should be the exception, because it circumvents our original design and loses two of its strong points: interactive, thread-safe updates and clustering support provided by OpenSG's ChangeList mechanism.

Another concern with regard to memory consumption are the maps that link objects in the OpenSG scene to their representatives in the back-end scene. In our tests, these never grew beyond a few kilobytes, even for large scenes, and remained negligible compared to geometry and texture data.

5.6 Outlook

There is a project underway to build a general framework for visual computing [105], which will be partly based on an extended version of the approach described in this chapter. The most important extensions are:

General clustering. The new system will be based on OpenSG 2.0 (our current implementation uses 1.8). In the future, we want to use OpenSG more as general data management layer, not only as a scene graph. The goal is to be able to build more general clustered applications. Currently, the whole scene graph (and a few associated things like viewports) is simply mirrored on each cluster node in a client-server cluster [130]. Moving away from the rendering-centric scene graph and ClusterWindow concepts would allow a directed distribution of arbitrary data in a cluster with more specialized cluster nodes while keeping the benefits of OpenSG's sophisticated synchronization mechanism.

Not only rendering. OpenSG as a general data management layer would also make it easier to extend our mediator approach to semantics other than rendering. For example an application scene (interaction), a physics scene (simulation), and a graphics scene (rendering) could coexist and could be kept in sync almost automatically. And these components could even be moved on different cluster nodes without major changes to the application.

Decouple viewport from back-end. To gain more flexibility, we plan to remove the tight coupling of a mediator with its specialized viewport. We want to use OpenSG 2.0's Stage concept [141] to plug in mediator layers (at least for rendering). There will be only one specialized viewport to which different stages (i.e. different back-ends) can be attached. The viewport defines *what* is to be rendered, the stage *how* it should be rendered.

5.7 Conclusions

We have described a pragmatic, practice-proven approach to using exchangeable rendering back-ends with a common application layer in heterogeneous computing environments. The approach is based on a mediator layer that can be plugged into the OpenSG infrastructure. The design allows the mediator to easily use OpenSG's multithreading and clustering capabilities while retaining the strengths of the specialized back-ends. It also allows the mediator to sync incremental changes very elegantly and efficiently. The approach is very flexible and supports a wide range of renderers. The mediator layer has to comply with only two basic terms: it has to expose the back-end's functionality through a specialized viewport and it has to be able to understand a basic OpenSG scene. Other than that, mediators are free in their decisions what to support, how to map scene elements, and – most importantly – what to ignore. We have demonstrated results and problems with two case studies and a concrete application.

A weakness of the approach is the high memory consumption in some cases. Another issue we want to address with future work is support for a more general (less rendering-centric) clustering approach.

Appendix 5.A CommonSurfaceShader

In this appendix, we briefly sketch the CommonSurfaceShader node, our proposal for a portable, physically-based material description for X3D. This is a condensed version of two Web3D papers [123, 124].

5.A.1 Introduction

CommonSurfaceShader is a declarative surface shader for the X3D standard that allows for a compact, expressive, and implementation-independent specification of surface appearance for physically-based rendering. X3D's Material

node is portable, but its feature set has become inadequate over the last years. Explicit shader programs, on the other hand, offer the expressive power to specify advanced shading techniques, but are highly implementation-dependent. The motivation for our proposal is to bridge the gap between these two worlds: to provide X3D with renderer-independent support for modern materials and to increase interoperability with digital content creation tools.

At the core of our proposal is the CommonSurfaceShader node. This node provides no explicit shader code, only a slim declarative interface consisting of a set of parameters with clearly defined semantics. Implementation details are completely hidden and portability is maximized. It supports diffuse and glossy surface reflection, bump mapping, and perfect specular reflection and refraction. This feature set can capture the appearance of many common materials accurately and is easily mappable to the material descriptions of other software packages and file formats.

Since the original publication in 2010 [123], the CommonSurfaceShader node has been in use in InstantReality [43], where it serves as an up-to-date supplement for the Material node. In addition, it is used in the WebGL-based X3DOM framework [9, 10]. A third implementation for the generic scene graph library OpenSG [106] is currently in the works.

5.A.2 SurfaceShader Node

The goal of our proposal is to bridge the gap between the portability of the Material node and the expressiveness that the *Programmable Shaders* component offers. We do this by introducing a new class of *declarative* shader nodes that do *not* use explicit and imperative shader programs, but a declarative interface consisting of fields with defined semantics. An implementation is then free to implement these semantics in an appropriate way. In order to group all such declarative shader nodes in X3D's inheritance hierarchy, we propose the SurfaceShader node as a common base.

```
SurfaceShader : X3DShaderNode {
  [...]
}
```

5.A.3 CommonSurfaceShader Node

The fields of the CommonSurfaceShader node can be grouped into three logical components, each of which is designed to capture a different aspect of surface appearance. The core component captures diffuse and glossy reflection, the bump component can perturb shading normals to produce a richer appearance, and the perfect specular component models perfect specular reflection and refraction. Below is an overview of the fields of the CommonSurfaceShader node.

```
CommonSurfaceShader : SurfaceShader {
  [...]

  # core component
  SFNode    [in,out] emissionTexture              NULL
  SFNode    [in,out] ambientTexture               NULL
  SFNode    [in,out] diffuseTexture               NULL
  SFNode    [in,out] specularTexture              NULL
  SFNode    [in,out] shininessTexture             NULL
  SFNode    [in,out] alphaTexture                 NULL
  SFVec3f   [in,out] emissionFactor         0.0 0.0 0.0
  SFVec3f   [in,out] ambientFactor          0.2 0.2 0.2
  SFVec3f   [in,out] diffuseFactor          0.8 0.8 0.8
  SFVec3f   [in,out] specularFactor         0.0 0.0 0.0
  SFVec2f   [in,out] shininessFactor            0.0 0.0
  SFFloat   [in,out] alphaFactor                    1
  SFBool    [in,out] invertAlphaTexture         FALSE

  # bump mapping component
  SFNode    [in,out] normalTexture                NULL
  SFString  [in,out] normalFormat               "UNORM"
  SFString  [in,out] normalSpace              "TANGENT"
  SFVec3f   [in,out] normalScale                  2 2 2
  SFVec3f   [in,out] normalBias                -1 -1 -1
  SFNode    [in,out] heightTexture                NULL
  SFFloat   [in,out] heightScale                     1
```

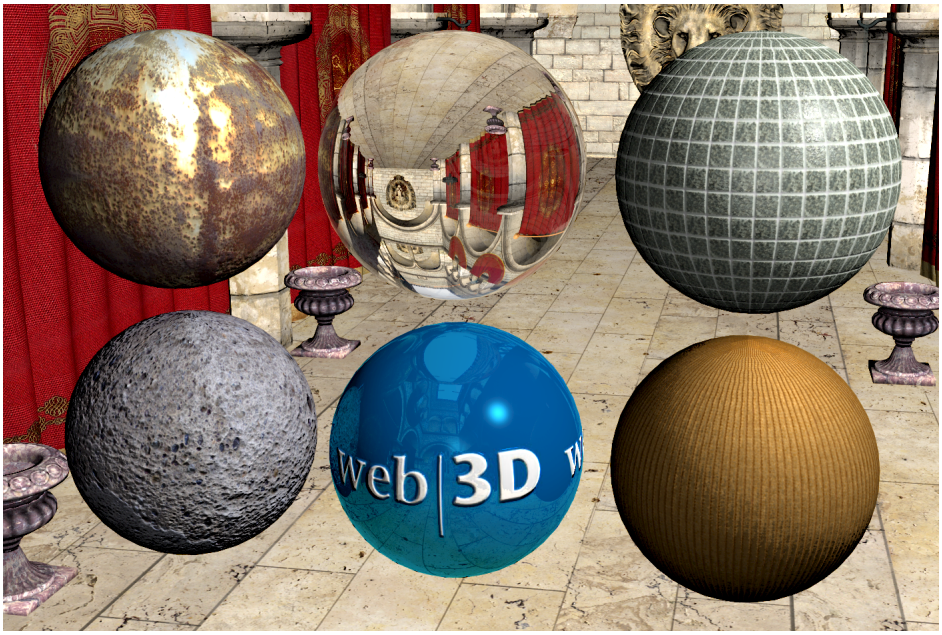

Figure 5.14: *Some examples of CommonSurfaceShaders.*

```
  # perfect specular component
  SFNode    [in,out] reflectionTexture              NULL
  SFNode    [in,out] transmissionTexture            NULL
  SFVec3f   [in,out] reflectionFactor             0 0 0
  SFVec3f   [in,out] transmissionFactor           0 0 0
  SFVec3f   [in,out] relativeIndexOfRefraction    1 1 1
  SFFloat   [in,out] fresnelBlend                   1.0

  # extensions
  SFInt32   [in,out] tangentTextureCoordinatesId     -1
  SFInt32   [in,out] binormalTextureCoordinatesId    -1
  SFNode    [in,out] environmentTexture             NULL
  SFVec3f   [in,out] environmentFactor            0 0 0
}
```

Figure 5.14 shows some examples of CommonSurfaceShaders and illustrates that a wide range of materials can be captured by combinations of the core, bump, and specular components. Figure 5.15 shows a photorealistic rendering of a car. All materials in these two figures were specified with CommonSurfaceShader. For a more thorough evaluation and more details on the nodes, see the original proposals [123, 124].

Figure 5.15: *Path traced image of a car under environment map lighting. All materials were specified using CommonSurfaceShader.*

Appendix 5.B CommonVolumeShader

This appendix is a condensed version of the CommonVolumeShader proposal [120], which was presented at Web3D 2011.

5.B.1 Introduction

Rendering volumetric phenomena with believable appearance can add tremendous realism to virtual scenes. The CommonVolumeShader node is an extension of the X3D standard that has been specifically designed for physically-based rendering of participating media. It is inspired by the CommonSurfaceShader node and provides a declarative (i.e. highly portable) description of volumetric optical properties for physically-based rendering. Surprisingly few parameters are needed to accurately describe many volumetric phenomena, resulting in a compact but still widely applicable node. As part of an Appearance node, a CommonVolumeShader can be attached to arbitrary geometry and can be used in combination with other Appearance properties, for example a CommonSurfaceShader node that describes surface appearance.

5.B.2 VolumeShader Node

Analogously to the CommonSurfaceShader, we have decided to group all volume shaders under a common base node, the VolumeShader node. This design allows us to combine a VolumeShader with a SurfaceShader in a single Appearance instance. So, an implementation may shade the surface of an object with a CommonSurfaceShader and the interior with a CommonVolumeShader, or it may ignore one (or even both) if it does not support the nodes.

```
VolumeShader : X3DShaderNode {
  [...]
}
```

Refraction and reflection at boundaries are the responsibility of the corresponding SurfaceShader nodes and are independent of the VolumeShader.

5.B.3 CommonVolumeShader Node

The CommonVolumeShader contains the following fields:

```
CommonVolumeShader : VolumeShader {
  [...]
  SFVec3f    [in,out] emissionFactor        0.0 0.0 0.0
  SFNode     [in,out] emissionTexture              NULL
  SFVec3f    [in,out] absorptionFactor      0.0 0.0 0.0
  SFNode     [in,out] absorptionTexture            NULL
  SFVec3f    [in,out] scatteringFactor      0.0 0.0 0.0
  SFNode     [in,out] scatteringTexture            NULL

  SFNode     [in,out] phaseFunction                NULL
}
```

The semantics of most parameters should be intuitively clear. The phaseFunction field specifies the phase function to be used. If it is NULL, the isotropic phase function should be used.

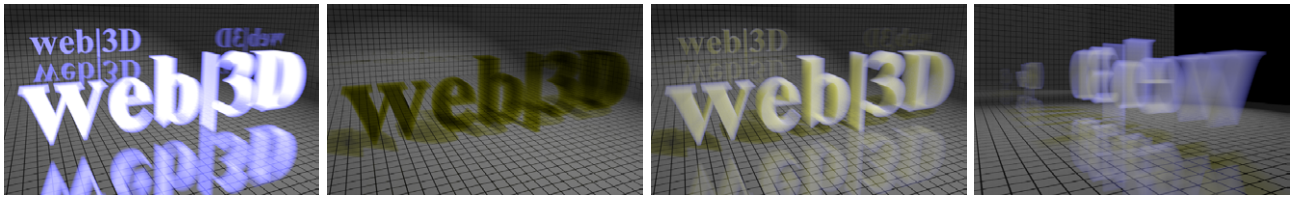

Figure 5.16: *A CommonVolumeShader node can be used to specify volumetric light transport for backends that support it (here the Whitted-style ray tracer). From left to right: emission only, absorption only, (single) scattering only, reduced scattering viewed from behind.*

Figure 5.17: *Strongly scattering media in combination with a comparatively strong surface reflectance result in a characteristic look that – without CommonVolumeShader – can only be achieved by explicit (and unportable) subsurface scattering shaders in X3D.*

Figure 5.16 shows the principal components of CommonVolumeShader: emission, absorption, and scattering. Figure 5.17 shows some examples where a CommonVolumeShader was combined with a CommonSurfaceShader to simulate subsurface scattering. For more details on the nodes and a thorough evaluation see the original proposal [120].

6 Conclusions and Future Work

This chapter concludes the dissertation by summarizing contributions and findings. It also gives some directions for future work.

In this dissertation, we have addressed two problems in the emerging field of interactive stochastic ray tracing. The first problem is to provide interactive, low-noise previews of high-quality global illumination renderings. To this end, we have developed two filtering techniques to reduce noise in interactive progressive rendering. The second problem is to extent interactive systems with specialized rendering back-ends, including (but not limited to) interactive ray tracers. For that purpose, we have presented a system architecture that enables the use of flexibly exchangeable rendering back-ends under a common application layer with focus on distributed and multithreaded systems.

In the following section, we summarize the results in more detail. After that, we briefly discuss some directions for future work.

6.1 Summary

6.1.1 Filtering Pixel Radiance

In Chapter 3, we have presented a noise reduction method based on filtering image pixels. The basic idea is to reduce the variance to a user-defined threshold by biasing the estimator. The two key contributions are an

adapted version of the cross bilateral filter and a per-pixel blending operator.

Variants of the cross bilateral filter have been used in rendering before, but the specific range buffer used in this dissertation is novel. We have shown that the resulting filter is more robust than similar approaches in difficult scenes with perfect specular surfaces and high-frequency textures.

Rather than trying to completely remove noise, our primary goal was to attenuate noise to a user-defined level for reliable previews. This is where the perceptually-based blending operator comes into play. It can leave a controlled amount of high-frequency noise in the image, which is often preferable to an overfiltered image. The blending operator also guarantees the consistency of the algorithm in a progressive setup. Overall, the method achieves a strong noise reduction with very little blurring. We have demonstrated that with various test scenes.

We have also presented some optimizations for the original method. The first notable contribution here is the antialiasing recovery step, which improves the performance on antialiased edges. A second contribution is the heuristic for adaptive bandwidth selection of the bilateral filter, which can improve runtime and reduce blurring. Finally, we have described a simple method to attenuate spike noise, which can remove objectionable "fireflies" from the rendered images.

However, some limitations remain. Blurring across unrecognized edges can still be a problem, and for optimal results too many parameters have to be tweaked manually.

6.1.2 Filtering Incident Radiance

An inherent weakness of the bilateral filter is that some blurring across details in texture and geometry is inevitable, even with high-quality approaches such as the one presented in Chapter 3. This led to the development of radiance filtering, which we have presented in Chapter 4.

Instead of filtering pixel values or irradiance, radiance filtering filters the incident radiance at surface points. We have shown that this approach significantly reduces noise without blurring details in geometry or textures. It is also more general than irradiance filtering in that it can be applied to non-diffuse surfaces. In addition, radiance filtering handles antialiased edges and distribution effects such as depth of field better than image filtering. In a theoretical analysis, we have shown that the algorithm is consistent and derived asymptotic boundaries for bias and variance.

Pure radiance filtering has lower efficiency on reflecting/refracting surfaces. We have shown how to alleviate this with a hybrid filter that combines techniques from Chapters 3 and 4. As an additional optimization, the blending operator from Chapter 3 can be adapted to radiance filtering.

The biggest limitation of radiance filtering is that it only works well when applied to smooth illumination, since it essentially blurs the radiance signal. We have shown how to mitigate this problem with bilateral kernels, but it remains an issue. Especially in the early stages of rendering, when the kernels are large. The intended target for radiance filtering is indirect illumination, which will usually be smooth enough. However, if there are sharp features in the indirect illumination, such as caustics, they will be blurred.

At this point, it makes sense to compare the two filtering approaches presented here against each other. In short, pixel filtering (Ch. 3) is faster and performs better on reflecting/refracting surfaces. Radiance filtering (Ch. 4) is more robust and produces higher quality on all other surfaces. We think pixel filtering has more potential toward real-time rendering, but for high-quality previews at interactive rates we believe sample-based approaches such as radiance filtering are better suited.

6.1.3 System Architecture

In Chapter 5, we have described a novel approach to using multiple rendering back-ends with the same application layer in distributed systems. The

primary intent was to extend OpenSG, but the ideas should be applicable to similar systems.

Two principles guided the design. First, each back-end should be allowed to manage its own specialized scene representation, to ensure near-optimal rendering performance. Second, the design should work in a distributed, multithreaded system, without burdening the application layer or the back-ends with details of the cluster environment.

Key to the approach is a mediator layer that can be plugged into the OpenSG infrastructure. We have demonstrated results and problems with two case studies and an example application. In particular, we have shown how incremental changes can be handled efficiently by a scene adapter that snoops on ChangeLists, even in multithreaded cluster environments. Another strength of the approach is that it can support a wide range of renderers, owing mostly to the to the generic scene/viewport interface. The downside of this generic design is that the scene adapter can become bloated if many different nodes and events have to be handled with a complex scene-to-scene mapping. Memory consumption can be an issue, too.

6.2 Future Work

The focus of this dissertation is on low-noise previews and our primary objective was to reduce variance in the estimators. However, as stated in Section 2.3.4 (Eq. 2.24), the expected error is composed of variance *and* bias. Incorporating an estimate of bias into the algorithms would allow users to impose a limit on the expected overall error, not just on the variance. This could improve the performance with features that currently suffer from high bias (e.g. caustics). On the other hand, it is important to notice that bias is not completely ignored in this dissertation. The test cases clearly show a reduced overall error, which indicates that (in general) the variance reduction is not eaten up by bias. We also have discussed the types of bias that occur with our algorithms and their manifestations in the image. For the radiance filtering algorithm, we have even derived an asymptotic bound.

Another direction for future work is to adapt the filters presented here for real-time rendering. Astonishing progress has been made in this field since we started working on this dissertation and it would be interesting to see if some of the concepts presented here can be applied.

Filtering approaches are typically only applied to indirect (or smooth direct) illumination. Of course, direct illumination can suffer from high variance as well. A few specialized approaches exist for direct illumination, but it would be interesting to evaluate the potential of more general filtering methods for direct illumination.

For the software engineering part of this dissertation (Ch. 5), the most pressing matter is to remove the restriction to client-server clusters. In fact, the first steps toward this are currently undertaken in the VCoRE project [105].

Bibliography

[1] Adams, A., Gelfand, N., Dolson, J., and Levoy, M. Gaussian kd-trees for fast high-dimensional filtering. *ACM Trans. Graph. 28* (July 2009), 21:1–21:12.

[2] Arcila, T., Allard, J., Ménier, C., Boyer, E., and Raffin, B. FlowVR: A framework for distributed virtual reality applications. In *Journees de l'AFRV* (Rocquencourt, France, November 2006).

[3] Arvo, J. Backward ray tracing. In *In ACM SIGGRAPH '86 Course Notes – Developments in Ray Tracing* (1986), pp. 259–263.

[4] Aurich, V., and Weule, J. Non-linear gaussian filters performing edge preserving diffusion. In *Mustererkennung 1995, 17. DAGM-Symposium* (London, UK, 1995), Springer-Verlag, pp. 538–545.

[5] Banterle, F., Corsini, M., Cignoni, P., and Scopigno, R. A low-memory, straightforward and fast bilateral filter through subsampling in spatial domain. *Computer Graphics Forum 31*, 1 (2012), 19–32.

[6] Barash, D. A fundamental relationship between bilateral filtering, adaptive smoothing, and the nonlinear diffusion equation. *IEEE Trans. Pattern Anal. Mach. Intell. 24* (June 2002), 844–847.

[7] Bauszat, P., Eisemann, M., and Magnor, M. Guided image filtering for interactive high-quality global illumination. *Computer Graphics Forum (Proceedings. of EGSR 2011) 30*, 4 (2011), 1361–1368.

[8] Behr, J., Dähne, P., and Roth, M. Utilizing X3D for immersive environments. In *Proceedings of the ninth international conference on 3D Web technology* (New York, NY, USA, 2004), Web3D '04, ACM, pp. 71–78.

[9] Behr, J., Eschler, P., Jung, Y., and Zöllner, M. X3DOM – a DOM-based HTML5/X3D integration model. In *Proceedings Web3D 2009: 14th International Conference on 3D Web Technology* (New York, USA, 2009), ACM Press, pp. 127–135.

[10] Behr, J., Jung, Y., Keil, J., Drevensek, T., Eschler, P., Zöllner, M., and Fellner, D. W. A scalable architecture for the HTML5/ X3D integration model X3DOM. In *Proceedings Web3D 2010: 15th International Conference on 3D Web Technology* (New York, USA, 2010), ACM Press, pp. 185–193.

[11] Bekaert, P., Sbert, M., and Halton, J. Accelerating path tracing by re-using paths. In *Proceedings of the 13th Eurographics workshop on Rendering* (Aire-la-Ville, Switzerland, Switzerland, 2002), EGRW '02, Eurographics Association, pp. 125–134.

[12] Berthelot, R. B., Royan, J., Duval, T., and Arnaldi, B. Scene graph adapter: an efficient architecture to improve interoperability between 3D formats and 3D applications engines. In *Proceedings of the 16th International Conference on 3D Web Technology* (New York, NY, USA, 2011), Web3D '11, ACM, pp. 21–29.

[13] Bierbaum, A., Just, C., Hartling, P., Meinert, K., Baker, A., and Cruz-Neira, C. VR Juggler: A virtual platform for virtual reality application development. In *Proceedings of the Virtual Reality 2001 Conference (VR'01)* (Washington, DC, USA, 2001), VR '01, IEEE Computer Society, pp. 89–96.

[14] Bikker, J. *Ray Tracing in Real-time Games.* PhD thesis, Delft University of Technology, Delft, The Netherlands, 2012. `http://igad.nhtv.nl/~bikker/files/thesis_jbikker.pdf`.

[15] Black, M. J., Sapiro, G., Marimont, D. H., and Heeger, D. Robust anisotropic diffusion. *IEEE Transactions on Image Processing 7* (1998), 421–432.

[16] Blackwell, H. R. Luminance difference thresholds. *Visual Psychophysics (Handbook of Sensory Physiology) 7*, 4 (1972), 78–101.

[17] Born, M., and Wolf, E. *Principles of Optics*, 7th ed. Cambridge University Press, 1999.

[18] Brigade Dev Team. Brigade real-time path tracer. http://igad.nhtv.nl/~bikker/, 2012.

[19] Brüderlin, B., Heyer, M., and Pfützner, S. Interviews3D: A platform for interactive handling of massive data sets. *IEEE Comput. Graph. Appl. 27* (November 2007), 48–59.

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

[21] Buades, A., Coll, B., and Morel, J.-M. Neighborhood filters and PDEs. *Numerische Mathematik 105* (October 2006), 1–34.

[22] Chen, J., Paris, S., and Durand, F. Real-time edge-aware image processing with the bilateral grid. *ACM Trans. Graph. 26* (July 2007), 103:1–103:8.

[23] Chen, Y.-C., Lei, S. I. E., and Chang, C.-F. Spatio-temporal filtering of indirect lighting for interactive global illumination. *Computer Graphics Forum 31*, 1 (2012), 189–201.

[24] Choudhury, P., and Tumblin, J. The trilateral filter for high contrast images and meshes. In *ACM SIGGRAPH 2005 Courses* (New York, NY, USA, 2005), SIGGRAPH '05, ACM.

[25] Christensen, P. H., and Batali, D. An irradiance atlas for global

illumination in complex production scenes. In *Rendering Techniques* (2004), A. Keller and H. W. Jensen, Eds., Eurographics Association, pp. 133–142.

[26] Colbert, M., Premoze, S., and Francois, G. Importance sampling for production rendering, 2010.

[27] Dachsbacher, C., Křivánek, J., Hašan, M., Arbree, A., Walter, B., and Novák, J. Scalable realistic rendering with many-light methods. *Computer Graphics Forum 32*, 1 (2013).

[28] Dammertz, H., Sewtz, D., Hanika, J., and Lensch, H. P. A. Edge-avoiding À-trous wavelet transform for fast global illumination filtering. In *Proceedings of the Conference on High Performance Graphics* (2010), HPG '10, Eurographics Association, pp. 67–75.

[29] DeCoro, C., Weyrich, T., and Rusinkiewicz, S. Density-based outlier rejection in monte carlo rendering. *Computer Graphics Forum 29*, 7 (2010), 2119–2125.

[30] Dietrich, A., Wald, I., Benthin, C., and Slusallek, P. The OpenRT application programming interface – towards a common API for interactive ray tracing. In *Proceedings of the 2003 OpenSG Symposium* (Darmstadt, Germany, 2003), Eurographics Association, pp. 23–31.

[31] Dietrich, A., Wald, I., Wagner, M., and Slusallek, P. VRML scene graphs on an interactive ray tracing engine. In *Proceedings of the IEEE Virtual Reality 2004* (Washington, DC, USA, 2004), IEEE Computer Society, pp. 109–116.

[32] Dirk Reiners and OpenSG Dev Team. OpenSG documentation. http://www.opensg.org/, 2009.

[33] Döllner, J., and Hinrichs, K. A generalized scene graph. In *VMV-00: Proceedings of the 2000 Conference on Vision Modeling and Visualization* (2000), Aka GmbH, pp. 247–254.

[34] Döllner, J., and Hinrichs, K. A generic rendering system. *IEEE Trans. Vis. Comput. Graph. 8*, 2 (2002), 99–118.

[35] Durand, F., and Dorsey, J. Fast bilateral filtering for the display of high-dynamic-range images. *ACM Trans. Graph. 21* (July 2002), 257–266.

[36] Durand, F., Holzschuch, N., Soler, C., Chan, E., and Sillion, F. X. A frequency analysis of light transport. *ACM Trans. Graph. 24*, 3 (2005), 1115–1126.

[37] Düssel, T., Zilken, H., Frings, W., Eickermann, T., Gerndt, A., Wolter, M., and Kuhlen, T. Distributed collaborative data analysis with heterogeneous visualisation systems. In *EGPGV 2007: Eurographics Symposium on Parallel Graphics and Visualization* (2007), Eurographics Association, pp. 21–28.

[38] Dutre, P., Bala, K., Bekaert, P., and Shirley, P. *Advanced Global Illumination*. AK Peters Ltd, 2006.

[39] Egan, K., Hecht, F., Durand, F., and Ramamoorthi, R. Frequency analysis and sheared filtering for shadow light fields of complex occluders. *ACM Trans. Graph. 30*, 2 (2011), 9:1–9:13.

[40] Egan, K., Tseng, Y.-T., Holzschuch, N., Durand, F., and Ramamoorthi, R. Frequency analysis and sheared reconstruction for rendering motion blur. In *ACM SIGGRAPH 2009 papers* (New York, NY, USA, 2009), SIGGRAPH '09, ACM, pp. 93:1–93:13.

[41] Eisemann, E., and Durand, F. Flash photography enhancement via intrinsic relighting. *ACM Trans. Graph. 23* (August 2004), 673–678.

[42] Felsberg, M., Forssen, P.-E., and Scharr, H. Channel smoothing: Efficient robust smoothing of low-level signal features. *IEEE Trans. Pattern Anal. Mach. Intell. 28* (February 2006), 209–222.

[43] Fraunhofer IGD – InstantReality Dev Team. The InstantReality system. http://instantreality.org/, 2012.

[44] Frey, S., and Ertl, T. Patraco: A framework enabling the transparent and efficient programming of heterogeneous compute networks. In *EGPGV 2010: Eurographics Symposium on Parallel Graphics and Visualization* (2010), Eurographics Association, pp. 131–140.

[45] Gastal, E. S. L., and Oliveira, M. M. Domain transform for edge-aware image and video processing. *ACM Trans. Graph. 30* (August 2011), 69:1–69:12.

[46] Gastal, E. S. L., and Oliveira, M. M. Adaptive manifolds for real-time high-dimensional filtering. *ACM Trans. Graph. 31*, 4 (July 2012), 33:1–33:13.

[47] Georgiev, I., Křivánek, J., Popov, S., and Slusallek, P. Importance caching for complex illumination. *Computer Graphics Forum (Proceedings of EG 2012) 31*, 2 (2012).

[48] Georgiev, I., Křivánek, J., Davidovič, T., and Slusallek, P. Light transport simulation with vertex connection and merging. *ACM Trans. Graph. 31*, 6 (Nov. 2012), 192:1–192:10.

[49] Georgiev, I., Křivánek, J., and Slusallek, P. Bidirectional light transport with vertex merging. In *SIGGRAPH Asia 2011 Sketches* (New York, NY, USA, 2011), SA '11, ACM, pp. 27:1–27:2.

[50] Georgiev, I., and Slusallek, P. RTfact: Generic concepts for flexible and high performance ray tracing. In *Proceedings of the IEEE / EG Symposium on Interactive Ray Tracing 2008* (2008), IEEE Computer Society, Eurographics Association, IEEE, pp. 115–122.

[51] Glare Technologies Limited. Indigo RT. http://www.indigorenderer.com/indigo_rt, 2012.

[52] Hachisuka, T., Jarosz, W., Bouchard, G., Christensen, P.,

Frisvad, J. R., Jakob, W., Jensen, H. W., Kaschalk, M., Knaus, C., Selle, A., and Spencer, B. State of the art in photon density estimation. In *ACM SIGGRAPH 2012 Courses* (New York, NY, USA, 2012), SIGGRAPH '12, ACM, pp. 6:1–6:469.

[53] Hachisuka, T., Jarosz, W., and Jensen, H. W. A progressive error estimation framework for photon density estimation. *ACM Trans. Graph. 29*, 6 (Dec. 2010), 144:1–144:12.

[54] Hachisuka, T., Jarosz, W., Weistroffer, R. P., Dale, K., Humphreys, G., Zwicker, M., and Jensen, H. W. Multidimensional adaptive sampling and reconstruction for ray tracing. *ACM Trans. Graph. 27*, 3 (Aug. 2008), 33:1–33:10.

[55] Hachisuka, T., and Jensen, H. W. Stochastic progressive photon mapping. *ACM Trans. Graph. 28*, 5 (Dec. 2009), 141:1–141:8.

[56] Hachisuka, T., Ogaki, S., and Jensen, H. W. Progressive photon mapping. *ACM Trans. Graph. 27*, 5 (Dec. 2008), 130:1–130:8.

[57] Hachisuka, T., Pantaleoni, J., and Jensen, H. W. A path space extension for robust light transport simulation. *NVIDIA Technical Report NVR-2012-001* (Dec. 2012).

[58] He, K., Sun, J., and Tang, X. Guided image filtering. In *Proceedings of the 11th European conference on Computer vision: Part I* (Berlin, Heidelberg, 2010), ECCV'10, Springer-Verlag, pp. 1–14.

[59] Hecht, E. *Optics*, 4th ed. Addison-Wesley, 1998.

[60] Heckbert, P. S. Adaptive radiosity textures for bidirectional ray tracing. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1990), SIGGRAPH '90, ACM, pp. 145–154.

[61] Henrich, N., Baerz, J., Grosch, T., and Mueller, S. Accelerating path tracing by eye-path reprojection. *International Congress*

on Graphics and Virtual Reality (GRVR) (2011).

[62] Humphreys, G., Houston, M., Ng, R., Frank, R., Ahern, S., Kirchner, P. D., and Klosowski, J. T. Chromium: a stream-processing framework for interactive rendering on clusters. *ACM Trans. Graph. 21* (July 2002), 693–702.

[63] Igehy, H. Tracing ray differentials. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1999), SIGGRAPH '99, ACM Press/Addison-Wesley Publishing Co., pp. 179–186.

[64] Immel, D. S., Cohen, M. F., and Greenberg, D. P. A radiosity method for non-diffuse environments. *SIGGRAPH Comput. Graph. 20*, 4 (Aug. 1986), 133–142.

[65] Intel Software. Embree – photo-realistic ray tracing kernels. http://software.intel.com/en-us/articles/embree-photo-realistic-ray-tracing-kernels/, 2012.

[66] Jensen, H. W. *Realistic Image Synthesis Using Photon Mapping*. A. K. Peters, Ltd., Natick, MA, USA, 2001.

[67] Jensen, H. W., and Christensen, N. J. Optimizing path tracing using noise reduction filters. In *The Third International Conference in Central Europe on Computer Graphics and Visualization* (1995), WSCG '95, pp. 134–142.

[68] Kajiya, J. T. The rendering equation. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques* (1986), SIGGRAPH '86, ACM, pp. 143–150.

[69] Kalantari, N. K., and Sen, P. Removing the noise in Monte Carlo rendering with general image denoising algorithms. *Computer Graphics Forum (Proceedings of Eurographics 2013) 32*, 2 (2013).

[70] Kaplanyan, A. S., and Dachsbacher, C. Adaptive progressive photon mapping. *ACM Trans. Graph. 32*, 2 (2013), 16:1–16:13.

[71] Keller, A. Instant radiosity. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1997), SIGGRAPH '97, ACM Press/Addison-Wesley Publishing Co., pp. 49–56.

[72] Keller, A. *Quasi-Monte Carlo Methods for Photorealistic Image Synthesis*. Shaker, 1998.

[73] Keller, A., Premoze, S., and Raab, M. Advanced (quasi) monte carlo methods for image synthesis. In *ACM SIGGRAPH 2012 Courses* (New York, NY, USA, 2012), SIGGRAPH '12, ACM, pp. 21:1–21:46.

[74] Knaus, C., and Zwicker, M. Progressive photon mapping: A probabilistic approach. *ACM Trans. Graph. 30*, 3 (May 2011), 25:1–25:13.

[75] Kontkanen, J., Räsänen, J., and Keller, A. Irradiance filtering for monte carlo ray tracing. In *Monte Carlo and Quasi-Monte Carlo Methods 2004* (2004), Springer, pp. 259–272.

[76] Kopf, J., Cohen, M. F., Lischinski, D., and Uyttendaele, M. Joint bilateral upsampling. *ACM Trans. Graph. 26* (July 2007).

[77] Křivánek, J., Fajardo, M., Christensen, P. H., Tabellion, E., Bunnell, M., Larsson, D., and Kaplanyan, A. Global illumination across industries. In *ACM SIGGRAPH 2010 Courses* (New York, NY, USA, 2010), SIGGRAPH '10, ACM.

[78] Křivánek, J., Gautron, P., Pattanaik, S., and Bouatouch, K. Radiance caching for efficient global illumination computation. *IEEE Transactions on Visualization and Computer Graphics 11*, 5 (2005), 550–561.

[79] Křivánek, J., Gautron, P., Ward, G., Jensen, H. W., Christensen, P. H., and Tabellion, E. Practical global illumination

with irradiance caching. In *ACM SIGGRAPH 2008 classes* (2008), SIGGRAPH '08, ACM, pp. 60:1–60:20.

[80] Lafortune, E. P., and Willems, Y. D. Bi-directional path tracing. In *Proceedings of COMPUGRAPHICS '93* (1993), pp. 145–153.

[81] Lee, M. E., and Redner, R. A. Filtering: A note on the use of nonlinear filtering in computer graphics. *IEEE Comput. Graph. Appl. 10* (May 1990), 23–29.

[82] Lehtinen, J., Aila, T., Chen, J., Laine, S., and Durand, F. Temporal light field reconstruction for rendering distribution effects. *ACM Trans. Graph. 30*, 4 (July 2011), 55:1–55:12.

[83] Lehtinen, J., Aila, T., Laine, S., and Durand, F. Reconstructing the indirect light field for global illumination. *ACM Trans. Graph. 31*, 4 (July 2012), 51:1–51:10.

[84] Mantiuk, R., Kim, K. J., Rempel, A. G., and Heidrich, W. HDR-VDP-2: A calibrated visual metric for visibility and quality predictions in all luminance conditions. *ACM Trans. Graph. 30*, 4 (July 2011), 40:1–40:14.

[85] McAuley, S., Hill, S., Hoffman, N., Gotanda, Y., Smits, B., Burley, B., and Martinez, A. Practical physically-based shading in film and game production. In *ACM SIGGRAPH 2012 Courses* (New York, NY, USA, 2012), SIGGRAPH '12, ACM, pp. 10:1–10:7.

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

[87] McDonald, R., and Smith, K. J. CIE94 - a new colour-difference formula. *Journal of the Society of Dyers and Colourists 111*, 12 (1995), 376–379.

[88] Mehta, S. U., Wang, B., Ramamoorthi, R., and Durand, F. Axis-aligned filtering for interactive physically-based diffuse indirect

lighting. *ACM Trans. Graph. 32*, 4 (July 2013).

[89] Mitchell, D. P. Generating antialiased images at low sampling densities. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques* (1987), SIGGRAPH '87, ACM, pp. 65–72.

[90] Moon, B., Jun, J., Lee, J., Kim, K., Hachisuka, T., and Yoon, S. Robust image denoising using a virtual flash image for monte carlo ray tracing. *Computer Graphics Forum* (2013). To appear.

[91] Nickolls, J., Buck, I., Garland, M., and Skadron, K. Scalable parallel programming with CUDA. *Queue 6*, 2 (Mar. 2008), 40–53.

[92] OTOY/Refractive Software. Octane Render. http://render.otoy.com, 2012.

[93] Ou, J., Karlik, O., Krivanek, J., and Pellacini, F. Toward evaluating progressive rendering methods in appearance design tasks. Tech. Rep. TR2012-715, Dartmouth College, Computer Science, Hanover, NH, May 2012.

[94] Overbeck, R. S., Donner, C., and Ramamoorthi, R. Adaptive wavelet rendering. *ACM Trans. Graph. 28*, 5 (Dec. 2009), 140:1–140:12.

[95] Ošlejšek, R., and Sochor, J. A flexible, low-level scene graph traversal with explorers. In *Proceedings of the 21st spring conference on Computer graphics* (New York, NY, USA, 2005), SCCG '05, ACM, pp. 203–210.

[96] Pajot, A., Barthe, L., and Paulin, M. Sample-space bright spots removal using density estimation. In *Proceedings of Graphics Interface 2011* (2011), GI '11, Canadian Human-Computer Communications Society, pp. 159–166.

[97] Paris, S., and Durand, F. A fast approximation of the bilateral filter using a signal processing approach. In *In Proceedings of the Eu-*

ropean Conference on Computer Vision (2006), pp. 568–580.

[98] Paris, S., and Durand, F. A fast approximation of the bilateral filter using a signal processing approach. *Int. J. Comput. Vision 81* (January 2009), 24–52.

[99] Parker, S. G., Bigler, J., Dietrich, A., Friedrich, H., Hoberock, J., Luebke, D., McAllister, D., McGuire, M., Morley, K., Robison, A., and Stich, M. Optix: a general purpose ray tracing engine. *ACM Transactions on Graphics 29* (July 2010), 66:1–66:13.

[100] Perona, P., and Malik, J. Scale-space and edge detection using anisotropic diffusion. *IEEE Trans. Pattern Anal. Mach. Intell. 12* (July 1990), 629–639.

[101] Petschnigg, G., Szeliski, R., Agrawala, M., Cohen, M., Hoppe, H., and Toyama, K. Digital photography with flash and no-flash image pairs. *ACM Trans. Graph. 23* (August 2004), 664–672.

[102] Pham, T. Q., and Vliet, L. J. Separable bilateral filtering for fast video preprocessing. In *IEEE Internat. Conf. on Multimedia & Expo, CD1–4* (2005), IEEE, pp. 1–4.

[103] Pharr, M., and Humphreys, G. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann, 2004.

[104] Porikli, F. Constant time O(1) bilateral filtering. In *CVPR* (2008), pp. 1–8.

[105] Raffin, B., Carbonnier, H., Esnault, J., Lombardo, J.-C., Felix, R., Duval, T., Chauffaut, A., Dumont, G., Gaugne, R., Faure, V. G. F., Allard, J., Primet, R., Huot, S., Jung, Y., Bockholt, U., Behr, J., Schwenk, K., and Voss, G. The VCoRE project: First steps towards building a next-generation visual computing platform. In *Journees de l'AFRV* (Strasbourg, France, October 2012).

[106] Reiners, D., Voss, G., and Behr, J. OpenSG: Basic concepts. In *In 1. OpenSG Symposium OpenSG* (2002).

[107] Reinhard, E., Ward, G., Pattanaik, S., and Debevec, P. *High Dynamic Range Imaging: Acquisition, Display, and Image-Based Lighting (The Morgan Kaufmann Series in Computer Graphics)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[108] Reitmayr, G., and Schmalstieg, D. Flexible parametrization of scene graphs. In *Proceedings of the 2005 IEEE Conference 2005 on Virtual Reality* (Washington, DC, USA, 2005), VR '05, IEEE Computer Society, pp. 51–58.

[109] Repplinger, M., Löffler, A., Rubinstein, D., and Slusallek, P. URay: A flexible framework for distributed rendering and display. *Technical Report, Saarland University* (2008).

[110] Ritschel, T., Dachsbacher, C., Grosch, T., and Kautz, J. The state of the art in interactive global illumination. *Computer Graphics Forum 31*, 1 (2012), 160–188.

[111] Roland, S. Bias compensation for photon maps. *Computer Graphics Forum 22*, 4 (2003), 729–742.

[112] Roth, M., Voss, G., and Reiners, D. Multi-threading and clustering for scene graph systems. *Computers & Graphics 28*, 1 (2004), 63 – 66.

[113] Rougeron, G., and Péroche, B. Color fidelity in computer graphics: A survey. *Computer Graphics Forum 17*, 1 (1998), 3–16.

[114] Rousselle, F., Knaus, C., and Zwicker, M. Adaptive sampling and reconstruction using greedy error minimization. *ACM Trans. Graph. 30*, 6 (Dec. 2011), 159:1–159:12.

[115] Rousselle, F., Knaus, C., and Zwicker, M. Adaptive rendering with non-local means filtering. *ACM Trans. Graph. 31*, 6 (Nov. 2012),

195:1–195:11.

[116] Rubinstein, D., Georgiev, I., Schug, B., and Slusallek, P. RTSG: Ray tracing for X3D via a flexible rendering framework. In *Proceedings of the 14th International Conference on 3D Web Technology* (New York, NY, USA, 2009), Web3D '09, ACM, pp. 43–50.

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

[118] Scherzer, D., Nguyen, C. H., Ritschel, T., and Seidel, H.-P. Pre-convolved radiance caching. *Computer Graphics Forum (Proceedings of EGSR 2012) 4*, 31 (2012).

[119] Schwenk, K. Radiance filtering: Interactive low-noise previews of path traced global illumination, 2012. Submitted to *Computer Graphics Forum*, under review.

[120] Schwenk, K., Behr, J., and Fellner, D. W. CommonVolumeShader: simple and portable specification of volumetric light transport in X3D. In *Proceedings of the 16th International Conference on 3D Web Technology* (2011), Web3D '11, ACM, pp. 39–43.

[121] Schwenk, K., Behr, J., and Fellner, D. W. Filtering noise in progressive stochastic ray tracing – four optimizations to improve speed and robustness. *The Visual Computer 29*, 5 (2013), 359–368. Also appeared in *Proceedings of CGI 2012*.

[122] Schwenk, K., and Drevensek, T. Radiance filtering for interactive path tracing. In *ACM SIGGRAPH 2012 Posters* (New York, NY, USA, 2012), SIGGRAPH '12, ACM, pp. 109:1–109:1.

[123] Schwenk, K., Jung, Y., Behr, J., and Fellner, D. W. A modern declarative surface shader for X3D. In *Proceedings of the 15th*

International Conference on Web 3D Technology (2010), Web3D '10, ACM, pp. 7–16.

[124] Schwenk, K., Jung, Y., Voss, G., Sturm, T., and Behr, J. CommonSurfaceShader revisited: improvements and experiences. In *Proceedings of the 17th International Conference on 3D Web Technology* (New York, NY, USA, 2012), Web3D '12, ACM, pp. 93–96.

[125] Schwenk, K., Kuijper, A., Behr, J., and Fellner, D. W. Practical noise reduction for progressive stochastic ray tracing with perceptual control. *IEEE Computer Graphics and Applications 32* (2012), 46–55.

[126] Schwenk, K., Voss, G., Behr, J., Jung, Y., Limper, M., Herzig, P., and Kuijper, A. Extending a distributed virtual reality system with exchangeable rendering back-ends. *The Visual Computer* (2013), 1–11. Online first. This is an extended version of our CW2012 paper *A System Architecture for Flexible Rendering Back-ends in Distributed Virtual Reality Applications*.

[127] Sen, P., and Darabi, S. On filtering the noise from the random parameters in monte carlo rendering. *ACM Trans. Graph. 31*, 3 (June 2012), 18:1–18:15.

[128] Shirley, P., Wade, B., Hubbard, P. M., Zareski, D., Walter, B., and Greenberg, D. P. Global illumination via density-estimation. In *Proceedings of 6th Workshop on Rendering* (1995), Springer, pp. 219–230.

[129] Smith, S. M., and Brady, J. M. SUSAN – a new approach to low level image processing. *Int. J. Comput. Vision 23* (May 1997), 45–78.

[130] Staadt, O. G., Walker, J., Nuber, C., and Hamann, B. A survey and performance analysis of software platforms for interactive cluster-based multi-screen rendering. In *ACM SIGGRAPH ASIA 2008 courses* (New York, NY, USA, 2008), SIGGRAPH Asia '08, ACM, pp. 41:1–41:10.

[131] Steinicke, F., Ropinski, T., and Hinrichs, K. A generic virtual reality software system's architecture and application. In *Proceedings of the 2005 international conference on Augmented tele-existence* (New York, NY, USA, 2005), ICAT '05, ACM, pp. 220–227.

[132] Suykens, F., and Willems, Y. D. Adaptive filtering for progressive monte carlo image rendering. In *The 8th International Conference in Central Europe on Computer Graphics, Visualization and Interactive Digital Media* (2000), WSCG '00.

[133] Tole, P., Pellacini, F., Walter, B., and Greenberg, D. P. Interactive global illumination in dynamic scenes. *ACM Trans. Graph. 21*, 3 (July 2002), 537–546.

[134] Tomasi, C., and Manduchi, R. Bilateral filtering for gray and color images. In *Proceedings of the Sixth International Conference on Computer Vision* (Washington, DC, USA, 1998), ICCV '98, IEEE Computer Society, pp. 839–846.

[135] van Antwerpen, D. A survey of importance sampling applications in unbiased physically based rendering. Tech. rep., TU Delft, 2011. `http://graphics.tudelft.nl/~dietger/survey.pdf`.

[136] van Antwerpen, D. Unbiased physically based rendering on the gpu. Tech. rep., TU Delft, 2011. `http://graphics.tudelft.nl/~dietger/thesis/index.html`.

[137] Veach, E. *Robust monte carlo methods for light transport simulation*. PhD thesis, Stanford University, Stanford, CA, USA, 1998. AAI9837162.

[138] Veach, E., and Guibas, L. Bidirectional estimators for light transport. In *Proceedings of Eurographics Rendering Workshop* (1994).

[139] Veach, E., and Guibas, L. J. Metropolis light transport. In *Proceedings of the 24th annual conference on Computer graphics and inter-*

active techniques (New York, NY, USA, 1997), SIGGRAPH '97, ACM Press/Addison-Wesley Publishing Co., pp. 65–76.

[140] Voss, G., Behr, J., Reiners, D., and Roth, M. A multi-thread safe foundation for scene graphs and its extension to clusters. In *Proceedings of the Fourth Eurographics Workshop on Parallel Graphics and Visualization* (Aire-la-Ville, Switzerland, Switzerland, 2002), EGPGV '02, Eurographics Association, pp. 33–37.

[141] Voss, G., and Reiners, D. Towards a flexible back-end for scenegraph-based rendering systems. In *Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia* (New York, NY, USA, 2006), GRAPHITE '06, ACM, pp. 303–309.

[142] Wald, I., Kollig, T., Benthin, C., Keller, A., and Slusallek, P. Interactive global illumination using fast ray tracing. In *Proceedings of the 13th Eurographics workshop on Rendering* (2002), EGRW '02, Eurographics Association, pp. 15–24.

[143] Walter, B. *Density estimation techniques for global illumination.* PhD thesis, Cornell University, Ithaca, NY, USA, 1998.

[144] Walter, B., Drettakis, G., and Parker, S. Interactive rendering using the render cache. In *Rendering techniques '99 (Proceedings of EGWR 1999)* (1999), vol. 10, Springer-Verlag/Wien, pp. 235–246.

[145] Ward, G. J., Rubinstein, F. M., and Clear, R. D. A ray tracing solution for diffuse interreflection. In *Proceedings of the 15th annual conference on Computer graphics and interactive techniques* (1988), SIGGRAPH '88, ACM, pp. 85–92.

[146] Weiss, B. Fast median and bilateral filtering. *ACM Trans. Graph. 25* (July 2006), 519–526.

[147] Woo, M., Neider, J., Davis, T., and Shreiner, D. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version*

1.2, 3rd ed. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[148] Xu, Q., and Sbert, M. A new way to re-using paths. In *Proceedings of the 2007 international conference on Computational science and Its applications - Volume Part II* (Berlin, Heidelberg, 2007), ICCSA'07, Springer-Verlag, pp. 741–750.

[149] Xu, R., and Pattanaik, S. N. A novel monte carlo noise reduction operator. *IEEE Comput. Graph. Appl. 25* (March 2005), 31–35.

[150] Yang, L., Sander, P. V., Lawrence, J., and Hoppe, H. Antialiasing recovery. *ACM Trans. Graph. 30* (May 2011), 22:1–22:9.

[151] Yang, Q., Tan, K.-H., and Ahuja, N. Real-time O(1) bilateral filtering. In *CVPR* (2009), pp. 557–564.