Hierarchical Variance Reduction Techniques for Monte Carlo Rendering

Clarberg, Petrik

2012

Hierarchical Variance Reduction Techniques for Monte Carlo Rendering

Petrik Clarberg

Department of Computer Science
Lund University, Sweden

October 2012

This dissertation was submitted to Lund University in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Engineering.

LUND UNIVERSITY

ISBN 978-91-976939-9-8
ISSN 1404-1219
Dissertation 40, 2012
LU-CS-DISS:2012-3

Department of Computer Science
Lund Institute of Technology
Lund University
Box 118
SE-221 00 Lund
Sweden

E-mail: petrik@cs.lth.se
WWW: http://www.cs.lth.se/home/Petrik_Clarberg

Typeset using $\LaTeX 2\varepsilon$
Printed in Sweden by Tryckeriet i E-huset, Lund, 2012

Abstract

Ever since the first three-dimensional computer graphics appeared half a century ago, the goal has been to model and simulate how light interacts with materials and objects to form an image. The ultimate goal is *photorealistic rendering*, where the created images reach a level of accuracy that makes them indistinguishable from photographs of the real world. There are many applications — visualization of products and architectural designs yet to be built, special effects, computer-generated films, virtual reality, and video games, to name a few. However, the problem has proven tremendously complex; the illumination at any point is described by a recursive integral to which a closed-form solution seldom exists. Instead, computer simulation and *Monte Carlo* methods are commonly used to statistically estimate the result. This introduces undesirable noise, or *variance*, and a large body of research has been devoted to finding ways to reduce the variance. I continue along this line of research, and present several novel techniques for variance reduction in Monte Carlo rendering, as well as a few related tools.

The research in this dissertation focuses on using *importance sampling* to pick a small set of well-distributed point samples. As the primary contribution, I have developed the first methods to explicitly draw samples from the *product* of distant high-frequency lighting and complex reflectance functions. By sampling the product, low noise results can be achieved using a very small number of samples, which is important to minimize the rendering times. Several different *hierarchical* representations are explored to allow efficient product sampling. In the first publication, the key idea is to work in a compressed wavelet basis, which allows fast evaluation of the product. Many of the initial restrictions of this technique were removed in follow-up work, allowing higher-resolution uncompressed lighting and avoiding precomputation of reflectance functions. My second main contribution is to present one of the first techniques to take the *triple* product of lighting, visibility and reflectance into account to further reduce the variance in Monte Carlo rendering. For this purpose, *control variates* are combined with importance sampling to solve the problem in a novel way. A large part of the technique also focuses on analysis and approximation of the visibility function. To further refine the above techniques, several useful tools are introduced. These include a fast, low-distortion map to represent (hemi)spherical functions, a method to create high-quality quasi-random points, and an optimizing compiler for analyzing shaders using interval arithmetic. The latter automatically extracts bounds for importance sampling of arbitrary shaders, as opposed to using a priori known reflectance functions.

In summary, the work presented here takes the field of computer graphics one step further towards making photorealistic rendering practical for a wide range of uses. By introducing several novel Monte Carlo methods, more sophisticated lighting and materials can be used without increasing the computation times. The research is aimed at domain-specific solutions to the rendering problem, but I believe that much of the new theory is applicable in other parts of computer graphics, as well as in other fields.

Acknowledgements

There is a long list of people and organizations to whom I owe my sincerest gratitude for supporting this work. The work presented in this dissertation was partly carried out within the Computer Graphics Group at the Department of Computer Science, Lund University, and partly at Intel Corporation. It was partly funded by the Swedish Foundation for Strategic Research (SSF) and Intel Corporation.

First and foremost, I would like to thank my supervisor professor Tomas Akenine-Möller for always believing in me, for all the feedback, and for allowing me to freely pursue the research directions that I have found most interesting and challenging. A large part of the research was conducted in the Lund University Graphics Group, and I would like to thank my fellow research colleagues at the time, Jacob Munkberg, Jon Hasselgren, Jim Rasmusson, and Calle Lejdfors, for providing a fun, inspiring and challenging research environment. Second, I owe my gratitude to the staff at the Department of Computer Science at Lund University, notably assistant supervisor Lennart Ohlsson, professor and director for graduate studies Görel Hedin, and associate professor Michael Doggett.

Part of the research was carried out in collaborations with other universities; I especially want to thank Henrik Wann Jensen for supporting my research visit to the Computer Graphics Laboratory at UCSD, and Wojciech Jarosz for our valuable collaboration on wavelet products. Second, I express my gratitude towards Victor Ostromoukhov for inviting me to the Laboratoire d'Informatique Graphique de l'Université de Montréal (LIGUM). My stay there was a real highlight, and I enjoyed all the interesting discussions with Pierre Poulin, Fabrice Rousselle, Luc Leblanc, and other members of LIGUM. Fabrice and Cynthia deserve a special mentioning for being a great host family during my stay in Montréal.

The last part of the research was carried out after transitioning to the Advanced Rendering Technology team at Intel Corporation. I truly appreciate the stimulating research environment provided by this team. In particular, I would like to thank my current and former co-workers at the office in Lund, many of whom have been mentioned above, as well as Robert Toth, Jim Nilsson, Magnus Andersson, Björn Johnsson, and Rasmus Barringer, for many valuable discussions. Our team's former and current leads Craig Kolb, Matt Pharr, Aaron Lefohn, and Chuck Lingle, also deserve a big thank you for supporting the work.

On a personal note, I express my greatest appreciation to my friends and family for all their encouragement and support. In particular, I want to thank my parents, Hazel and Kennet, and my wonderful sister Vendela for always being there for me. Finally, I thank my wife Kristen for her endless love and support, and little Juliette for keeping us awake and all the joy she spreads. Without them, none of this would have been possible.

List of Publications

This doctoral dissertation is based on the following papers, which will be referred to by their roman numerals in the text:

I P. Clarberg, W. Jarosz, T. Akenine-Möller and H. W. Jensen. *Wavelet Importance Sampling: Efficiently Evaluating Products of Complex Functions*. ACM Transactions on Graphics (Proceedings of SIGGRAPH 2005), 24(3), pp. 1166–1175, 2005.

II P. Clarberg and T. Akenine-Möller. *Practical Product Importance Sampling for Direct Illumination*. Computer Graphics Forum (Proceedings of Eurographics 2008), 27(2), pp. 681–690, 2008.

III P. Clarberg and T. Akenine-Möller. *Exploiting Visibility Correlation in Direct Illumination*. Computer Graphics Forum (Proceedings of EGSR 2008), 27(4), pp. 1125–1136, 2008.

IV F. Rousselle, P. Clarberg, L. Leblanc, V. Ostromoukhov and P. Poulin. *Efficient Product Sampling using Hierarchical Thresholding*. The Visual Computer (Proceedings of CGI 2008), 24(7-9), pp. 465–474, 2008.

V P. Clarberg. *Fast Equal-Area Mapping of the (Hemi)Sphere using SIMD*. Journal of Graphics Tools, 13(3), pp. 53–68, 2008.

VI P. Clarberg, R. Toth, J. Hasselgren and T. Akenine-Möller. *An Optimizing Compiler for Automatic Shader Bounding*. Computer Graphics Forum (Proceedings of EGSR 2010), 29(4), pp. 1259–1268, 2010.

VII J. Munkberg, P. Clarberg, J. Hasselgren, R. Toth, M. Sugihara and T. Akenine-Möller. *Hierarchical Stochastic Motion Blur Rasterization*. In Proceedings of High Performance Graphics, pp. 107–118, 2011.

The following papers were also published by the author, but are not included in this dissertation:

▷ J. Munkberg, P. Clarberg, J. Hasselgren and T. Akenine-Möller. *High Dynamic Range Texture Compression for Graphics Hardware*. ACM Transactions on Graphics (Proceedings of SIGGRAPH 2006), 25(3), pp. 698–706, 2006.

▷ H. Malm, M. Oskarsson, E. Warrant, P. Clarberg, J. Hasselgren and C. Lejdfors. *Adaptive Enhancement and Noise Reduction in Very Low Light-Level Video*. IEEE International Conference on Computer Vision (ICCV), pp. 1–8, 2007.

- ▷ J. Munkberg, P. Clarberg, J. Hasselgren and T. Akenine-Möller. *Practical HDR Texture Compression*. Computer Graphics Forum, 27(6), pp. 1664–1676, 2008.

- ▷ J. Ström, P. Wennersten, J. Rasmusson, J. Hasselgren, J. Munkberg, P. Clarberg and T. Akenine-Möller. *Floating-Point Buffer Compression in a Unified Codec Architecture*. In Proceedings of Graphics Hardware, pp. 75–84, 2008.

- ▷ M. Andersson, B. Johnsson, J. Munkberg, P. Clarberg, J. Hasselgren and T. Akenine-Möller. *Efficient Multi-view Ray Tracing using Edge Detection and Shader Reuse*. The Visual Computer (Proceedings of CGI 2011), 27(6-8), pp. 665–676, 2011.

- ▷ J. Nilsson, P. Clarberg, B. Johnsson, J. Munkberg, J. Hasselgren, R. Toth, M. Salvi and T. Akenine-Möller. *Design and Novel Uses of Higher-Dimensional Rasterization*. In Proceedings of High Performance Graphics, pp. 1–11, 2012.

Contents

Introduction

1 Overview

Broadly speaking, the field of *computer graphics* studies the science (and art) behind creating or manipulating graphics using computers. Graphics can refer to any kind of visual content, but it usually means digital images, which may be printed or displayed, or moving pictures in the form of animations or video. Computer graphics has had a profound impact on the development of computers and the way we interact with them. The research presented in this dissertation has taken the field a small step forward, but before delving into the subject and explaining our work, let me give a brief historical context and introduce many of the terms that will be used later.

1.1 Historical Background

The modern computer is the evolution of early mechanical calculators into more general machines, capable of storing intermediate results and following the instructions of a *program* to perform advanced calculations. At first purely mechanical, these machines developed into the first fully electronic computers in the 1940s. In the early days, computers were built solely to speed up the slow and error-prone task of performing complex numerical calculations, often military, and there was no graphical output to speak of; the results were written on punch cards or teleprinters. The first computers to use a *display* for simple interactive visual output were the Whirlwind [114] and SAGE [62] projects, completed in the 1950s for use in aircraft simulator and air defense systems, respectively. The graphical output were in these cases markers on a radar-like screen.

The invention of the transistor in 1947, followed by the integrated circuit in 1958, were key breakthroughs in the development of reliable, less expensive, and much more powerful computers. It now became viable to build smaller computer systems in research laboratories, such as the TX-2 at MIT and the DEC PDP-1, both transistor-based with a memory in the range of tens to hundreds of kilobytes and using CRT (cathode-ray-tube) displays for output. The higher performance and greater availability of computers allowed a wider range of applications to be ex-

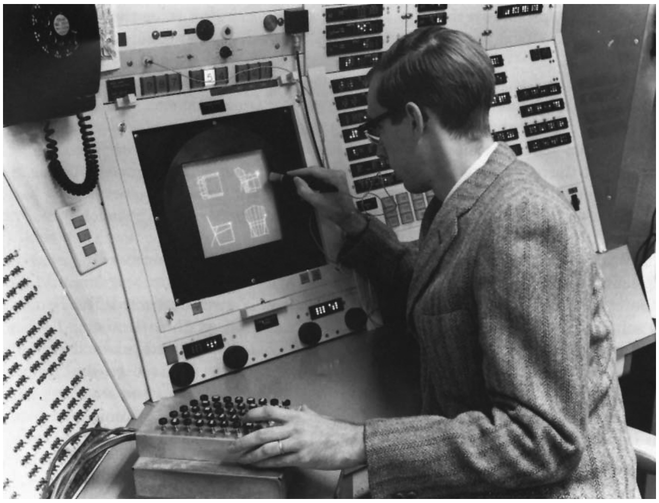


Figure 1: *The Sketchpad system by Ivan Sutherland [139] was one of the very first applications of computer graphics, here demonstrated at MIT in 1963.*

plored. The *Sketchpad* software, published by Sutherland in 1963 [139], is considered by many to be the birth of computer graphics. Running on TX-2, the program allowed a user to sketch simple shapes on the display using a light pen, as shown in Figure 1.

At this time, computer graphics were mostly two-dimensional (2D), created using flat geometric shapes such as lines, curves, and text. When the computer internally works with a three-dimensional geometric model, we speak of *3D graphics*, even though the result is usually converted into a two-dimensional image for display. The research of the 1960s laid the groundwork for many of the basic concepts of 2D and 3D graphics, and there was a growing commercial interest in the new capabilities of the computer. It became apparent that computer graphics would not only revolutionize the design and engineering of new products, and how we interact with computers, but also play a vital role in the entertainment industry. The first video game, *Spacewar*, was created already in 1961 and became a huge success. The game running on PDP-1 featured two blips on the screen representing spaceships, and the goal was to destroy the other player.

The first computer displays were vector based, i.e., lines were drawn by directly controlling the electron beam of the CRT display. As computers got more advanced, so-called *raster* displays emerged. On a raster display, the image is represented by a rectangular grid of dots, or *pixels*. Initially, each pixel could only be either turned on or off, but the technology soon evolved to allow a range of different intensities and even color. Much of the early research focused on the seemingly mundane task of drawing lines, curves and text on raster displays. The process of transforming a geometric shape (e.g., line, curve, or triangle) into a raster image is called *rasterization*. One of the earliest examples is Bresenham's line algorithm [16] developed in 1962 at IBM, which determines which pixels to turn on in order to approximate a straight line between two points. The same principles apply today and computers still work mainly with raster images, i.e., images made up of rectangular grids of pixels. Superficially, one could say that computer graphics is largely about determining the color of each pixel, even though the machinery behind doing so can be tremendously complex, as we will see.

In 1971, the first commercially available *microprocessor* was launched – the Intel 4004. The core of the computer now fit on a single integrated circuit, which set off the introduction of personal computers. The development of faster computers went hand-in-hand with research in more advanced computer graphics algorithms, and new use cases. The 1970s marked the invention of a wide range of techniques, many of which are still in use today.

Previously, the graphical output had mainly been limited to showing the edges or outlines of geometrical objects by drawing lines and curves, but now it became possible to also fill their interiors. By drawing many simple geometric *primitives*, for example, triangles or polygons, more complex three-dimensional surfaces and objects could be constructed. However, first the *visibility* problem [140, 154] had to be solved, which refers to the task of determining which surfaces lie in front of others in the three-dimensional world, and hence are visible in the final image. The problem is difficult, as primitives may partially overlap or even intersect each other. Two main approaches have survived the test of time. The first is to rasterize all primitives, i.e., convert them into pixels, but only keep the pixel that is closest to the viewer at each point on the screen. This is referred to as depth or z-buffering[1] [18]. The second approach, known as *ray tracing* [5, 159], is conceptually simpler; a ray is cast from the observer into the world through each pixel, and the first surface that is encountered along each ray is reported as visible. In practice, however, making ray tracing efficient is difficult and much research has since been done to speed up the operation.

As surfaces and objects could now be created out of filled primitives, and not just their outlines drawn, there was also a need to compute an intensity or color for each pixel. The process of computing the color is known as *shading*, and the color is determined by the surface material of an object and the incoming light. The simplest method is to assign a single color to each primitive, which is called flat shading, but this creates ugly sharp edges between primitives. The interpolation techniques [46, 108] developed in the early 70s were important to get smoother results. Figure 2 shows an overview of these early shading techniques.

Another important milestone was the invention of *texture mapping* by Catmull [18], where an image is placed on the surface of a three-dimensional object to add color and surface details, similar to putting wallpaper on a blank wall. A related technique is *bump mapping* [12], which adds the appearance of a rough surface, for

[1]The "depth" or z-value of a pixel is its distance from the viewer in the three-dimensional world.

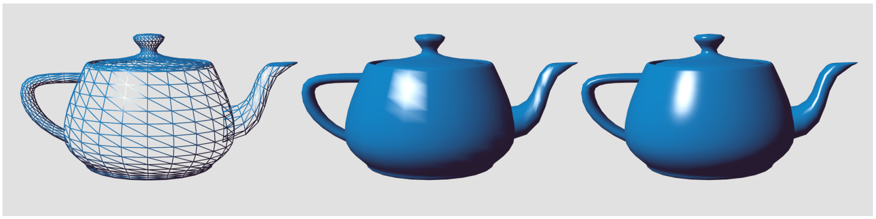


Figure 2: *In the 1970s, it became possible to draw filled geometric primitives instead of just their outlines, as shown on the left. The different interpolation techniques that were developed, such as Gouraud (middle) [46] and Phong (right) [108] interpolation, were the first steps towards realistic shading.*

example, with bumps and wrinkles. To simulate how light from different directions reflects off a surface, and hence affects the shading, simple light *reflection models* started to appear. Some well-known early examples are the Phong [108], Blinn-Phong [11], and Cook-Torrance [25] reflection models. By this time, *rendering*, i.e., computer generation of images from a three-dimensional model, started to become a viable technique for use in movies. Although there had been some earlier attempts, the industry really took off when George Lucas formed the computer graphics division at Lucasfilm in 1979.

By the 1980s, computer graphics was a well-established research field, and a lot of effort was focused on understanding and formalizing the mathematics behind rendering and image formation. Computer graphics up to now had a very artificial look with harsh lighting and sharp edges. A better understanding and simulation of the physics was a necessary step towards *photorealistic* rendering. This applied to all aspects of rendering. For example, the early reflection models were formalized as *bidirectional reflectance distribution functions* (BRDFs), following the definition used in optics [98]. By simulating a physical camera and distributing rays randomly [24], it was possible to render effects such as motion blur and depth of field, which appear in real photographs. The jaggedness, or *aliasing*, of edges that had troubled rendering, was recognized as a signal processing problem; since a display has a limited resolution, each pixel represents a sample of an underlying continuous image. By placing the samples in a non-regular pattern and applying proper filtering and reconstruction techniques, aliasing could be alleviated [30]. Most importantly, the light propagation through a scene was defined by the *rendering equation* [65] as a recursive integral. The physical basis for this is the principle of conservation of energy. The theory behind the rendering equation is often denoted *light transport*, as it deals with the transfer of energy from light sources, via reflections and refractions, to the camera.

Subsequent work in photorealistic rendering, ours included, has attempted to solve the rendering equation using various numerical methods. The solution is usually referred to as *global illumination*, as it takes all aspects of the illumination into

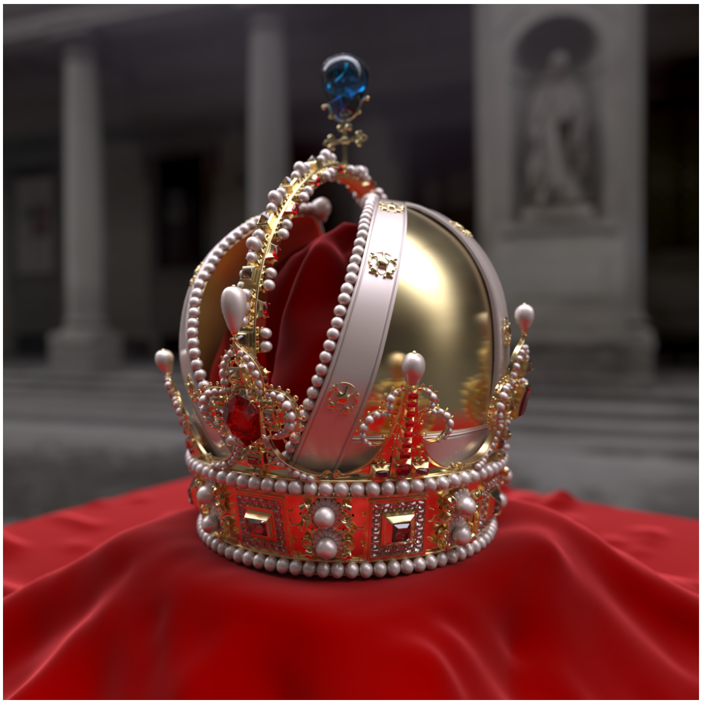


Figure 3: *Example of a photorealistic image rendered using the Intel® Embree ray tracer. The lighting comes from the Uffizi gallery in Florence, and was captured in a light probe courtesy of Paul Debevec [27]. The model is courtesy of Martin Lubich (`http://www.loramel.net/`)*

account, both light that arrives directly from light sources, and the indirect illumination that occurs when light first bounces off other surfaces. Inspired by engineering problems in other fields, two main directions emerged in the mid-80s. The first class of algorithms are *radiosity* methods, which are based on the finite element method (FEM) originating from the 1940s and used for solving civil and structural engineering problems. Although the details vary, the common theme is that the continuous domain is discretized into finite sub-domains (elements), over which the solution is computed. In computer graphics, this means dividing up the three-dimensional world into small patches, and computing the light transport between these patches [44]. The second category is Monte Carlo algorithms, which rely on repeated random sampling; an approximation of the result is computed by averaging over a large number of random, or *stochastic*, samples [23, 30]. The Monte Carlo method was developed in the 1940s by mathematicians working on the Manhattan project, and has been an extremely valuable tool for computer simulation of many physical and mathematical systems.

Although the theory of light transport was now better understood, the rendering problem was by no means solved. Naïve application of both radiosity and Monte Carlo methods requires an enormous amount of computations to give acceptable results, and much further research was needed. Both methods continue to be developed, but the Monte Carlo based approaches have received a lot of attention due to their simplicity and scalability. Their main problem is that the stochastic sampling introduces random *noise* in the result, which leads to grainy images similar to the grain in photographic film. The noise is measured as statistical *variance*, and many techniques have focused on reducing the variance. The 1990s saw the emergence of many advanced Monte Carlo algorithms, such as bidirectional path tracing [71], Metropolis light transport [148], and photon mapping [64]. Figure 3 shows an example of an image created with a modern renderer. The research presented in this dissertation follows along this line of work. We introduce several novel algorithms for smarter sampling in Monte Carlo rendering, along with a few associated tools. In particular, the focus is mainly on evaluating the direct illumination from light sources using Monte Carlo methods.

1.2 Outline

The remainder of this dissertation is organized as follows. First, I will describe the theory of light transport in more detail in Section 2. Each of the terms in the rendering equation are explained, and examples are given of each of the components necessary in an accurate simulation, i.e., light sources, cameras, and reflectance functions. Following this, we will take a closer look at Monte Carlo theory, and in particular, the theory behind the various variance reduction techniques used in our research, in Section 3. Then, the specific contributions of my research are summarized in Section 4. It is my intent that this section should help the reader gain a clearer picture how the different publications relate to each other, and the line of thought that lead us to focus on the chosen problems. This section is followed by a summary of my research methodology in Section 5, which includes a brief discussion on the implementation of the methods. Some concluding remarks and discussion of future research directions are given in Section 6.

Finally, I include the seven publications that form the basis for this dissertation. The papers have been reformatted to fit the layout of this dissertation, but are otherwise unmodified reproductions of the original manuscripts.

2 Light Transport

Photorealistic rendering of an image requires that a sophisticated three-dimensional model of the scene to be depicted is created. The geometric properties of the scene are defined by three-dimensional objects, often created out of simpler rendering primitives, such as triangles or surface patches. The color and appearance of objects are specified by assigning different materials to them, which are defined by their reflectance functions. Finally, the content of the image is decided by placing a virtual camera in the scene, and light is introduced by adding one or more virtual light sources. The rendering of the final image involves a physically accurate simulation of how light propagates through the virtual world. The transfer of light, or *energy*, from the light sources, via surfaces in the scene, and ultimately to the camera, is described by *light transport* theory. In computer graphics, the *rendering equation* [65] neatly summarizes the mathematics under the geometric optics approximation.

In this section, I will explain the details of light transport and the rendering equation. We will also discuss some common camera, light source, and reflection models used in computer graphics, as well as the approximations and representations used in our research.

2.1 The Properties of Light

Let us start by defining the physical quantities of light, and the assumptions commonly made in computer graphics. Visible light is electromagnetic radiation that is visible to the human eye. The characteristics of light are described by its wavelength, polarization, and radiant intensity. As a type of electromagnetic radiation, light additionally exhibits both particle and wave properties.

2.1.1 Physical Characteristics

The *wavelength* of light determines its perceived color; visible light lies in the spectrum from about 380 nm (blue) to 740 nm (red). However, light transport is usually only simulated at three distinct wavelengths corresponding to the additive primary colors red, green, and blue (RGB). These can be combined to any color as the human vision is trichromatic due to the three types of cones in the eye.[2] For details on different color models, we refer to Reinhard et al.'s book [116]. Note that while all colors can be represented, the simulation of light transport only at a few different wavelengths is an approximation. The consequence is that wavelength dependent effects, such as *dispersion* and *fluorescence* are not captured.

Dispersion is the effect that causes white light to split into a rainbow as it passes

[2]Note that the response curves of the cones do not directly correspond to the red, green, and blue primaries. The peaks are rather at greenish-yellow, green, and blue wavelengths, but the response curves are relatively wide to include red on the far side.

Quantity	*Unit*	*Description*
Radiant energy (Q)	J	Energy, measured in joules [J].
Radiant flux (Φ)	W	Energy per unit time in watts [W=J/s].
Radiant intensity (I)	$\mathrm{W{\cdot}sr^{-1}}$	Power per solid angle.
Irradiance (E)	$\mathrm{W{\cdot}m^{-2}}$	Power arriving at a surface per area.
Radiance (L)	$\mathrm{W{\cdot}m^{-2}{\cdot}sr^{-1}}$	Power per projected area per solid angle.

Table 1: *The radiometric SI units used in photorealistic rendering.*

a prism due to the refractive index of materials being wavelength dependent. This effect is generally minor, and indeed, photographic lenses are specifically designed to minimize it. Fluorescence is the emission of light by a substance that has absorbed light at a different wavelength. The emitted light is usually of a longer wavelength, e.g., many fluorescent minerals emit visible light when exposed to ultraviolet light. Although the RGB model is the most common, some rendering methods work at a larger number of wavelengths to achieve so-called *spectral rendering*, in order to simulate dispersion, fluorescence, and related effects [28, 138].

While light propagates in vacuum at a high speed of $3 \cdot 10^8$ m/s, in computer graphics, the speed of light is almost always ignored; energy transfer is assumed to be instantaneous. For all practical purposes, this is a perfectly acceptable approximation. Similarly, the *polarization* of light is often ignored for performance reasons, as its visual impact is typically limited. However, some rendering techniques [144, 160, 162] and reflection models [58] do take polarization into account.

The wave properties of light gives rise to *diffraction* phenomena, e.g., the apparent bending of light around very small objects or light spreading out past small openings. Diffraction effects are mainly visible when light interacts with geometrical features of a size on the same order of magnitude as the light's wavelength. As such fine detail is rare in current computer graphics, it is generally safe to ignore diffraction effects and most rendering methods do so. The exception is, for example, the colorful reflections of CD/DVD discs and certain insects or minerals. Approximations exist to handle such cases [1, 134, 135]. Holographic rendering, i.e., computer generation of holograms, is another application where simulation of diffraction is essential. This is beyond the scope of this dissertation.

2.1.2 Radiometry

The power of light is measured in several different *radiometric* units. The terminology is somewhat confusing since there is an equivalent set of *photometric* units, which measure the perceived power of light, i.e., adjusted for the sensitivity of the human eye. However, in physically-based rendering, we are interested in computing the total energy arriving at each pixel in the virtual camera and work with radiometric units. These quantities are then converted into an image for display using a *tone mapping* operator, which takes the limited dynamic range of the

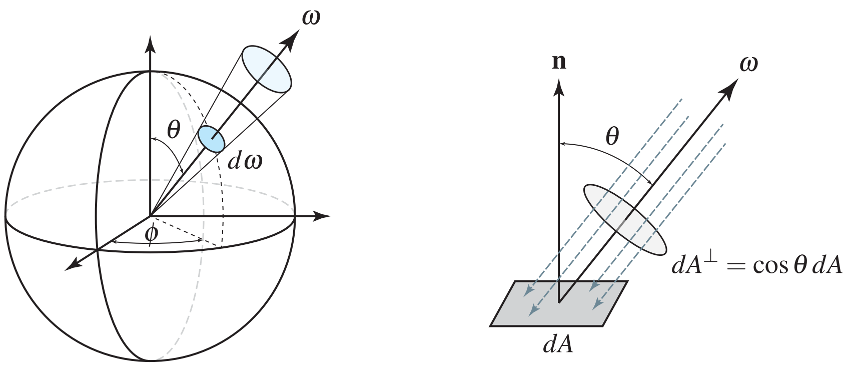


Figure 4: *Left: The direction of a ray is given in terms of its spherical coordinates on the unit sphere, i.e., $\omega = (\theta, \phi)$. In integrals over the (hemi)sphere, we commonly integrate over differential solid angle, $d\omega$. Right: When a ray of light hits (or leaves) a surface with differential area dA, the amount of energy transferred is proportional to the projected area, i.e., the cross-sectional differential area, $dA^{\perp}$, which is perpendicular to the ray. The relation between the two is determined by the angle, θ, between the surface normal,* $\mathbf{n}$*, and the ray direction, ω.*

output device into account. For more information, we refer to the book by Reinhard et al. [115]. The main radiometric units of interest to us are summarized in Table 1.

The *solid angle* refers to the apparent size of an object as seen from a point in space, and its (dimensionless) unit is *steradians* [sr]. The solid angle subtended by a surface is defined as the surface area that its projection covers on the unit sphere. Hence, the full space of directions is 4π [sr], i.e., the area of the unit sphere. To describe a direction, ω, on the sphere or hemisphere, it is convenient to use *spherical coordinates*, (r, θ, ϕ). A point in three-dimensional space is defined by its radial distance, r, polar angle, θ, and azimuthal angle, ϕ. For a unit vector, $r = 1$, and we use $\omega = (\theta, \phi)$ to denote its direction. To integrate over the (hemi)sphere, we use the differential direction, or *differential solid angle*, $d\omega$. These important concepts are illustrated in Figure 4 (left).

We are usually interested in integrating the power of light arriving at a surface per *differential area*, dA. For this purpose, we introduce the *projected area*, $dA^{\perp}$, which is the hypothetical differential area that is *perpendicular* to a given ray. The projected area takes the foreshortening due to the angle of incidence, θ, into account. The relationship between the two is:

$$dA^{\perp} = (\mathbf{n} \cdot \omega)\, dA = \cos\theta\, dA, \tag{1}$$

where $\mathbf{n}$ is the surface normal, and θ is the angle between the ray and the normal, or equivalently, the polar angle of the ray's direction expressed in the coordinate frame of the surface. See Figure 4 (right).

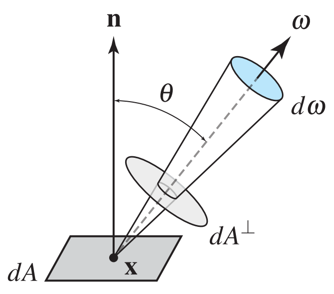


Figure 5: *The radiance of a ray of light that arrives at (or leaves) a surface with differential area dA, is its power (flux), Φ, per projected area per solid angle.*

The *radiant flux*, Φ, of a light source is its total emitted power in watts [W]. However, its *intensity*, I, is the power per solid angle, i.e., $I = d\Phi/d\omega$. For example, if a 100 W point-like light emits light equally in all directions, it has an intensity of $100/4\pi = 7.96$ W·sr^{-1}. In Section 2.4, the properties of some commonly used light sources will be discussed. The *irradiance*, E, is the radiant flux density, or power per unit area, *arriving* at a point, $\mathbf{x}$, on a surface:

$$E(\mathbf{x}) = \frac{d\Phi(\mathbf{x})}{dA(\mathbf{x})}. \tag{2}$$

The equivalent term for light leaving the surface is *radiant exitance* (or *radiosity*), which has the same unit [W·m^{-2}]. To continue our example, consider a point, $\mathbf{x}$, on a surface that is perpendicular to the light source and lies at a distance $r = 10$ m. The irradiance here will be $E(\mathbf{x}) = \Phi/4\pi r^2 = 0.0796$ W·m^{-2}. This also shows that the irradiance from a point-like light source is inversely proportional to its squared distance.

Finally, the *radiance*, L, is perhaps the most useful quantity. It describes the power of light that arrives from a differential solid angle, $d\omega$, and falls on a hypothetical perpendicular differential area, $dA^{\perp}$, as follows (see Figure 5):

$$L(\mathbf{x},\omega) = \frac{d^2\Phi(\mathbf{x},\omega)}{dA^{\perp}d\omega} = \frac{d^2\Phi(\mathbf{x},\omega)}{\cos\theta\, dA d\omega}. \tag{3}$$

We use the notations $L(\mathbf{x} \leftarrow \omega)$ and $L(\mathbf{x} \leftarrow \mathbf{y})$ for incident radiance that arrives at a point $\mathbf{x}$ from direction ω and other point $\mathbf{y}$, respectively, and similarly $L(\mathbf{x} \rightarrow \ldots)$ is used to denote emitted or outgoing radiance. The *radiance invariance law* states that radiance does not change along a ray in vacuum, i.e., $L(\mathbf{x} \leftarrow \mathbf{y}) = L(\mathbf{y} \rightarrow \mathbf{x})$, which makes it a convenient quantity to work with. The relationship between irradiance and radiance at a point, $\mathbf{x}$, on a surface is given by:

$$E(\mathbf{x}) = \int_{\Omega} L(\mathbf{x} \leftarrow \omega) \cos\theta\, d\omega, \tag{4}$$

where Ω is the visible hemisphere, i.e., the irradiance is the integral over all incident radiance adjusted for projected area.

2.2 The Rendering Equation

The rendering equation [65] is based on the law of conservation of energy. In its basic form, it describes the outgoing radiance, $L(\mathbf{x} \to \omega_o)$, leaving the point $\mathbf{x}$ on a surface in direction ω_o, as the sum of the radiance emitted, L_e, by the surface itself and the total radiance reflected, L_r, towards ω_o. The total reflected radiance is computed by integrating over the visible hemisphere, Ω. We have:

$$L(\mathbf{x} \to \omega_o) = L_e(\mathbf{x} \to \omega_o) + \underbrace{\int_\Omega L(\mathbf{x} \leftarrow \omega_i) f_r(\mathbf{x}, \omega_i, \omega_o) \cos\theta_i \, d\omega_i}_{L_r(\mathbf{x} \to \omega_o)}, \tag{5}$$

where $L(\mathbf{x} \leftarrow \omega_i)$ is the incident radiance arriving at the point $\mathbf{x}$ from direction ω_i. The incident radiance is weighted by the cosine of the angle between the surface normal and the incident direction, $\cos\theta_i$, to take the projected area into account. The f_r term is the *bidirectional reflectance distribution function* (BRDF), which describes the reflectance at $\mathbf{x}$ between incident and outgoing directions. The properties of the BRDF will be further discussed in Section 2.3. Note that the BRDF is dimensionless with the unit sr^{-1}, so the result of the integral is the reflected radiance with unit $\mathrm{W{\cdot}m^{-2}{\cdot}sr^{-1}}$, as expected. In practice, most surfaces are not emitting light by themselves, so L_e is generally zero, except at a few light sources in the scene. In Section 2.4, we will describe several types of light sources commonly used in computer rendering.

The rendering equation is challenging because the radiance, L, appears both inside the integral and as a quantity to solve for on the left-hand side. The reason for this is that the outgoing radiance from each point affects the incident radiance at all other places visible from that point, i.e., we have a huge system of integrals. This type of “recursive” integral is formally known as a Fredholm integral equation of the second kind. Analytic solutions are impossible in all but the most trivial cases. In Section 3, we will discuss numerical solutions based on Monte Carlo methods, which is the focus of this dissertation.

It should be noted that the idea behind the rendering equation was not fundamentally new when introduced by Kajiya in 1986 [65]; the problem had been extensively studied in the field of radiative heat transfer [60], but it was now presented in a form suitable for computer graphics.

2.2.1 Limitations and Extensions

There are a number of well-known limitations of the rendering equation expressed in the form of Equation 5. First, it only attempts to model geometric optics, which treats light as rays that travel in straight lines and only bend due to reflection or refraction. As such, it does not take the phase and polarization of light into account, and it does not model diffraction or interference effects.

Second, the wavelength, λ, of light is omitted. Instead, the radiance, L, is usually assumed to be a vector-valued quantity, measuring the radiance of a ray of light at a number of distinct wavelengths (i.e., three in the RGB model). For spectral rendering of dispersion effects, the wavelength can be added as an explicit parameter. In the most general case, both the wavelengths of incident and outgoing radiance, λ_i and λ_o, have to be included, and the integral must be extended to integrate over λ_i in addition to ω_i. The latter approach allows simulation of fluorescence, but adds one extra dimension to an already difficult integration problem.

Similarly, the rendering equation expresses the radiance at a single instance in time, omitting the time parameter, t. By adding time as an extra dimension, correct *motion blur* can be computed, i.e., the simulation of a finite, non-zero shutter time of the camera. This may include dynamic geometry, in which case L changes, time-varying emission in L_e, and even time-varying BRDFs [137]. Furthermore, by separating the notion of time for incident and outgoing radiance, t_i and t_o, and integrating over t_i, materials that exhibit *phosphorescence* [43] can be simulated. In this case, unlike with fluorescence, a material re-emits absorbed light at a much later time (e.g., as used in glow-in-the-dark toys).

The basic formulation of the rendering equation only handles opaque surfaces, but it can easily be extended to transparent materials by including a *bidirectional transmittance distribution function* (BTDF) and integrating over the entire sphere.[3] However, in either case, the incident illumination is assumed to be reflected/refracted around the same point. This is not enough for accurate rendering of *translucent* materials, which require simulation of *subsurface scattering*. In a translucent material, light enters through the surface and is scattered within the material, before being re-emitted at a potentially different location. There are many examples of visibly translucent materials, e.g., marble, wax, skin, and many plastics. To simulate subsurface scattering, the BRDF is replaced by a *bidirectional scattering surface reflectance distribution function* (BSSRDF) [99]. In the general case, the BSSRDF is an 8-dimensional (8D) function parameterized over the positions and directions of both incident and outgoing light, which makes translucency a difficult problem. We note that several advanced BSSRDF models have been developed for approximation, acquisition, and representation of BSSRDFs [31, 32, 54, 105], but we will not discuss the topic further.

Finally, the rendering equation assumes that light travels in straight paths from one surface to another, without taking into consideration the medium it travels through. This assumption is really only true in vacuum, as all other media contain particles that interact with the light. However, it is a reasonably good approximation in the typical case of light traveling short distances in clean air. To handle other cases, e.g., light scattering due to smoke, fog, water, or dusty air, we must simulate how the radiance changes as light travels through these so-called *participating media*. This is a large topic in itself, and we will only very briefly touch on the subject. The participating media is usually modeled as a statistical distribution of independent microscopical particles, which gives rise to four types of light interactions (absorption, emission, in- and out-scattering). Based on these, a *volume rendering*

[3] In many cases, a BTDF representing simple mirror transmittance is used, which means we add the radiance arriving along the refracted view direction, i.e., w_o refracted about the surface normal.

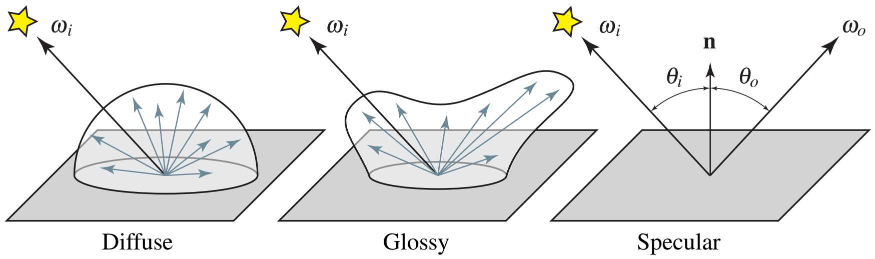


***Figure 6:** The appearance of different materials is often classified by their ability to reflect light, ranging from perfectly diffuse to ideal specular reflection. Most real-world materials lie somewhere in between, exhibiting glossy reflection.*

equation can be formulated that is similar to the rendering equation, but of higher dimensionality. For an overview of rendering techniques for participating media, we refer to the survey by Cerezo et al. [19]. In particular for Monte Carlo methods, the PhD dissertation by Jarosz [63] also presents a good summary.

2.3 Reflection Models

The visual appearance of a material is determined by the ability of the surface to reflect light in different directions. The amount of light reflected at different wavelengths decides its color, i.e., the ability to absorb light, and the directions of reflection are decided by the surface properties. When a surface is microscopically very rough, it scatters light in wide range of directions and appears dull or matte. In this case, we speak of *diffuse reflection*. On the other hand, an ideal smooth surface reflects all light according to *specular reflection*, i.e., mirror-like reflection. Most real-world surfaces lie in between these two extremes, and exhibit varying degrees of *glossy reflection*. The different types of reflection are illustrated in Figure 6.

Perfect specular reflection is governed by the law of reflection, which states that the angles of incident and outgoing light to the surface normal are equal, i.e., $\theta_i = \theta_o$, and all three vectors are coplanar ($\phi_o = \phi_i + \pi$). The physical substrate and the angle of incidence also often have a great effect on the reflectivity of a surface.

Definition Formally, the appearance of an opaque surface is described by the *bidirectional reflectance distribution function* (BRDF), $f_r(\mathbf{x}, \omega_i, \omega_o)$, which is defined as follows [99]:

$$f_r(\mathbf{x}, \omega_i, \omega_o) = \frac{dL(\mathbf{x} \to \omega_o)}{dE(\mathbf{x} \leftarrow \omega_i)} \quad [\mathrm{sr}^{-1}], \tag{6}$$

i.e., the BRDF defines the relation of outgoing radiance in direction ω_o, to irradiance arriving from ω_i, at a point, $\mathbf{x}$, on the surface.

2.3.1 Physical Properties

Physically-based BRDFs are naturally positive, as a surface cannot absorb more light than falls on it:[4]

$$f_r(\mathbf{x}, \omega_i, \omega_o) \;\geq\; 0. \tag{7}$$

The BRDF must also obey the law of *energy conservation*, which leads to the following condition:

$$\int_\Omega f_r(\mathbf{x}, \omega_o, \omega_i) \cos\theta_o \, d\omega_o \;\leq\; 1, \qquad \forall \omega_i. \tag{8}$$

Finally, it obeys the *Helmholtz reciprocity* principle [21], which follows from the laws of classical electromagnetism, and states that:

$$f_r(\mathbf{x}, \omega_i, \omega_o) = f_r(\mathbf{x}, \omega_o, \omega_i), \tag{9}$$

i.e., the incident and outgoing directions can be swapped with no effect on the reflectivity. This is important in practice, as it allows computer graphics algorithms to follow rays in either direction, i.e., from the light towards the camera or in the reverse direction, without changing how the BRDF is represented.

Most natural materials have spatially-varying reflectance, in which case f_r is a six-dimensional (6D) function; two dimensions describe the position on the surface, $\mathbf{x}$, and each of the incident and outgoing directions are described by two spherical coordinates, $\omega = (\theta, \phi)$. In computer graphics, the spatially varying parameters controlling the BRDF model at any given point, are usually taken from texture maps applied to the surface and/or are procedurally computed.

Looking at a particular point, $\mathbf{x}$, the general BRDF is a four-dimensional (4D) function. When the reflectance depends on the absolute orientation of the surface, as is the case for materials like brushed metal, satin, or hair, the BRDF is *anisotropic*. In a large sub-class of materials, the reflectance does not significantly depend on the absolute orientation, but only on the relative angle between the incident and outgoing directions. These materials can be modeled by simpler *isotropic* BRDFs, which are three-dimensional functions (at any given surface point):

$$f_r(\theta_i, \theta_o, |\phi_i - \phi_o|). \tag{10}$$

In the following, we will take a brief look at some of the commonly used BRDF models in computer graphics.

2.3.2 Examples of BRDFs

A wide range of BRDF models have been developed in computer graphics and other fields to simulate the visual appearance of materials. The problem of creating

[4]However, it should be noted that in non-physically based rendering, it is common to have BRDFs with negative reflectance to "subtract" light from selected areas of a scene.

a physically-based BRDF is not trivial, as there are many competing goals; it is often desirable that the BRDF model is easily controllable using a small set of intuitive parameters, that it is computationally efficient, and able to reproduce a wide range of materials, while obeying the physical laws detailed above.

Many reflection models include a term to model ideal diffuse reflection. A surface that only exhibits diffuse reflection is described by the *Lambertian* BRDF:

$$f_r(\omega_i, \omega_o) = \frac{\rho_d}{\pi}, \tag{11}$$

where $\rho_d \in [0,1]$ is the diffuse reflection coefficient or *albedo*, describing the fraction of light that is diffusely reflected. The $1/\pi$ factor normalizes the BRDF so that the integral in Equation 8 is exactly one when $\rho_d = 1$. Some real-world materials are nearly perfectly Lambertian, e.g., matte white paper, or perhaps more interestingly, the lunar surface. Note that since the Lambertian BRDF does not depend on the incident or outgoing directions, the surface will appear equally bright from all viewing angles; indeed the moon does not look darker towards its edges.

Empirically Based Models Many of the early reflection models were *empirically based*, designed to simulate the visual look or data measured from real surfaces. However, they did not always obey the physical laws. A good example is the *Phong* reflection model [108], which has later been modified to be physically plausible [73]. It models isotropic reflectance as a combination of perfectly diffuse reflection with a simple specular lobe, which gives a *plastic*-like look. The model is still popular due to its simplicity. A later example is the *Ward* anisotropic BRDF model [152] (also later improved [34, 96]), which was created to fit measured reflectance data. It similarly models a combination of diffuse and specular reflection, but the specular component is represented as an anisotropic Gaussian lobe. The Ward model is well-suited to represent *metallic* surfaces, and in particular brushed metals, as it supports anisotropy. We include its mathematical expression here to give an idea of the operations involved in evaluating a typical BRDF. The model is usually expressed on the form [150]:

$$f_r(\omega_i, \omega_o) = \frac{\rho_d}{\pi} + \frac{\rho_s}{4\pi\alpha_x\alpha_y\sqrt{(\omega_i\cdot\mathbf{n})(\omega_o\cdot\mathbf{n})}}\, e^{-\frac{((\mathbf{h}\cdot\mathbf{x})/\alpha_x)^2+((\mathbf{h}\cdot\mathbf{y})/\alpha_y)^2}{(\mathbf{h}\cdot\mathbf{n})^2}}, \tag{12}$$

where $\mathbf{n}$ is the local surface normal, $\mathbf{x}$ and $\mathbf{y}$ are two orthogonal basis vectors in the surface's tangent plane, and $\mathbf{h}$ is the *halfway vector* representing the normalized direction between ω_i and ω_o:

$$\mathbf{h} = \frac{\omega_i + \omega_o}{||\omega_i + \omega_o||}. \tag{13}$$

The reflectance is controlled by four intuitive parameters, the diffuse and specular reflection coefficients, (ρ_d, ρ_s), and two *roughness* parameters, (α_x, α_y), deciding the shape of the specular lobe (where smaller values give sharper highlights).

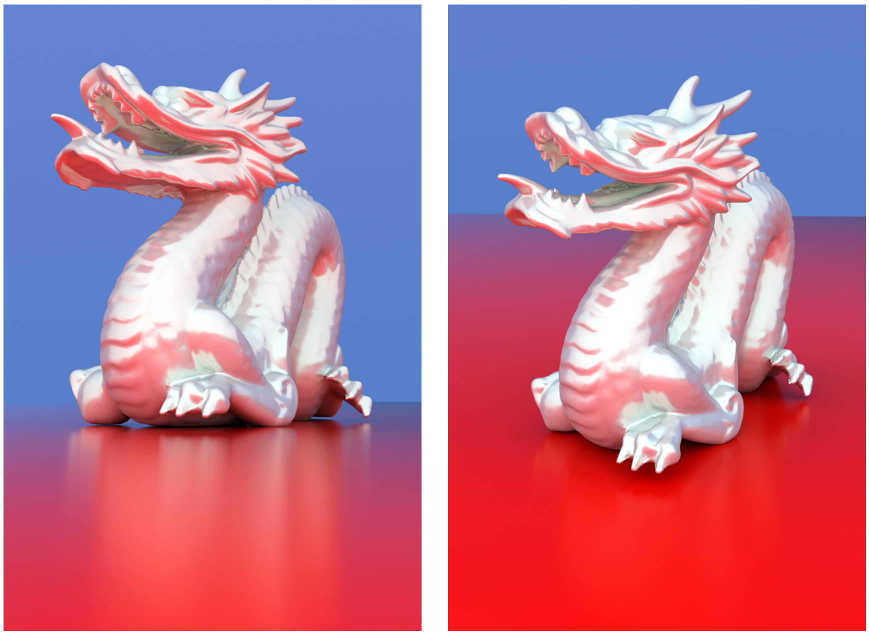


Figure 7: *Examples of two renderings using the same physically-based BRDF based on a microfacet model. The Fresnel term approximates the increasing reflectivity towards grazing angles, as shown in the image on the left. The dragon model is courtesy of Stanford University Computer Graphics Laboratory.*

The ease of control of the Ward model is a great advantage over more complicated models. However, in order to address some of its shortcomings, Ashikmin and Shirley [6] presented a more advanced empirical anisotropic reflection model, which gives good results for a variety of materials.

Physically Based Models Another important class of BRDFs are *physically-based* models, which are derived from physical rather than empirical models of the light–surface interactions. A lot of inspiration has been gained from reflection models in other fields, such as the Beckmann distribution for scattering of electromagnetic waves. The earliest example in computer graphics is probably the *Blinn-Phong* [11] model, which is similar to the Phong model. The difference is that the specular lobe is based on evaluating a normal distribution function (ndf) centered around the halfway vector. The motivation for this is that surface roughness can be modeled as a random distribution of *microfacets*, i.e., tiny fragments of a surface, where the reflection between ω_i and ω_o is strongest for facets pointing towards the halfway vector. (The paper by Edwards et al. [37] has a good discussion on the halfway vector.)

Following along this line, the *Cook-Torrance* model [25], inspired by the Torrance-Sparrow model used in physics [145], was the first BRDF in computer graphics to explicitly model the interaction and shadowing between microfacets. It was designed to simulate metallic surfaces, and includes a so-called *Fresnel* term [14] to approximate the color shifts and increasing reflectance towards grazing angles that many materials exhibit. See Figure 7 for an example. Several more accurate microfacet-based models have followed [7, 109], including some targeting non-glossy materials. For example, the Oren-Nayar model [102] uses diffusely reflecting microfacets rather than mirrors. This gives a more accurate imitation of rough diffuse surfaces, such as concrete, plaster, sand, and clay, than the ideal Lambertian. There are even physically-based BRDFs based on simulating wave optics for certain classes of materials [58, 134].

However, as physically-based reflection models get increasingly complex, they tend to require a larger set of parameters. Many of these are often unintuitive, which can make it difficult to manually tweak the appearance. The computational complexity of evaluating the more advanced physically-based models can also be very high. As always in computer graphics, there is a tradeoff between rendering times and accuracy. There have been attempts to simplify and improve the efficiency of physically-based models using mathematical approximations, e.g., the work by Schlick [124]. However, many of the simpler empirical models continue to be widely used, despite a weaker theoretical basis.

Data Driven Models As an alternative to trying to create physical or empirical models of the reflectance, one can take a *data-driven* approach and measure the reflectance of real materials. For this purpose a *gonioreflectometer* is used, i.e., a device capable of measuring reflectance at different angles. Although the design varies, the measurement setup typically involves a light source and sensor (e.g., camera), whose relative positions can be varied with respect to a material sample in an automated manner. As a general BRDF is a four-dimensional function, or even six-dimensional for spatially-varying materials, the process can be time-consuming. It also faces difficulties such as radiometric calibration, mechanical accuracy, and sensor noise. Hence, accurate reflectance data is only available commercially, or from a few publicly available datasets, such as the CUReT database [26] and the MERL BRDF database [88]. In our research, we have used the latter in Paper I and Paper IV.

In order to use these high-dimensional datasets for rendering, it is often practical to use mathematical fitting techniques to fit a simpler representation to the measured data. For example, Lafortune et al. [75] approximate measured or simulated reflectance data by fitting multiple generalized Phong lobes. This representation has an intuitive meaning and can be modified, albeit with some difficulty. Matusik et al. [88] use principal component analysis (PCA) to show that many BRDFs can be represented using a relatively small set of principal components, or even as a linear combination of BRDFs [89]. Many other approaches directly project the reflectance data onto an orthogonal basis, and compress/approximate it by storing only a small set of basis coefficients. Examples of bases used in these techniques include spherical harmonics (SH) [131, 158], Zernike polynomials [66], and wavelets [79, 127]. It should be mentioned that such compressed BRDF representations have been very useful also outside the field of photorealistic rendering, and they are a necessity in real-time relighting methods based on precomputed radiance transfer (PRT) [132]. In fact, the BRDF representation used in Paper I was adopted from a wavelet-based PRT technique [97].

To conclude, the modeling and representation of reflectance is an interesting topic because it is an essential part of simulating light transport; the BRDF directly controls the visual appearance of a surface under varying lighting conditions. The problem is also very complex, and there are endless possibilities to refine the models, e.g., to simulate more accurate surface detail, texture, reflectance, or subsurface scattering. Current research focuses largely on developing specialized models for particular classes of materials, such as metallic car paint [123]. The SIGGRAPH course by Dorsey and Rushmeier [33] gives a good overview.

2.4 Light Sources

In order to form an image and create meaningful pictures, light must be added to the computer model of the world, just like the photographer needs to pay careful attention to the lighting of her scene. Without light, the images would be black. In this section, we will discuss some common types of *light sources*, and how the rendering equation can be rewritten to explicitly take these into account.

We have seen how the rendering equation in Equation 5 models the propagation of light in the three-dimensional world using a recursive integral formulation. On this form, light is injected into the simulation by surfaces that emit light, i.e., surfaces where $L_e(\mathbf{x} \to \omega_o) > 0$. Technically speaking, any surface can be a light source and Equation 5 is sufficient to solve the light transport problem. In reality, however, usually only a few surfaces emit light, while the majority do not. Therefore, it is often more convenient to express the rendering equation as an area integral, in order to integrate the light that arrives directly from these surfaces.

2.4.1 Area Formulation of the Rendering Equation

Equation 5 expresses the outgoing radiance, $L(\mathbf{x} \to \omega_o)$, as a sum of emitted light, L_e, and reflected light, L_r. The reflected light is computed by integrating over the visible hemisphere at $\mathbf{x}$. Therefore, this is called the *hemispherical formulation* of the rendering equation. As we will see, it is often more convenient to integrate over other surfaces in the scene rather than over the hemisphere, which leads to the *area formulation* of the equation. We express the reflected light as an integral over all other points, $\mathbf{y}$, on the surfaces, A, in the scene.

First, note that the surface at $\mathbf{y}$ has a normal $\mathbf{n}_\mathbf{y}$ and differential area $dA_\mathbf{y}$. The hemispherical form integrates over solid angle, and the relationship between differential solid angle, $d\omega_i$, and differential area, $dA_\mathbf{y}$, is found by considering the

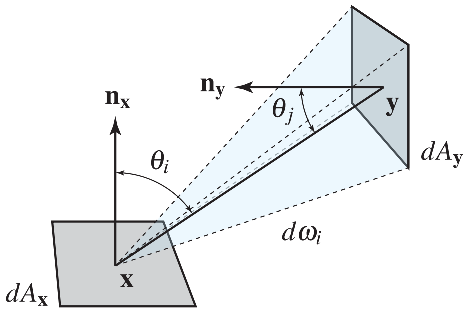


Figure 8: *The geometry term in the area formulation of the rendering equation describes the relation of energy transfer between two differential surfaces, $dA_\mathbf{x}$ and $dA_\mathbf{y}$, based on their relative angles and distance. As seen from the point* $\mathbf{x}$*, the solid angle, $d\omega_i$, that the surface $dA_\mathbf{y}$ subtends is equal to its projected area,* $\cos\theta_j dA_\mathbf{y}$*, divided by the squared distance,* $||\mathbf{x}-\mathbf{y}||^2$.

solid angle subtended by $dA_\mathbf{y}$ at $\mathbf{x}$. This is illustrated in Figure 8, and is given by:

$$d\omega_i = \frac{\cos\theta_j\, dA_\mathbf{y}}{||\mathbf{x}-\mathbf{y}||^2}, \qquad \text{where} \quad \cos\theta_j = (\mathbf{n_y}\cdot -\omega_i). \tag{14}$$

The cosine term comes from the projected area being $dA_\mathbf{y}^\perp = \cos\theta_j\, dA$, which at a distance $r = ||\mathbf{x}-\mathbf{y}||$, corresponds to the solid angle given in Equation 14. We are now ready to formulate the rendering equation as an integral over all surfaces, A, in the scene, as follows:

$$\begin{aligned} L(\mathbf{x}\to\omega_o) &= L_e(\mathbf{x}\to\omega_o) + \int_{\mathbf{y}\in A} L(\mathbf{x}\leftarrow\mathbf{y}) f_r(\mathbf{x},\omega_i,\omega_o) G(\mathbf{x},\mathbf{y}) dA_\mathbf{y}, \\ & \qquad \text{where} \quad G(\mathbf{x},\mathbf{y}) = \frac{\cos\theta_i \cos\theta_j}{||\mathbf{x}-\mathbf{y}||^2}, \end{aligned} \tag{15}$$

is called the *geometry term*, as it takes the relative geometries of the differential areas at $\mathbf{x}$ and $\mathbf{y}$ into account. The $L(\mathbf{x}\leftarrow\mathbf{y})$ term denotes radiance that *arrives* at $\mathbf{x}$ from the surface at $\mathbf{y}$. Note that this requires a clear line of sight between the two points, as otherwise no light will arrive and $L(\mathbf{x}\leftarrow\mathbf{y})$ is zero. This relationship is often made explicit by introducing a binary *visibility* function, $V(\mathbf{x},\mathbf{y})$, which describes the mutual visibility of two points:

$$V(\mathbf{x},\mathbf{y}) = \begin{cases} 1 & \text{if } \mathbf{x} \text{ and } \mathbf{y} \text{ are mutually visible,} \\ 0 & \text{otherwise.} \end{cases} \tag{16}$$

Under the general assumption of light transport in vacuum, where no attenuation or scattering occurs due to participating media, we have:

$$L(\mathbf{x}\leftarrow\mathbf{y}) = L(\mathbf{y}\to\mathbf{x})V(\mathbf{x},\mathbf{y}). \tag{17}$$

2.4.2 Direct and Indirect Illumination

The area formulation of the rendering equation is useful, since it allows us to directly integrate over the surfaces of light sources. Intuitively, by doing so, we can compute the light that arrives directly from the light sources, i.e., the *direct illumination*, separately from the light that arrives due to reflection from other surfaces, the so-called *indirect illumination*. The outgoing radiance is given as the sum of self-emitted radiance, and the direct and indirect illumination reflected in the outgoing direction:

$$L(\mathbf{x} \to \omega_o) = L_e(\mathbf{x} \to \omega_o) + L_{direct}(\mathbf{x} \to \omega_o) + L_{indirect}(\mathbf{x} \to \omega_o). \tag{18}$$

As we will see in Section 3, the two terms L_{direct} and $L_{indirect}$ are usually computed using different techniques for efficiency reasons. The indirect illumination is the result of light bouncing off other surfaces, possibly an infinite number of times. The incident radiance arrives from all directions on the hemisphere, and a hemispherical formulation is usually appropriate:

$$L_{indirect}(\mathbf{x} \to \omega_o) = \int_{\Omega} L_r(\mathbf{x} \leftarrow \omega_i) f_r(\mathbf{x}, \omega_i, \omega_o) \cos\theta_i \, d\omega_i, \tag{19}$$

where L_r is reflected radiance arriving at $\mathbf{x}$, i.e., excluding light emitted in the last bounce, computed using the integral on the right-hand side of Equation 5 at each visible point over the hemisphere. Note that the indirect illumination is thus defined on a recursive integral form.

For the direct illumination, the most convenient formulation depends on the type of light sources used. In the general case, the direct illumination can be expressed as an integral over all light-emitting surfaces, $A' \subseteq A$, using the area formulation, as follows:

$$L_{direct}(\mathbf{x} \to \omega_o) = \int_{\mathbf{y} \in A'} L_e(\mathbf{y} \to \mathbf{x}) f_r(\mathbf{x}, \omega_i, \omega_o) V(\mathbf{x}, \mathbf{y}) G(\mathbf{x}, \mathbf{y}) dA'_{\mathbf{y}}, \tag{20}$$

where $L_e(\mathbf{y} \to \mathbf{x})$ is the radiance emitted by the surface at $\mathbf{y}$ towards the point $\mathbf{x}$.

Note that the direct illumination, although described by a complex integral, is not recursive in its nature. The goal of this dissertation is largely to develop efficient ways of computing the direct illumination, L_{direct}, under various conditions. In the following, we will discuss some examples of light sources.

2.4.3 Examples of Light Sources

There are many different types of light sources, and the best choice for a given application is usually a compromise between the desire for accuracy and the rendering efficiency. The most common variants are described below.

Point Light The simplest model of a light source is the *point light*, i.e., a point-like light source that emits light of uniform intensity in all directions. More formally, the point light is said to have an *omnidirectional* light distribution. Note that no true point light exists in the real world, as a physical light source always has some extent in space. Nevertheless, it can be a good approximation if the size of the light source is small relative to the rest of the scene. Point lights are also commonly used to approximate other light emitters, e.g., by placing many virtual point lights (VPLs) over a surface to approximate the light it emits or reflects.

To compute the direct illumination from a point light, there is no need to integrate over incident radiance, as the point light subtends an infinitely small solid angle. Following the radiometric definitions in Section 2.1.2, we find that:

$$L_{direct}(\mathbf{x} \to \omega_o) = \frac{\Phi}{4\pi||\mathbf{x}-\mathbf{y}||^2} f_r(\mathbf{x}, \omega_i, \omega_o) V(\mathbf{x}, \mathbf{y}) \cos\theta_i, \tag{21}$$

where Φ is the flux of the point light in watts.

While most point lights are omnidirectional, variations that have non-uniform light distributions are sometimes used in specialized applications. For example, a directional light distribution may be specified by so-called goniometric diagrams, available from light manufacturers. Additionally, in some rendering algorithms, indirect illumination is approximated using a cloud of virtual point lights, which sometimes have non-omnidirectional properties.

Spotlight A *spotlight* is a type of light source that is closely related to the point light. Instead of radiating in all directions, the emitted light is focused in a cone centered around the principal direction of the spotlight. The distribution of light often varies within the cone, following a smooth falloff towards the edges and being completely cut off outside the cone.

Spotlights are convenient tools for the artists creating the three-dimensional worlds, as they allow more control than point lights and the light can be focused on particular objects. From the rendering system's point of view, the spotlight emits light from a single point in space similar to a point light, but it is often possible to take advantage of the fact that the intensity is zero outside the cutoff angle of the cone.

Area Light The *area light* is a more general form of light source, which emits light over the entire surface of a shape. The light emitting shape is usually a geometric primitive, such as a triangle, quadrilateral, or disc. Both the spatial and directional light distributions can be controlled, and by combining multiple simpler area lights, a more complex light source can be created. We have already seen how the direct illumination from area lights can be computed as an area integral using Equation 20. When a scene contains many different light sources, their contributions are usually computed separately and the results added together.

Directional Light Finally, there are a few important types of light sources that are not localized to any position in the scene. A *directional light* is an imaginary light source that illuminates all objects equally from a given direction. It can thus be seen as an infinite area light that is infinitely far away. For example, sunlight in an outdoor environment is often modeled using a directional light source, as the rays arriving at Earth are nearly parallel. The directional light is either visible or occluded, as seen from a point $\mathbf{x}$. This is described by a directional visibility function, defined as follows:

$$V(\mathbf{x}, \omega_i) = \begin{cases} 1 & \text{if clear line of sight from } \mathbf{x} \text{ in direction } \omega_i, \\ 0 & \text{otherwise.} \end{cases} \tag{22}$$

The light emitted from a directional light pointing in direction $-\omega_i$ with a radiant exitance of M_e [W·m^{-2}], arrives unmodified at $\mathbf{x}$ if there is a clear line of sight, since all emitted light is assumed to travel in parallel rays. The irradiance is thus $E(\mathbf{x}) = M_e V(\mathbf{x}, \omega_i) \cos\theta_i$ due to the projected area of the receiver, which gives the direct illumination as:

$$L_{direct}(\mathbf{x} \to \omega_o) = M_e f_r(\mathbf{x}, \omega_i, \omega_o) V(\mathbf{x}, \omega_i) \cos\theta_i. \tag{23}$$

Environment Map Last, we have *environment mapping*, which is an image-based lighting (IBL) technique, where one or more images are used to specify the illumination in each direction over the sphere. Similar to the directional light, the light from an environment map is assumed to arrive from infinitely far away and it illuminates all objects equally, although the angular distribution varies. Originally, the technique was mostly used to provide sharp reflections of an environment or background [13, 47, 93]. It was later developed into an accurate method of lighting a scene using high-dynamic range photographs of real environments [27]. In this context, the environment map is usually called a *light probe*, as it measures the light in each direction. Environment mapping is a very useful tool in photorealistic rendering, as it provides a convenient way to approximate the direct lighting in a real scene. The technique is, for instance, widely used for rendering special effects in feature films. By capturing one or more light probes at a real set, it is relatively easy to achieve lighting conditions in the rendered images that match those of shots filmed at the set. This is a critical component for seamless integration of computer-generated imagery in real world footage.

As the environment map is assumed to be infinitely far away, the incident illumination does not depend on the position in the scene, but only on the direction. Therefore, the direct illumination under environment map lighting is given by:

$$L_{direct}(\mathbf{x} \to \omega_o) = \int_\Omega L_{env}(\omega_i) f_r(\mathbf{x}, \omega_i, \omega_o) V(\mathbf{x}, \omega_i) \cos\theta_i d\omega_i, \tag{24}$$

where $L_{env}(\omega_i)$ is the radiance arriving from the environment map, as seen in direction ω_i. The integral is defined over the entire visible hemisphere, as the lighting from all directions needs to be taken into account.

The computation of direct illumination in Equation 24 is non-trivial, as the integral is taken over the entire hemisphere and includes both a BRDF, f_r, and a visibility function, V, both of which can be complicated high-frequency functions depending on the materials and scene geometry used. A large part of this dissertation is devoted to finding efficient techniques for solving this integral. In particular, we have developed several novel Monte Carlo techniques aimed at evaluating the direct illumination under environment map lighting, using a minimal set of well-chosen random samples. In Section 3 and 4, we will look at the theory and our contributions in more detail.

2.5 Camera Models

In the previous sections, we have defined all the necessary components for simulating light transport in a three-dimensional world. Using the rendering equation, we can solve for the radiance at any point and direction in space, taking the light sources, material models, and geometry of the scene into account. However, in order to create a two-dimensional image, we must additionally model and simulate the optical system of a virtual *camera* placed in the scene. The image is formed by the camera lens projecting incoming light onto an *image plane*, where an image sensor or photographic film is *exposed* over the duration of the shutter time. In this section, I will give a brief overview of the process of image formation, and discuss some of the approximations commonly employed in computer graphics.

2.5.1 Exposure

In a real camera, the sensor or photographic film records the exposure to light at each point on the image plane. In both cases, the image that is formed is a function of the *radiant exposure*, H, i.e., total energy per unit area that arrived at each point, $\mathbf{u} = (u_x, u_y)$, on the image plane. The exposure is computed by integrating irradiance, $E(\mathbf{u})$, over time, as follows:

$$H(\mathbf{u}) = \int_t E(\mathbf{u}) S(\mathbf{u},t)\,dt \qquad [\mathrm{J} \cdot \mathrm{m}^{-2}], \tag{25}$$

where $S(\mathbf{u},t)$ is the shutter function, which describes the time interval where the shutter is open, as seen from the point $\mathbf{u}$. If the irradiance on the image plane changes over time, e.g., by objects or the camera moving, the result is an effect called *motion blur*. This appears as blurry motion trails along the path of objects. In photography, motion blur is often (but not always) an unwanted effect and short shutter times are used to avoid it. In film, however, motion blur enhances the perception of motion, and without it, the moving pictures appear stuttering.

The irradiance, $E(\mathbf{u})$, is itself computed by integrating the incident radiance on the image plane over solid angle (see examples below). In computer graphics, it is often assumed that the shutter opens and closes instantaneously:

$$S(\mathbf{u},t) = \begin{cases} 1 & t_0 \leq t \leq t_1, \\ 0 & \text{otherwise,} \end{cases} \tag{26}$$

where $[t_0,t_1]$ is the time interval the shutter stays open. In reality, a camera shutter is usually implemented as a mechanical device with shutter blades that travel across the image plane at a certain speed, or as an electro-optical device that has a certain switching time between fully opaque and transparent. Equation 26 ignores these unwanted side effects of the physical construction, although specialized applications may need to model them explicitly.

Traditional photographic film is coated with crystals or "*grains*" of light-sensitive silver halide salts, which are activated by a photochemical process and later chemically developed into a negative image (i.e., darker where more light arrived). The density of the developed film is typically logarithmic to the exposure, which means film can record a wide dynamic range, i.e., preserve detail in both dark and bright areas. On the other hand, the image sensor in digital cameras, which is typically based on CCD or CMOS technology, transforms light energy quanta (photons) into an electrical charge. This process typically follows a nearly linear response curve, which traditionally gave a narrower dynamic range. However, the accuracy and dynamic range of image sensors have improved to a point where they often surpass those of photographic film.

For computer-generated images, the dynamic range is only limited by the numerical range of the data format used for computing and storing the image. For photorealistic rendering, a *high dynamic range* (HDR) format is usually used, where the radiometric quantities are represented using (at least) 16-bit or more often 32-bit floating-point values per color component. Before use, the rendered high dynamic range image has to be mapped to the (limited) dynamic range of the output device or medium. The process is known as tone mapping, and is an important step before the image can be displayed or used for other purposes. Similarly, rendered images do not exhibit traditional film grain, which is therefore often simulated by a random process and added to the final image in order to match the look of photographed images or footage.

2.5.2 Pinhole Camera

Before the light arrives at the image plane in a real camera, it passes through the camera lens – an optical system designed to project an accurate picture of the world onto the image plane. The simplest possible "lens" system is the *camera obscura* or pinhole camera, which consists of a box with a small hole in one side and no optical lenses at all. On the opposite side, an image is projected (upside-down) on a screen or photographic film. The image gets sharper the smaller the hole is because the light gets more focused, but at a too small opening, diffraction effects limit the sharpness. The image also gets dimmer at smaller openings, since less light reaches the image plane.

Formally, the irradiance at the image plane, $E(\mathbf{u})$, is given by integrating the incident radiance over the pinhole opening, D. Assuming the opening is parallel to the image plane, we arrive at the following expression:

$$E(\mathbf{u}) = \int_{\mathbf{v}\in D} L(\mathbf{u} \leftarrow \mathbf{v}) \frac{\cos\theta}{||\mathbf{v}-\mathbf{u}||^2} \cos\theta \, dA, \tag{27}$$

where the first cosine factor accounts for the solid angle subtended by dA (Equation 14), and the second for the projected area on the image plane (Equation 1). If the distance from the pinhole opening to the image plane is d along the optical axis, the expression can be rewritten:

$$E(\mathbf{u}) = \frac{1}{d^2} \int_{\mathbf{v}\in D} L(\mathbf{u} \leftarrow \mathbf{v}) \cos^4\theta \, dA, \tag{28}$$

since $\cos^2\theta = d^2/||\mathbf{v}-\mathbf{u}||^2$ due to standard trigonometry.

For a pinhole camera, the solid angle subtended by the opening is very small and the integral can safely be approximated as [67]:

$$E(\mathbf{u}) \approx L(\mathbf{u}, \omega) \frac{A}{d^2} \cos^4\theta, \tag{29}$$

where A is the area of the pinhole opening, and L is the radiance arriving from its center in direction ω. The $\cos^4\theta$ factor causes light falloff, or *vignetting*, towards the edges of the image, which is clearly visible in photographs taken with real pinhole cameras.

In an *ideal* pinhole camera model, the opening is assumed to be infinitely small, i.e., all light passes through a single point. Although not physically realizable, this model is commonly used in computer graphics due to its simplicity. Since all light is projected through a single point, the entire image will be sharp and in focus. Additionally, the $\cos^4\theta$ falloff factor is usually ignored [24], so we have:

$$H(\mathbf{u}) \propto L(\mathbf{u}, \omega), \tag{30}$$

which leads to the common conception that a camera "sees" radiance. Note that for an ideal pinhole camera, we cannot speak in physical terms about the exposure, since an infinitely small opening would allow no light to pass.

2.5.3 Thin Lens Model

To get around the limitations of the camera obscura, optical lenses can be inserted in the light path in order to focus light from different directions onto the image plane. Light that enters a lens is refracted due to the index of refraction, η, being different between air (η close to 1) and the lens material (e.g., optical glass or plastic, typically with $\eta = 1.5 \ldots 1.9$). As light exits on the other side, it is again refracted. The angles between incident and refracted light are governed by Snell's law: $\eta_1 \sin\theta_1 = \eta_2 \sin\theta_2$, where θ_i is measured with respect to the normal on each

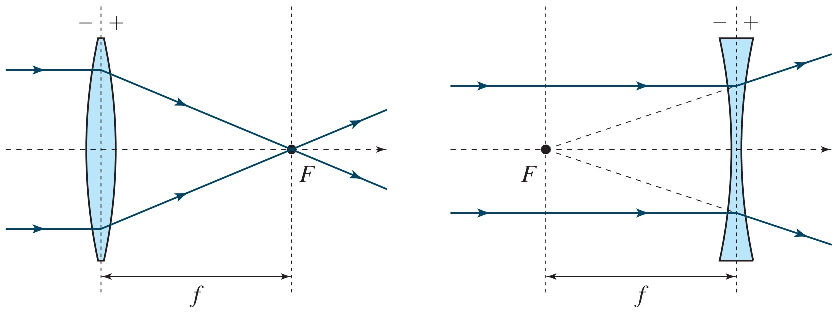


Figure 9: *Parallel light that reaches an optical lens is either converged (left) or diverged (right). The outgoing rays converge at the (ideal) lens' focal point, F, which lies either on the positive or negative side of the lens. The focal length, f, is the distance from the center of the lens to F. The thin lens model assumes ideal lenses, and that light refracts in the principal plane through the center of the lens.*

side. By making the sides of the lens curved, a positive (convex) lens that focuses light, or a negative (concave) lens that spreads light can be constructed. Parallel light that reaches an *ideal* positive lens will converge to a single point along the optical axis, referred to as the *focal point*, F. The *focal length*, f, which is the distance between the center of the lens and the focal point, indicates how strongly the lens converges or diverges light. This is illustrated in Figure 9.

In reality, not all light traveling through the lens is converging at exactly the same point. There are several different physical effects in play. For example, the index of refraction is wavelength dependent, which causes different *optical aberrations*, i.e., distortions in the image. To counter these effects, modern camera lenses are often assembled from a large number (up to about 20) of optical lens elements. The overarching design goal is to maximize the amount of light gathered (the irradiance, E, in Equation 25), while minimizing optical aberrations.

The *thin lens approximation* assumes the lens behaves as an ideal lens, and that light is refracted at a single plane – the *principal plane* – ignoring optical effects due to the thickness of the lens. This significantly simplifies the analysis of simple optical systems. Using the thin lens approximation, a camera lens can be modeled as a single positive lens that projects light onto the image plane. The lens is focused at a certain distance, z_f, by adjusting the distance between the image plane and the lens, z_l. The relation between the two is given by the thin lens formula:

$$\frac{1}{z_l} + \frac{1}{z_f} = \frac{1}{f} \quad \Leftrightarrow \quad z_l = \frac{f z_f}{z_f - f}. \tag{31}$$

Any object that is a distance z_f away from the lens is said to lie on the *focus plane*, and will be in focus. For example, with a $f = 100$ mm lens, an object at a distance $z_f = 3$ m is in focus if the lens–image plane distance is $z_l = 103.45$ mm.

The *aperture* of a lens is the opening through which light can pass to the image plane. The aperture is physically limited by the size of the lens and the enclosure in which the lens is mounted. In camera lenses, there is usually also a mechanical variably sized aperture stop inserted somewhere in the optical path to further limit the aperture and control how much light reaches the image plane. The aperture is measured in f-stops (usually written, e.g., $f/5.6$), which is defined as the ratio of focal length to the diameter of the aperture, a_{diam}, as follows:

$$f_{stop} = \frac{f}{a_{diam}}. \tag{32}$$

Light emanating from a point closer or farther away than the focus plane will converge to a point behind or in front of the image plane, respectively. At the intersection with the image plane, the cone of light will therefore result in a disc rather a single point. This is called the *circle of confusion*, and the effect causes objects that are not in focus to be blurred. The circle of confusion is smaller at smaller apertures (i.e., using a larger f-stop number), as the cone of light from any single point will be narrower. At the same time, less light is reaching the image plane, so a longer shutter time is necessary. The range of depths for which the projected image is acceptably sharp is called the *depth of field* (DOF).

The thin lens model is commonly used in rendering systems as it provides a simple and intuitive model for image formation. It gives adequate results for most applications, and allows important artistic control. For example, a large aperture gives a very shallow depth of field and the possibility to isolate objects at a certain distance to direct the viewer's attention. Rendering using the thin lens model is usually achieved using ray tracing, i.e., shooting rays through different random positions on the image plane and lens [24], although rasterization can be generalized to support the thin lens model [2, 3].

2.5.4 Advanced Models

In some applications, a more sophisticated simulation of the optics of the camera lens is necessary. This is the case when the rendered images have to precisely match photographed ones, for example, in production of special effects. The many lens elements of real photographic camera lenses cannot be accurately approximated as thin lenses. In some situations, the *thick lens approximation* can be used instead. Similar to the thin lens model it assumes ideal lenses, but it takes their thickness into account by working with two separate principal planes.

For even more flexibility, it is necessary to model the full lens system as a set of lens elements and apertures [67]. This allows simulation of many of the optical imperfections that real camera lenses exhibit, as well as accurate *bokeh*, i.e., the aesthetic quality of the blur from out-of-focus regions. The shape of the aperture stop and different optical aberrations cause the circle of confusion to have a certain shape, and an often non-uniform intensity on the image plane. These effects have a huge impact on how out of focus areas are depicted by a real camera. Advanced camera models that can simulate these types of effects are increasingly important in the strive for a higher level of realism. For this purpose, efficient light transport through complex lens systems continues to be an important research field [61].

We have now discussed the difficulties of simulating the camera system and the most common approximations used. For further information, refer to the surveys by Barsky et al. [9, 10]. In our work, we have implemented several different camera models, including the thin lens model for simulation of depth of field. The research presented in Paper VII focuses on motion blur rendering, where the exposure is computed by integrating over time as described in Section 2.5.1. In the other papers, however, the focus lies on simulating the light–surface interactions within the scene, which is largely orthogonal to the choice of camera model. Therefore, to make the evaluation of the algorithms easier, we present results rendered using a simple pinhole camera model.

This part concludes the overview of the mechanics of light transport. In the following section, I will discuss how light transport is solved using numerical Monte Carlo methods.

3 Monte Carlo Rendering

As we have seen in Section 2, photorealistic rendering requires accurate simulation of light transport by solving the rendering equation. This a difficult problem as it involves a multi-dimensional integral that is theoretically of infinite dimensionality due to its recursive formulation. Hence, closed-form solutions are intractable, and we must resort to numerical methods. However, even deterministic numerical integration methods using quadrature rules are not practical, as their error bounds grow exponentially with increasing dimensionality. Many other fields, including the physical sciences, engineering, and mathematics, face problems of similar character. For these classes of multi-dimensional integrals, stochastic *Monte Carlo methods* are well-suited.

Monte Carlo (MC) methods are based on random sampling to solve deterministic problems. By averaging over a large number of random samples from the problem domain, an *estimate* of the correct result is obtained. The methods are thus *stochastic* in that a correct answer is not guaranteed, but the expected results will generally be correct. In this section, I will briefly describe Monte Carlo theory and give an overview of the methods commonly used in computer graphics, before introducing our novel techniques in Section 4. Let us start by reviewing basic probability theory.

3.1 Probability Theory

The study and analysis of random phenomena are mathematically described by probability theory, which is divided into its discrete and continuous counterparts. Discrete probability theory deals mostly with probabilities of discrete events (e.g., dice rolls) and combinatorial problems, while for Monte Carlo rendering, we work almost entirely in continuous space and will therefore focus on the latter.

3.1.1 Random Variables

A *random variable*, X, is a variable that does not have a fixed single value, but rather it takes on a value based on the outcome of a random experiment. The *sample space*, S, is the set of all possible outcomes of an experiment. Formally, a random variable is a function defined over the sample space, and each time an outcome $s \in S$ occurs, the random variable takes on the value $X(s)$. The value resulting from a given experiment is called a *realization* of the random variable. We follow standard conventions in that we denote random variables in upper case, and their realizations in lower case. The probability of each outcome is defined by a *probability distribution*. If X is a discrete random variable, its probability distribution is given by a probability mass function (pmf), $p_X(x)$, defined as:

$$p_X(x) \;=\; P(X = x) \;=\; P(\{s \in S : X(s) = x\}), \tag{33}$$

where $P(X=x)$ denotes the probability of X taking on the value x. In the discrete case, the total probability must sum to one, i.e., $\sum p_X(x) = 1$. For example, if X denotes the outcome of a die roll, we have $p_X(x) = 1/6$ for each possible realization $x \in \{1,\ldots,6\}$. (Note that X is here an identity function on S so that $X(s) = s$.)

A *continuous* random variable can take on a continuous range of values, and its probability distribution is given by a *probability density function* (pdf), $f_X(x)$. While the probability of a continuous random variable taking on *exactly* a single value x is zero, the pdf defines the probability of an outcome falling within a range of values, $[a,b]$, as follows:

$$P(a \leq X \leq b) \;=\; \int_a^b f_X(x)dx. \tag{34}$$

As per the definition, the probability density function is positive, $f_X(x) \geq 0$, and integrates to one, i.e., $\int f_X(x) = 1$. The *cumulative distribution function* (cdf), $F_X(x)$, is the probability of a continuous random variable taking on a value that is less than or equal to x. The cdf is found by integrating over the pdf, as follows:

$$F_X(x) \;=\; P(X \leq x) \;=\; \int_{-\infty}^{x} f_X(t)dt. \tag{35}$$

Note that the cdf is monotonically increasing (not necessarily strictly), and has the values $F(x) = 0$ when $x \to -\infty$, and $F(x) = 1$ when $x \to \infty$. Note that a necessary condition for the probability density to exist is that the cumulative distribution function is differentiable everywhere, in which case the pdf is given as the derivative of the cdf:

$$f_X(x) = \frac{dF_X(x)}{dx}. \tag{36}$$

Note that we will omit the subscript, X, when it is obvious which random variable the functions refer to. The probability density function and the cumulate distribution function are two key concepts in probability theory, which I will make extensive use of in this dissertation.

3.1.2 Expectation and Variance

If a random experiment is repeated a very large number of times, the arithmetic mean of the results will converge to some value. This value is called the *mean* or *expected value* of a random variable describing the experiment. More formally, the expected value, $E(X)$, of a random variable, X, is the weighted average of the values it can take, with the weights given by the probability distribution of X. For example, the expected value of a random variable describing the outcome of a die roll, is $E(X) = 1 \cdot \frac{1}{6} + 2 \cdot \frac{1}{6} + \ldots + 6 \cdot \frac{1}{6} = 3.5$. For a continuous random variable, the expected value is (if it exists) given by:

$$E(X) = \int_{-\infty}^{\infty} x f_X(x)dx. \tag{37}$$

More generally, if $g(X)$ is a function of a random variable, X, with a probability density function f_X, its expected value is given by the inner product of f_X and g:

$$E[g(X)] = \int_{-\infty}^{\infty} g(x) f_X(x) dx. \tag{38}$$

By formulating the expected value as a Lebesgue integral, it follows from the linearity property that the expected value is a linear operator [41], that is:

$$E(aX + bY) = aE(X) + bE(Y). \tag{39}$$

The linearity holds true for *any* two random variables, X, Y, and $a, b \in \mathbb{R}$.

Now, assuming the mean of the results from an experiment is known, a natural question is, what is the variation of the results around this mean? The variation is measured as the *variance* of a random variable, which is informally a measure of how concentrated the probability distribution is around its mean. Formally, the variance, $V(X)$, of a random variable, X, is defined as the expected *squared* deviation from its mean or expected value, as follows:

$$V(X) = E[(X - E(X))^2], \tag{40}$$

which can alternatively be written as $V(X) = E(X^2) - E(X)^2$. For a continuous random variable, X, the variance is thus explicitly given by the integral:

$$V(X) = \int_{-\infty}^{\infty} (x - \mu_X)^2 f_X(x) dx, \qquad \text{where } \mu_X = E(X). \tag{41}$$

It should be noted that the units of the variance are the square of the units of the random variable. For example, the variance of a random variable representing irradiance [W$\cdot$m^{-2}], will have units W$^2\cdot$m^{-4}. Therefore, it is often more intuitive to talk about a random variable's *standard deviation*, σ, which has the same units as the variable itself. The standard deviation is defined as the square root of the variance (often denoted σ^2):

$$\sigma_X = \sqrt{V(X)}. \tag{42}$$

In addition, it is useful to know how linear operations on random variables affect their variance. For this purpose, we first need to know if the random variables are correlated or not, i.e., if there is a linear dependence. Correlation is defined in terms of the *covariance* between two random variables, X and Y, as follows:

$$Cov(X,Y) = E[(X - \mu_X)(Y - \mu_Y)] = E(XY) - E(X)E(Y). \tag{43}$$

Note that the covariance between two *independent* random variables is always zero, but the converse is not true; two random variables may have zero covariance, but still exhibit a non-linear dependence. It is often convenient to measure the covariance adjusted for standard deviation. This is defined as the (Pearson product-moment) *correlation coefficient*, $\rho_{X,Y}$:

$$\rho_{X,Y} = \frac{Cov(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X \sigma_Y}, \qquad -1 \le \rho_{X,Y} \le 1, \tag{44}$$

i.e., the correlation coefficient is limited to the range $[-1,1]$. A value of 1 implies a positive perfectly linear relationship, i.e., all pairs of observations from X,Y lie on a line for which y increases as x increases, while -1 implies a negative linear relationship. We are now ready to describe how the variance changes due to linear operations. First, note that scaling a random variables changes its variance by the square of the scale:

$$V(aX) = \int_{-\infty}^{\infty} (ax - a\mu_X)^2 f_X(x) dx = \ldots = a^2 V(X). \tag{45}$$

The variance of a linear combination of two random variables is given by:

$$V(aX + bY) = a^2 V(X) + b^2 V(Y) + 2ab \cdot Cov(X,Y). \tag{46}$$

If the variables are uncorrelated, which is often the case, their covariance is zero. This leads to the so-called Bienaymé formula, which states that the variance of a sum of uncorrelated random variables, X_i, is the sum of their variances:

$$V\left(\sum_{i=1}^{n} X_i\right) = \sum_{i=1}^{n} V(X_i). \tag{47}$$

We have now briefly touched upon the subject of probability theory, in order to introduce the terminology necessary to discuss Monte Carlo methods and variance reduction techniques. For a more comprehensive overview, we refer to textbooks in the field [41, 49].

3.2 Monte Carlo Integration

Broadly speaking, Monte Carlo methods build on probability theory to estimate the expected value of a random variable by random sampling. Hence, by posing the problem we want to solve, e.g., the integral of a function, as an expectation, Monte Carlo methods can be used to estimate the answer. Let us start with some definitions.

3.2.1 Estimators

In probability theory, a *random sample* of length n (i.e., sample size n) represents the outcomes of n independent experiments, in which the same quantity is measured. Given a random variable, X, with distribution f_X, a random sample is thus a set of n independent, identically distributed (*iid*) random variables with distribution f_X. The i^{th} experiment is described by X_i, which has a realized value x_i, i.e., the value obtained when actually making the experiment. In the following, we often use term *sample* loosely to refer to a random sample of length one (a single X_i) and sometimes its realization x_i. The term *sampling* refers to the process of generating samples from a given probability distribution.

As discussed in the previous section, the expected value is the limit of the mean result of a repeated experiment. Formally, the *law of large numbers* states that the sample mean converges to the expected value as $n \to \infty$. Hence, we can approximate the expected value of an arbitrary function, $E[g(X)]$, by drawing a set of samples $\{x_1, x_2, \ldots, x_n\}$ from X and taking the mean of $g(x)$ over the samples. This is called the n-sample *Monte Carlo estimate*, $\widetilde{g_n}$, of $E[g(X)]$. The corresponding random variable, $\widetilde{g_n}(X)$, is called a Monte Carlo *estimator*, and is given by:

$$\widetilde{g_n}(X) = \frac{1}{n} \sum_{i=1}^{n} g(X_i). \tag{48}$$

Each realization of the estimator $\widetilde{g_n}(X)$ gives an estimate of the expected value we are seeking. These estimates will almost certainly vary from the correct value, but there will be no statistical bias, i.e., the estimator is said to be *unbiased*. Mathematically, the bias of an estimator is measured as the difference between its expected value and the value it is estimating. In this case, $E[\widetilde{g_n}(X)] - E[g(X)] = 0$ due to the linearity of expected value (Equation 39).

The estimator in Equation 48 is also *consistent*, which means that it converges in probability to the quantity being estimated. Formally:

$$\lim_{n \to \infty} P(|\widetilde{g_n}(X) - E[g(X)]| < \varepsilon) = 1, \tag{49}$$

for any fixed $\varepsilon > 0$, i.e., the probability of being close to the estimated value increases with growing sample size, n, to become one in the limit. Being both unbiased and consistent are desirable properties of a Monte Carlo estimator, as it guarantees the estimates are correct on average and that the variance decreases with more samples. Note that the two properties do not always go hand in hand; an estimator may be unbiased, but not consistent, and similarly, many estimators are biased while still being consistent.

3.2.2 Integration

Monte Carlo estimators are very useful as we can pose almost any problem as an expectation, and hence estimate its value by averaging over random samples. For example, in our case, we are interested in solving difficult multi-dimensional integrals, i.e., the rendering equation. Assume we want to compute the integral, I, of a function $h(x)$:

$$I = \int_{x \in D} h(x) dx, \tag{50}$$

defined over some (possibly multi-dimensional) domain, D. The integral can be cast as an expectation by rewriting it as follows (c.f., Equation 38):

$$I = \int_{x \in D} h(x) dx = \int_{x \in D} \frac{h(x)}{f(x)} f(x) dx = E\left[\frac{h(X)}{f(X)}\right], \tag{51}$$

where X is a random variable with a probability density function $f(X)$. This equality holds true for any pdf defined over D, as long as $f(x) > 0$ when $h(x) \neq 0$. Hence, the Monte Carlo estimator of I is given by:

$$\widetilde{I}_n(X) = \frac{1}{n}\sum_{i=1}^{n}\frac{h(X_i)}{f(X_i)}, \tag{52}$$

where X_i are iid random variables with a probability distribution defined by $f(x)$. The process of approximating an integral by averaging over random samples is called *Monte Carlo integration*. It is a powerful technique as it works in any dimension and is conceptually simple. In order to estimate an integral, all we need to do is to generate random samples from a distribution with pdf $f(x)$, and average over the values of $h(x)/f(x)$, evaluated at each sample.

As a simple example, consider the evaluation of the definite integral: $I = \int_a^b h(x)dx$. The simplest Monte Carlo estimator draws samples from a uniform distribution defined over the integration domain, $U(a,b)$, which has a pdf:

$$f_X(x) = \begin{cases} \frac{1}{b-a} & \text{for } x \in [a,b], \\ 0 & \text{otherwise.} \end{cases} \tag{53}$$

The Monte Carlo estimate is given by:

$$\widetilde{I}_n = \frac{1}{n}\sum_{i=1}^{n}\frac{h(x_i)}{f_X(x_i)} = \frac{b-a}{n}\sum_{i=1}^{n}h(x_i), \qquad \text{for } X_i \sim U(a,b). \tag{54}$$

It is easy to show that the corresponding MC estimator, $\widetilde{I}_n(X)$, is an unbiased estimator of I. Using Equation 38 and 39, we find that:

$$\begin{aligned} E[\widetilde{I}_n(X)] &= \frac{b-a}{n}\sum_{i=1}^{n}E[h(X_i)] \\ &= \frac{b-a}{n}\sum_{i=1}^{n}\int_{-\infty}^{\infty}h(x)f_X(x)dx \\ &= \frac{b-a}{n}\frac{1}{b-a}\sum_{i=1}^{n}\int_a^b h(x)dx \;=\; \int_a^b h(x)dx \;=\; I. \end{aligned} \tag{55}$$

3.2.3 Variance Reduction

The variance of the general Monte Carlo estimator $\widetilde{g}_n(X)$ of $E[g(X)]$ (Equation 48), can easily be expressed in terms of the variance of $g(X)$. Using Equation 45 and 47, we note that:

$$V[\widetilde{g}_n(X)] = V\left[\frac{1}{n}\sum_{i=1}^{n}g(X_i)\right] = \frac{1}{n^2}\sum_{i=1}^{n}V[g(X_i)] = \frac{1}{n}V[g(X)], \tag{56}$$

since $X_i \sim X$ are iid random variables. Thus, the variance goes to zero as $n \to \infty$, assuming $V[g(X)]$ is bounded. To put a probabilistic bound on the error, *Chebyshev's inequality* states that the probability of a random variable, X, being more than k standard deviations away from its mean, μ_X, is at most $1/k^2$, for any real $k > 0$, that is:

$$P(|X - \mu_X| \geq k\sigma) \ \leq \ \frac{1}{k^2}. \tag{57}$$

This applies to any distribution with finite variance. Inserting Equation 56, we arrive at:

$$P\left(|\widetilde{g_n}(X) - E[g(X)]| \geq \frac{1}{\sqrt{n}} k \sqrt{V[g(X)]} \right) \ \leq \ \frac{1}{k^2}, \tag{58}$$

i.e., for any fixed k, the error decreases in the rate $O(1/\sqrt{n})$ as we increase the number of samples n. This is usually expressed informally as the *convergence rate* of the basic Monte Carlo estimator being $1/\sqrt{n}$. This rate is irrespective of the dimensionality, which explains the usefulness of Monte Carlo methods for multi-dimensional problems. Unfortunately, $1/\sqrt{n}$ is a slow rate of convergence; in order to reduce the error to a $1/10^{\text{th}}$, we have to use $100\times$ as many samples.

In Monte Carlo simulation of light transport, it is often very costly to draw new samples and evaluate their values. For example, in computation of indirect illumination, the sample space is defined as the set of all possible light paths between a light source and the image plane. Each such path may bounce numerous times, resulting in many light–surface interactions that need to be sampled and evaluated. Therefore, it is critical to reduce the variance of the Monte Carlo estimators used, in order to achieve more accurate results using fewer samples.

For standard Monte Carlo integration (Equation 52), the variance of the estimator $\widetilde{I_n}(X)$ is given by:

$$V[\widetilde{I_n}(X)] = \frac{1}{n} V \left[\frac{h(X)}{f(X)} \right]. \tag{59}$$

In order to reduce the variance of $\widetilde{I_n}(X)$, i.e., lower its variance at any given number of samples, n, there are three main strategies. First, we can select the probability distribution, $f(X)$, so that it places more samples in important areas of the domain, i.e., *importance sampling*. Second, by reformulating the problem so that the quantity being estimated, $h(X)$, has lower variance, e.g., using *control variates*, we can reduce the overall variance. Both of these techniques require knowledge of the integrand, $h(x)$. Last, by using a good sampling method for drawing *well-distributed* samples x_i from X_i, better results can be obtained than if uniform random sampling is used. This is possible even without further knowledge of the involved functions.

In our research, we have applied all three strategies for reducing the variance in Monte Carlo integration of direct illumination. In the following sections, I will discuss the different techniques in more detail.

3.3 Importance Sampling

The goal of importance sampling is to select a probability distribution, $f(x)$, which minimizes the variance in Monte Carlo integration (Equation 59), i.e., makes the variance $V[h(X)/f(X)]$ as small as possible.

3.3.1 Choosing a Probability Distribution

Intuitively, it is desirable to choose the probability distribution, $f(x)$, so that the quotient $h(x)/f(x)$ is as close to constant as possible, i.e., to keep the variation around its mean low (c.f., Equation 40). For the common case of $h(x) \geq 0$, $\forall x \in D$, it is obvious that the ideal choice is a probability distribution that is *proportional to the integrand* itself, i.e., $f(x) = c \cdot h(x)$. In this case, $h(x)/f(x) = 1/c$ everywhere, and the variance is zero. For the general case of an arbitrary integrand, it can be shown that the distribution:

$$f(x) \propto |\, h(x) \,|, \tag{60}$$

is the optimal choice [122].[5] Note that the optimal pdf is only of theoretical interest, as in order to integrate to one, the normalization factor c would have to be $c = 1/\int h(x)dx = 1/I$, but I is the unknown quantity that we are trying to estimate.

However, what this means in practice is that we should design the probability distribution (which is often called the *importance function*) so that more samples are placed where the function being integrated is large, and fewer where its value is small. The large sample values in dense regions will be weighted down since the probability density is higher there, and low values are weighted up. This may seem counterintuitive, but it keeps the variance of each random sample low, and hence reduces the overall variance of the estimator.

For correctness, it was earlier mentioned that $f(x) > 0$ must be fulfilled wherever the integrand is nonzero. However, this is usually not enough for good results. We must also be careful to ensure that $f(x)$ is not very small when the integrand is large. If $f(x) \ll h(x)$, we may get outliers with arbitrarily large values since we divide by $f(x)$, i.e., the tails of the distribution matters. In addition to being representative for the function being integrated, the most difficult criteria to fulfill is that the probability distribution should be easy to construct and draw samples from. Remember that the motivation for importance sampling in the first place is to reduce the computational cost of reaching a desired accuracy.

Depending on the application, there are many different choices of probability distributions. For example, for evaluating direct illumination, a common choice is to draw samples over the hemisphere with a distribution that is proportional to the BRDF. For a given point on a surface, $\mathbf{x}$, and outgoing direction, ω_o, the BRDF is a two-dimensional function defined over incident directions on the hemisphere, which means the probability distribution is $p(\omega_i) \propto f_r(\mathbf{x}, \omega_i, \omega_o)$. Another common choice is to sample according to the incident illumination from a light source,

[5] Note that we generally do not have to worry about negative integrands since we are integrating physical quantities (e.g., radiance) that are, by definition, positive.

e.g., an environment map, in which case $p(\omega_i) \propto L_{env}(\omega_i)$. Neither of these two choices work well in the general case, as they only take part of the integrand into account. To address this problem, Veach proposed a method called *multiple importance sampling* [147], which combines multiple importance sampling strategies into a combined estimator:

$$\widetilde{I}_n = \sum_{i=1}^{m} \frac{1}{n_i} \sum_{j=1}^{n_i} w_i(X_{i,j}) \frac{h(X_{i,j})}{f_i(X_{i,j})}, \tag{61}$$

where n_i samples ($\sum n_i = n$) are drawn from each of m probability distributions, f_i. The weights, w_i, determine how much influence each estimator, $h(X_{i,j})/f_i(X_{i,j})$, should have. By appropriate choice of weights, the combined estimator will be both unbiased and have a lower variance than if any one of the individual importance functions was used.

However, when none of the importance functions approximate the integrand, $h(x)$, well, the combined variance will still be high. Our research deals largely with how to construct and directly sample difficult probability distributions, in particular for the case of evaluating direct illumination. The work will be discussed in detail in Section 4. For now, assuming an appropriate pdf, $f_X(x)$, has been chosen, let us look at how samples from $f_X(x)$ can be generated.

3.3.2 Independent Sampling Methods

In this section, we will review some standard techniques for generating independently distributed random samples, given a probability density function. A good understanding of these methods is necessary to explain our work in Section 4.

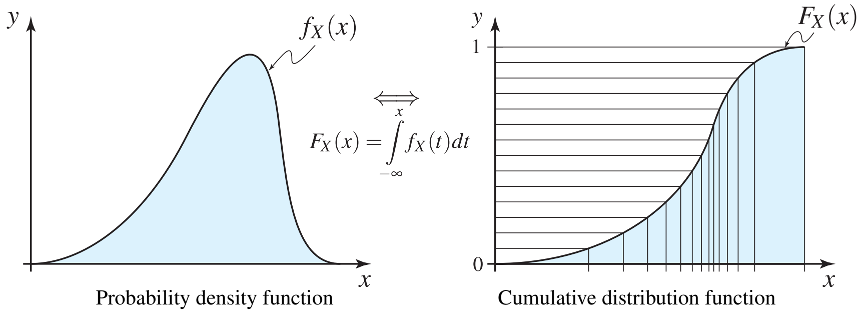
Inversion Sampling The most straightforward method for generating independent random samples from a continuous probability distribution, $f_X(x)$, is called *inversion sampling*. It is based on the *probability integral transform* theorem, which states that if X is a random variable with cumulative distribution function $F_X(x)$, then the random variable defined as:

$$Y = F_X(X), \tag{62}$$

has a uniform distribution, i.e., $Y \sim U(0,1)$. This leads to a powerful method for sampling by using the *inverse* cumulative distribution function, F_X^{-1}, given by:

$$X = F_X^{-1}(Y) \sim f_X(x), \qquad \text{if } Y \sim U(0,1), \tag{63}$$

i.e., by generating uniform random samples over the unit interval and mapping them using the inverse cdf, we get random samples, $X \sim f_X(x)$, following the desired distribution. The method generalizes to higher dimensions, e.g., by using separable marginal cumulative distribution functions. The rationale for the probability integral transform can be intuitively understood as the cdf is a non-uniform



y
$f_X(x)$
x
Probability density function
$\Longleftrightarrow$
$F_X(x) = \int_{-\infty}^{x} f_X(t)dt$
y
1
0
$F_X(x)$
x
Cumulative distribution function

Figure 10: *Inversion sampling is based on generating uniformly distributed samples, $Y \sim U(0,1)$, and evaluating $X = F_X^{-1}(Y)$ to get samples distributed according to the pdf $f_X(x)$. This can be visualized as placing samples with a uniform density on the y-axis and finding the intersection with the cdf (as shown on the right). In regions of high slope, i.e., where $f_X(x)$ is large, the samples will be packed more tightly along the x-axis, exactly according to $f_X(x)$.*

mapping onto $[0,1]$, i.e., $F_X : \mathbb{R} \to [0,1]$. In regions with a low probability density, f_X, the random observations will be sparse, but the slope of F_X will also be low (as its derivative is f_X). Hence, the observations are packed into a small range on the y-axis. The opposite is true in regions with a high probability density. In effect, the variations in probability density are exactly cancelled out by the mapping, F_X, resulting in a uniform density on the y-axis. Inversion sampling exploits this by applying the inverse mapping. This is illustrated in Figure 10.

Inversion sampling requires that the inverse of the cdf is known. Therefore, in computer graphics problems, the method is only practical when simple geometric models or analytical expressions for the probability distribution can be defined, for which closed-form expressions of the inverse cdf:s exist. For example, Shirley et al. [129] apply inversion sampling to uniformly emitting light sources of simple geometric shapes, e.g., spheres and polygons. Similarly, some of the simpler analytical reflection models can be directly sampled. Examples include the Phong [130], Lafortune [75], and Ward [152] BRDFs. To support arbitrary BRDFs, Lawrence et al. [81] use a data-driven approac where the reflectance is tabulated and stored in a compact factored representation, which is sampled by numerically inverting the cdf:s of 1D factors.

Rejection Sampling In many real problems, the cumulative distribution function is not known and cannot easily be computed or inverted. In such cases, *rejection sampling* may be used to sample an arbitrary target distribution. The method is most easily explained from the observation that, in order to sample an arbitrary distribution, we can sample *uniformly* from the region under the graph of its probability density function, $f(x)$. Where $f(x)$ is large, more samples will be placed,

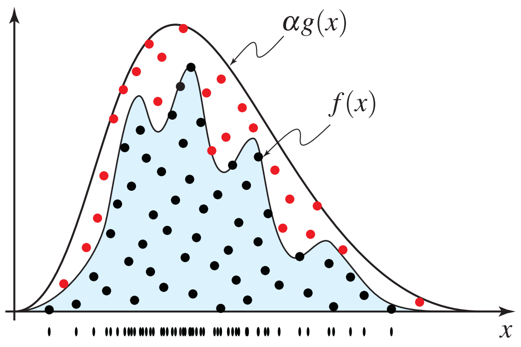


Figure 11: *Rejection sampling can be visualized as uniformly sampling the area under the graph of the envelope function, $\alpha g(x)$, which is designed to conservatively enclose the target distribution, $f(x)$. All samples above the graph of $f(x)$ are rejected (marked in red). The projection on the x-axis of the accepted samples will be distributed exactly according to $f(x)$.*

and vice versa. Hence, the samples' x-coordinates will be exactly distributed according to $f(x)$.

With rejection sampling, in order to uniformly sample the region under $f(x)$, uniform samples are first generated under the graph of a larger *envelope function* that fully encloses $f(x)$. Each sample is then either accepted or rejected based on whether it lies above or below $f(x)$. Only samples under $f(x)$ belongs to the target distribution. The envelope function is given by $\alpha g(x)$, which is chosen so that:

$$f(x) \leq \alpha g(x), \quad \forall x, \tag{64}$$

where $g(x)$ is a probability density function. The pdf $g(x)$ is called the *proposal distribution*, and should ideally be a distribution chosen so that it can be efficiently sampled using other methods, e.g., inversion sampling. The procedure is illustrated in Figure 11.

Formally, the method proceeds by first generating iid samples X_i from the distribution $g(x)$, and then randomly accepting each sample with a probability:

$$P(\text{accept} \mid X) = \frac{f(X)}{\alpha g(X)}. \tag{65}$$

In practice, this may be done by generating uniform random samples $U_i \sim U(0,1)$, and performing the operation:

$$\text{if} \quad U_i \leq \frac{f(X_i)}{\alpha g(X_i)} \quad \text{then } Y_j = X_i \ \ (\text{accept}). \tag{66}$$

The set of surviving samples, Y_j, are distributed according to $f(x)$. The total *acceptance rate*, i.e., the ratio of accepted samples to the total number of samples generated and tested, is given by:

$$P(accept) = \int_{-\infty}^{\infty} P(\text{accept} \mid X = x)\, g(x) dx = \int_{-\infty}^{\infty} \frac{f(x)}{\alpha g(x)} g(x) dx = \frac{1}{\alpha}, \tag{67}$$

i.e., the ratio between the areas under $f(x)$ and $\alpha g(x)$. The method generalizes to higher dimensions, where the acceptance rate is intuitively the ratio of volumes under $f(\mathbf{x})$ to $\alpha g(\mathbf{x})$. This diminishes exponentially with increasing dimensionality, i.e., the acceptance rate is α^{-s} in s dimensions, if the acceptance rate is $1/\alpha$ in each dimension. Hence, the method quickly becomes impractical, as it is often difficult to find an envelope function that tightly encloses the target distribution. Despite this, there are some examples of rendering techniques where rejection sampling has been successfully used. For instance, a method for sampling of the product of surface reflectance and environment map lighting was proposed by Burke et al. [17]. In order to accelerate rejection sampling of complex probability distributions, we present a novel hierarchical method in Paper IV. The rationale is to quickly reject large groups of samples, by hierarchically constructing and refining the envelope function. See Section 4.2.2 for further details.

Sampling Importance Resampling As we have seen, rejection sampling requires determining a suitable scale factor, α. However, for many pairs of distributions, $f(x)$ and $g(x)$, if the scale is sufficiently large to guarantee bounding of $f(x)$, then the acceptance rate is impractically low. This may, e.g., be the case if there are sharp peaks in the target distribution that are not accurately represented by the proposal distribution, $g(x)$. The *sampling importance resampling* (SIR) algorithm [120, 121] addresses this problem. The method is related to rejection sampling and similarly makes use of a proposal distribution, but it does not depend on determining a scale factor, α.

The SIR algorithm starts by generating M samples, X_i, from the known proposal distribution, $g(x)$. A weight, w_i, is then associated with each sample based on the the ratio between the probability densities of the target and proposal distributions:

$$w_i = \frac{f(X_i)}{g(X_i)}. \tag{68}$$

Based on these weights, a *discrete* probability distribution, $p(x_i)$, for the samples is defined by the normalized weights, as follows:

$$p(x_i) = \frac{w_i}{\sum_i w_i}. \tag{69}$$

Finally, in the resampling step, a smaller set of $n < M$ samples, Y_j, are drawn *with replacement* from the samples X_i according to the probability distribution$p(x_i)$. The final samples, Y_j, will only be *approximately* distributed according to the target distribution, $f(x)$, since the selection is limited by the initial sampling. Hence, as $M \to \infty$, the distribution of Y_j approaches $f(x)$. Note that since samples are drawn with replacement, the same sample may occur multiple times in the final distribution.

The resampling step is easy to implement as it works with a discrete probability distribution. One can, for example, compute its cumulative distribution and perform a binary search based on a uniform random number to pick a sample with the correct probability. The SIR algorithm has been applied in computer graphics for sampling of direct lighting [129, 143], BRDFs [143], and the product of environment map lighting and BRDF [17]. The main drawback is that, in order to limit the bias introduced by a finite M, it is important to keep n comparatively small, e.g., $n < 0.1M$. This limits the practically of using SIR in computer graphics, as the cost of drawing a large set of proposal samples often cannot be motivated.

Discussion In summary, all of the discussed independent sampling methods transform an s-dimensional uniform distribution, $U(0,1)^s$, into the desired target distribution. This is done either using direct methods (inversion sampling), or by generating and rejecting samples from a proposal distribution. In Paper I, we propose a further direct sampling method, called *hierarchical sample warping*. The method uses a hierarchy of conditional probabilities to transform a uniform distribution into a distribution that closely approximates an arbitrary target distribution, without relying on the distribution being normalized, nor its inverse cdf to be known.[6] The properties of our method will be further discussed in Section 4.2.1.

Although applicable in any dimension, the efficiency of all the mentioned independent sampling methods, ours included, is limited by the so-called *curse of dimensionality*. Generating independent random samples for efficient importance sampling in very high-dimensional spaces becomes practically impossible. The methods are therefore best suited for relatively low-dimensional problems, such as Monte Carlo integration of direct illumination (over the hemisphere or light source), depth of field (lens), and motion blur (time). For sampling in high-dimensional spaces, such as the space of all light paths in light transport problems [147], Markov chain methods, e.g., the Metropolis-Hastings algorithm, are better suited. These methods generate samples that are *correlated* rather than independent, which leads to a fundamentally different approach. Although not the primary focus of this dissertation, I will include a brief description.

3.3.3 Markov Chain Methods

For high-dimensional sampling problems, modeling the problem as a Markov chain allows it the be decomposed into a sequence of simpler problems. This makes sampling possible even if very little is known about the final target distribution, f, as is the case for the path formulation of light transport. The *Metropolis-Hastings* algorithm [57, 92], is one of the most general Markov chain Monte Carlo (MCMC) methods. To explain it, we will need a few definitions.

[6]Our method *exactly* samples the target distribution if its pdf is piecewise constant.

A *Markov chain*, $\{X^{(t)}\}$, is a sequence of *dependent* random variables:

$$X^{(0)}, X^{(1)}, \ldots, X^{(t)}, \ldots, \tag{70}$$

such that the conditional probability distribution (called *Markov kernel*) of $X^{(t)}$ given the previous variables, only depends on $X^{(t-1)}$. Informally, the Markov chain can be seen as wandering around the parameter space, remembering only where it has been in the previous iteration. In many cases, there exists a *stationary* distribution, f, such that if $X^{(t)} \sim f$, then $X^{(t+1)} \sim f$. The Markov chain will typically converge to f, regardless of the starting point, $X^{(0)}$. Hence, by defining a Markov chain with stationary distribution equal to our target distribution, it is possible to generate samples approximately distributed according to f once the chain has converged. In practice, it is hard to know when the chain has converged, so instead a fixed number of the first draws are usually discarded.

The main difference to the previous methods, is that the generated samples will not be statistically independent, as is required by the basic Monte Carlo estimator (Equation 48). Fortunately, under certain conditions, i.e., that the chain is aperiodic, irreducible and positive recurrent (for definitions, we refer to the statistical literature [118]), the chain is said to be *ergodic* and the ergodic theorem holds, which states that:

$$\frac{1}{M}\sum_{i=1}^{M} g(X^{(i)}) \;\rightarrow\; \int_{-\infty}^{\infty} g(x)f(x)dx \;=\; E[g(X)], \tag{71}$$

as $M \rightarrow \infty$, where $f(x)$ is the stationary distribution, and $X^{(1)}, \ldots, X^{(M)}$ are M values from the (ergodic) Markov chain. This is the Markov chain analogue of the law of large numbers that forms the basis of Monte Carlo methods. Hence, under the right conditions, Markov chains can safely be used in Monte Carlo algorithms, ignoring the correlation between the generated samples.

The Metropolis-Hastings algorithm [57] is a flexible method for defining a Markov kernel with an arbitrary desired stationary distribution. In the general case, the algorithm uses a conditional proposal distribution, $q(y\,|\,x)$, which should be chosen so that it is easy to sample. At each iteration, the algorithm generates a random variable $Y_t \sim q(y\,|\,x^{(t)})$, and randomly accepts it, i.e., $X^{(t+1)} = Y_t$, or otherwise rejects it, in which case $X^{(t+1)} = X^{(t)}$. The probability of accepting the proposal Y_t is given by $\rho(x^{(t)}, Y_t)$, where ρ is defined as follows:

$$\rho(x,y) \;=\; \min\left[\frac{f(y)}{f(x)}\frac{q(x\,|\,y)}{q(y\,|\,x)}, 1\right]. \tag{72}$$

The proposal distribution, q, can be chosen almost arbitrarily, with the only requirements that Equation 72 should be possible to evaluate, and that it should allow exploration over the entire support of f. In the independent Metropolis-Hastings algorithm (IMHA), which is a special case of the general algorithm, the proposal distribution, q, is required to be independent of the current state, i.e., $q(y\,|\,x) = q(y)$. This leads to an algorithm similar to rejection sampling, but with the difference that a scale α, such that $f(x) \leq \alpha g(x)$, does not have to be determined. Similar to sampling importance resampling (SIR), the same sample can occur multiple times due to the rejection step. However, note that as the samples generated by IMHA are correlated, the simpler independent sampling methods are preferable if possible.

When the Metropolis-Hastings algorithm is applied to solve the rendering equation, it is called *Metropolis light transport* (MLT) [148]. Here, the sampling domain is the entire space of light paths, and proposals are usually generated by *mutations* of the current path. The algorithm is one of few that can sample the entire path space. However, for lower-dimensional sampling problems, good estimates are achieved much faster using explicit methods, which is the main focus of this dissertation. In the next section, we will look at a variance reduction technique that is fundamentally different from importance sampling.

3.4 Control Variates

The method of *control variates* is a fundamental Monte Carlo technique, which takes advantage of correlation between different random variables to reduce the variance of the basic sample mean estimator (c.f., Equation 48). This is different from importance sampling, which adapts the distribution of samples to reduce the variance. The two techniques can be combined with good results.

3.4.1 Control Variate Estimator

Assume we have a function of a random variable, $g(X)$, and want to estimate the unknown quantity $E[g(X)]$. The basic Monte Carlo estimator gives the expectation as the sample mean of $g(X_i)$. However, in some categories of problems, it is possible to construct a related random variable, $h(X)$, such that $h(X)$ is correlated to $g(X)$, but has a *known* expectation, $E[h(X)] = \mu_h$. The idea is to exploit samples from $h(X)$ to improve the estimation of the unknown quantity, $E[g(X)]$. For this purpose, we introduce a new random variable:

$$Y = g(X) - \beta(h(X) - \mu_h), \tag{73}$$

which potentially has less variance than $g(X)$ itself, if β and $h(X)$ are chosen appropriately. The *control variate estimator* is the sample mean of Y:

$$\widetilde{g}_{n,\beta}(Y) = \frac{1}{n}\sum_{i=1}^{n}(g(X_i) - \beta h(X_i)) + \beta\mu_h. \tag{74}$$

It is easy to realize that $\widetilde{g}_{n,\beta}$ is an *unbiased* estimator of $E[g(X)]$, due to the linearity of expected value. Similarly, the estimator is *consistent*, as the sample means of $g(X_i)$ and $h(X_i)$ converge to $E[g(X)]$ and $E[h(X)]$, respectively, as $n \to \infty$.

To analyze the variance of the control variate estimator, $\widetilde{g}_{n,\beta}(Y)$, we introduce the variables $Z = g(X)$ and $W = h(X)$. Following Equation 46, the variance is:

$$V(\widetilde{g}_{n,\beta}(Y)) = \frac{1}{n}\left(\sigma_Z^2 + \beta^2\sigma_W^2 - 2\beta\, Cov(Z,W)\right). \tag{75}$$

The goal is to choose β so that the variance is minimized. As Equation 75 is a quadratic function on the form $a\beta^2 + b\beta + c = 0$, it has its minimum when the derivative is zero, i.e., where $2a\beta + b = 0 \Leftrightarrow \beta = -b/2a$. In this case, the minimum is found at β^*:

$$\beta^* = \frac{Cov(Z,W)}{\sigma_W^2}. \tag{76}$$

Inserted into Equation 75, the variance at the optimal β^*, is given by:

$$V(\widetilde{g}_{n,\beta^*}(Y)) = \frac{1}{n}\left(\sigma_Z^2 - \frac{Cov(Z,W)^2}{\sigma_W^2}\right) = \frac{1}{n}\sigma_Z^2\,(1 - \rho_{Z,W}^2), \tag{77}$$

where $\rho_{Z,W}$ is the correlation coefficient (c.f., Equation 44) between Z and W. This is an interesting result, since $(1 - \rho_{Z,W}^2) \leq 1$ for any Z and W (due to $\rho_{Z,W} \in [-1,1]$ by definition). Hence, under the assumption that β^* is known, the variance of the control variate estimator can never be larger than the variance of the basic Monte Carlo estimator, which is $\frac{1}{n}\sigma_Z^2$. If $g(X)$ and $h(X)$ are uncorrelated, the variance will be the same as before, while in all other cases the variance will be lower. In the special case of $\rho_{Z,W} = \pm 1$, the variance reduces to zero.

In practice, finding the optimal parameter β^* is not realistic, as it requires knowing the exact covariance between $g(X)$ and $h(X)$, which implies knowledge of $E[g(X)]$. Therefore, the variance reduction will not be as large as Equation 77 suggests, and it is even possible to end up with a larger variance than the original estimator if the control variate is only weakly correlated, or if β is chosen poorly.

There are a couple of different alternatives for finding a good β. First, one can draw an independent set of samples from $g(X)$ and $h(X)$, from which an unbiased estimate of β^* can be computed. However, this is costly, as these samples will not be used for anything else than determining β. As an alternative, β^* can be estimated from the same set of samples as are used to evaluate the final control variate estimator. This risks introducing bias, as the value of β will be dependent on the observed samples. Finally, in some cases, domain-specific knowledge about g and h can be used to select an appropriate β, either using some heuristic or by picking a fixed number.

3.4.2 Combination with Importance Sampling

Control variates may very well be applied to Monte Carlo integration, and the method combines naturally with importance sampling. In this case, the goal is to compute an integral $I = \int g(x)dx$, which can be posed as computing the expectation $E\left[g(X)/f(X)\right]$ where $X \sim f(x)$ (c.f., Equation 51). We introduce an approximation, $h(X)$, which should ideally be strongly correlated to $g(X)$, and define Z

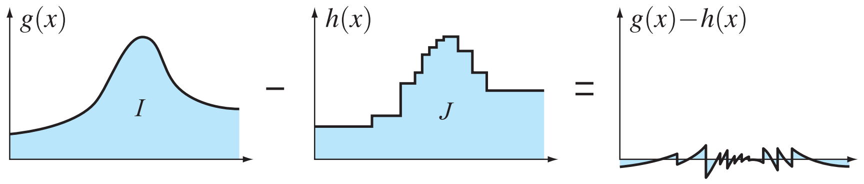


Figure 12: *The method of control variates can be illustrated as a Monte Carlo integration of the difference $g(x)-\beta h(x)$, where h is an approximation of g with known integral, J. The goal is to find the unknown integral $I=\int g(x)dx$. If g and h are correlated, random samples from their difference have a lower variance than samples from $g(X)$. In the figure, $\beta=1$ for illustrative purposes.*

and W above as:

$$Z = \frac{g(X)}{f(X)} \quad \text{and} \quad W = \frac{h(X)}{f(X)}, \qquad \text{where } X \sim f(x). \tag{78}$$

In order to use W as a control variate term, its expectation, $E(W)$, must be known. This is equal to the integral $J = \int h(x)dx$, as follows:

$$E(W) = E\left[\frac{h(X)}{f(X)}\right] = \int_{-\infty}^{\infty} h(x)dx = J, \qquad \text{for } X \sim f(x). \tag{79}$$

The control variate estimator for the unknown integral I is now, following the earlier definition, given by:

$$\widetilde{I}_{n,\beta} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{g(X_i)}{f(X_i)} - \beta\frac{h(X_i)}{f(X_i)}\right) + \beta J, \tag{80}$$

The only requirement on $h(x)$ is that the function should be possible to evaluate at any point in the parameter space that has a nonzero probability of being sampled. In addition, its true integral, J, must be known. If such a function can be found, it can thus immediately be used to create a control variate estimator for the integral we seek. The intuition for the method is straightforward; the *difference* between the integrand and an approximation of it, is evaluated using Monte Carlo integration, and then the known integral of the approximation, J, is added back as a correction term. The concept is illustrated in Figure 12.

Control variates have the potential to significantly reduce the variance in Monte Carlo simulations. Despite this, the method has not been widely used in computer graphics for solving light transport problems. This is likely due to the difficulty of finding a good enough approximation, h, that has both a significant correlation *and* a known integral. If the approximation is only weakly correlated, the variance reduction will not be large enough to motivate the added cost, and selecting an appropriate β may be difficult. On the other hand, if the analytical integral of h is not known, bias will be introduced.

Nevertheless, there are a few examples of techniques where control variates have been applied to light transport problems. For indirect illumination, control variate estimators based on an ambient term [72], the incident radiance field [74], or a radiosity solution [142], have been used. For direct illumination, Szécsi et al. [141] combine control variates with importance sampling, but their approach is limited in that it ignores visibility and assumes only diffuse materials. In Paper III, we propose a combined approach that takes all three terms in the integral for direct illumination under distant lighting into account, i.e., lighting, BRDF, and visibility. That work builds on our techniques for importance sampling developed in Paper I and II, and extends them with an accurate approximation of the visibility term.

In summary, as more accurate and faster methods for approximating light transport become available, there is a growing potential for using these approximations to define good estimators based on control variates. The approach is attractive as it, for example, could allow approximate solutions developed for real-time rendering to be reused in the context of unbiased Monte Carlo rendering. In the following section, we will discuss the last of our main variance reduction techniques.

3.5 High Quality Sampling Points

In classical Monte Carlo theory, each sample that drives the simulation is statistically independent of all others. For example, generating a sample from a given probability distribution usually involves first drawing a sample $X \sim U(0,1)$, and then transforming it into the desired distribution using one of the methods in Section 3.3. Note that nothing prevents two independent samples from randomly being close to each other, or even groups of samples from clumping together. For Monte Carlo integration, this can lead to a poor exploration of the integral and high variance as a result.

It was discovered early on that by spacing the samples more evenly to better fill out the sampling domain, the convergence rate of Monte Carlo methods can be greatly improved. Intuitively, by explicitly avoiding large gaps between the sampling points, rather than drawing each sample independently from a uniform distribution, the risk of missing important features is reduced. There are many different methods for constructing point sets and sequences with good properties. The term point set usually refers to a finite fixed number of points, while the term point sequence implies an ordered list of points. A sequence can sometimes be extended to include more, possibly infinitely many, points. There is a clear distinction made between *deterministic* and *stochastic* methods due to their very different theoretical foundations, with different terminology, applications, and measures. I will give examples of both types in this section.

The "quality" of a point set or sequence is often measured by its *discrepancy* and/or *spectral* properties (see below). Points that are deterministically constructed to have a low discrepancy, while still sharing some properties with random variables, are called *quasi-random* points. Combining deterministic quasi-random sampling with classical Monte Carlo methods, leads to *quasi-Monte Carlo* (QMC) theory.

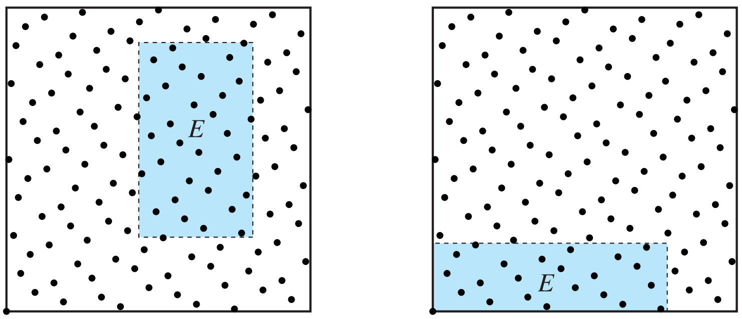


Figure 13: *A sequence is uniformly distributed modulo one if each subset, E, contains (asymptotically) a number of points proportional to the volume of E, i.e., $N\lambda_s(E)$, where N is the number of points. The star discrepancy, D_N^*, is defined using all subsets on the form $[\mathbf{0},\mathbf{a})$, i.e., axis-aligned boxes with one corner in the origin (as shown in the example on the right).*

This has been the focus of extensive research due to its many desirable properties, such as deterministic error bounds and fast convergence. On the other hand, with stochastically generated points, repeated experiments generate different results and Monte Carlo theory for, e.g., estimating the variance and convergence rate apply. Algorithmically, QMC methods generally only differ from their MC counterparts in the type of input samples used. Therefore, the majority of our research can be applied in both settings, and examples of both types are shown in Papers I–IV. Additionally, in Paper VII, a new method is presented for generating quasi-random points with properties optimized for motion blur rasterization.

In the following, I will give an overview of the theory and methods for generating high-quality sampling points for Monte Carlo and quasi-Monte Carlo applications.

3.5.1 Low Discrepancy Sequences

For a sequence of N s-dimensional points, $S = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$, in the unit cube $[0,1)^s$, let $A(E,N)$ be the number of points that fall into a subset $E \subseteq [0,1)^s$. The sequence is said to be (asymptotically) equidistributed or *uniformly distributed modulo one* [69], if for every interval $[\mathbf{a},\mathbf{b}) \subseteq [0,1)^s$, we have:

$$\lim_{N\to\infty} \frac{A([\mathbf{a},\mathbf{b}),N)}{N} = \lambda_s([\mathbf{a},\mathbf{b})), \tag{81}$$

where $\lambda_s(.)$ is the Lebesgue measure in s dimensions, i.e., generally the volume. Hence, the proportion of points falling in a subinterval is asymptotically proportional to the length (area, volume) of that interval, as illustrated in Figure 13.

Equation 81 is a rather weak requirement, as even independent draws from a uniform distribution will create a uniformly distributed sequence.[7] Some sequences are obviously more uniformly distributed than others. It is therefore important to be able to measure the uniformity of a point sequence. Several different measures exist. For example, different variations of *discrepancy* measures are commonly used, which indicate how *non-uniform* a sequence is. The *extreme discrepancy*, D_N, is defined as [101]:

$$D_N(S) = \sup_J \left| \frac{A(J,N)}{N} - \lambda_s(J) \right|, \tag{82}$$

where J is the set of all subintervals of the form $[\mathbf{a},\mathbf{b}) \subseteq [0,1)^s$, i.e., axis-aligned boxes, and the supremum (sup) returns the greatest element over that set. The geometrical interpretation is that the discrepancy is the largest absolute difference between the proportion of points in a subinterval to the volume of that interval. The lower the number, the more uniformly distributed the sequence is. Instead of considering all axis-aligned subintervals, the set J is often restricted to all subintervals on the form $[\mathbf{0},\mathbf{a})$, which gives the *star discrepancy*:

$$D_N^*(S) = \sup_{\mathbf{x}\in[0,1)^s} \left| \frac{A([\mathbf{0},\mathbf{x}),N)}{N} - \lambda_s([\mathbf{0},\mathbf{x})) \right| = \sup_{\mathbf{x}\in[0,1)^s} |\Delta_S(\mathbf{x})|, \tag{83}$$

where $\Delta_S(\mathbf{x})$ is called the *discrepancy function* of S. Another common variant is the $L_{q,N}$-discrepancy, which is the L_q norm ($q \geq 1$) of the discrepancy function. Additionally, it should be noted that for infinite sequences, the discrepancy is taken over the first N points of the sequence.

For deterministic sequences, tight bounds on the discrepancy can often be found. This is important, as it allows the integration error in quasi-Monte Carlo methods to be bounded. The *Koksma-Hlawka inequality* states that if a function, f, has bounded variation on the unit cube, then the quasi-Monte Carlo integration error using the sequence S, is bounded as follows [101]:

$$\left| \int_{[0,1)^s} f(\mathbf{u})d\mathbf{u} - \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}_i) \right| \leq V(f)D_N^*(S), \tag{84}$$

where $V(f)$ is the variation in the sense of Hardy-Krause. The details are beyond the scope of this introduction, but the conclusion is clear; the convergence rate of $D_N^*(S)$ with increasing N determines how fast the QMC integration error diminishes. We refer to the books by Niederreiter [101], and Dick and Pillichshammer [29] for further definitions.

For carefully selected deterministic *low-discrepancy* (quasi-random) point sets, the discrepancy can be in the order of $O(\log^{s-1} N/N)$ [101], which is often much better than the $O(1/\sqrt{N})$ probabilistic error bound of classical Monte Carlo methods. This explains the popularity of QMC methods in a wide range of applications.

[7]The reverse is not true, however, as many sequences are uniformly distributed without being realizations of independent uniform random variables.

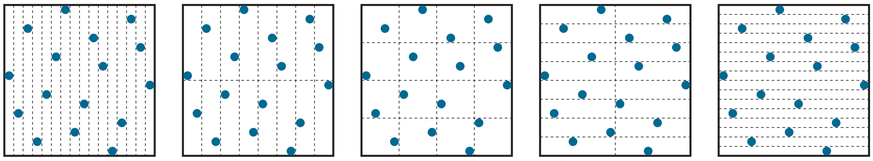


Figure 14: *Example of a (t,m,s)-net in base $b=2$ with parameters $t=0$, $m=4$, and dimensionality $s=2$. The five squares show all elementary intervals with area $b^{t-m}=2^{-4}$, which each holds exactly $b^t=1$ point.*

However, it should be noted that the integrals involved in light transport simulation typically have discontinuities that cause the variation to be unbounded. The theoretical error bounds thus do *not* apply in this context, although in practice, there is plenty of empirical evidence that QMC methods work well for rendering problems [68, 106]. Two methods for generating low-discrepancy point sets dominate: regular *lattices* and so-called *digital nets*. I will focus on the latter, as there are many important applications in computer graphics.

As an alternative to the fully deterministic methods, it is possible to use *randomized* quasi-random sequences [104] in Monte Carlo methods. In this case, the good uniformity properties are well preserved, while the random nature allows probabilistic error bounds (e.g., variance) to be estimated using classical Monte Carlo theory. This approach elegantly avoids the problem of determining QMC error bounds for difficult integrals, and such methods are referred to as *randomized quasi-Monte Carlo* (RQMC) methods.

3.5.2 Digital Nets

Digital (t,m,s)-nets or just *digital nets* [101], are quasi-random sequences with extensive stratification properties. Informally, let the sampling domain, i.e., the unit cube in s-dimensional space, $[0,1)^s$, be divided into a large number of axis-aligned strata called *elementary intervals*. In order for a point set to be a (t,m,s)-net, each such stratum must hold exactly the same number of quasi-random points, as defined by strict rules. This guarantees very well-distributed points. The net is called a *digital* net when digital methods are used to construct it, which represents an important category of sample generation methods. A brief definition follows.

The elementary intervals in base b are rectangular boxes in $[0,1)^s$, with end points at integer multiples of b^{-k}, for some $k \geq 0$ and $b \geq 2$. The set of *all* elementary intervals, E, is defined as:

$$E = \prod_{j=1}^{s} \left[\frac{l_j}{b^{k_j}}, \frac{l_j+1}{b^{k_j}} \right), \tag{85}$$

where $k_j \geq 0$ and $0 \leq l_j < b^{k_j}$. A sequence of b^m points is a (t,m,s)-net if every elementary interval of volume b^{t-m} contains exactly b^t points. As $t \geq 0$ controls the number of points per stratum, it is usually referred to as a "quality" parameter (lower value is better). Figure 14 shows a simple example of a $(0,4,2)$-net in base $b = 2$. The parameters indicate that there are $b^m = 2^4 = 16$ points defined over the two-dimensional unit square, $[0,1)^2$. In this case, each elementary interval of area $b^{t-m} = 2^{-4}$ has exactly $b^t = 1$ point, as illustrated in the figure.

Digital nets can be constructed using arithmetic defined on a finite field, $\mathbb{F}_q$, where q is prime [101]. The field $\mathbb{F}_q$ consists of elements numbered $\{0,\ldots,q-1\}$, and all operations are performed modulo q. For example, digital computers use the binary arithmetic of the finite field $\{0,1\}$. A digital net can be defined using a set of *generator matrices*, $C_1,\ldots,C_s$, over $\mathbb{F}_b$, where each matrix is an $m \times m$ matrix and b is the base as before. Working in base $b = 2$, the i^{th} component of the j^{th} point, $\mathbf{x}_j$, is given as:

$$x_j^{(i)} = \left(2^{-1},\ldots,2^{-m}\right) \left[C_i \begin{pmatrix} d_0(j) \\ \vdots \\ d_{m-1}(j) \end{pmatrix} \right] \in [0,1), \tag{86}$$

where $d_k(j)$ are the bits of the binary representation of j, where $j \in \{0,\ldots,2^m-1\}$, with d_0 being the least significant bit.

Some constructions of (t,m,s)-nets are extensible, which means points can be added to each stratum by shrinking the elementary intervals. In fact, there are ways to construct *infinite* sequences of points, so-called (t,s)-sequences, with the property that subsequences form (t,m,s)-nets. Formally, a (t,s)-sequence in base b is an infinite sequence of points, $\mathbf{x}_i$, such that any subsequence of length b^m (starting at a multiple ofb^m) is a (t,m,s)-net in base b. A simple example is the one-dimensional van der Corput sequence [146], which is a $(0,1)$-sequence constructed by reversing the base b representation of the natural numbers around the decimal point. Other often cited examples are the Halton [53], Sobol' [133], Faure [40], and Niederreiter [100] sequences. Similar to digital nets, there are important digital methods for constructing (t,s)-sequences. For an extensive treatment of digital nets and sequences, refer to the book by Dick and Pillichshammer [29].

Many different common constructions of digital (t,m,s)-nets and sequences are used for sampling purposes in computer graphics [68]. It should be noted that the (t,m,s)-property of a point set is not enough by itself to guarantee a large *minimum* distance between points, although the points are overall well-distributed. Two points in neighboring elementary intervals may, e.g., be arbitrarily close to each other. Some methods have been developed explicitly for generating points with a large or maximized minimum distance [50, 51]. This is an important property in computer graphics, as we will see in the next section. Additionally, in Paper VII, we develop a new digital construction of three-dimensional (t,m,s)-nets with properties that are particularly useful in our applications. The points generated using our method are sequentially ordered in one dimension, while have a well-distributed two-dimensional projection in the other two.

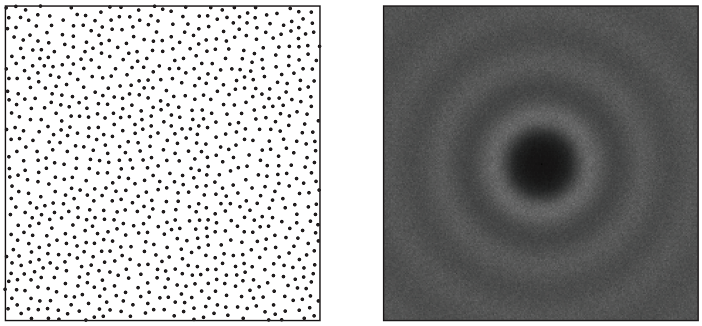


Figure 15: *Example of well-distributed sampling points in two dimensions and their associated power spectrum with blue noise characteristics. The points were generated using the method by Gamito and Maddock [42].*

3.5.3 Blue Noise Properties

In addition to being uniformly distributed, it is in many cases important that sampling points used for rendering are *irregular*, i.e., without any obvious symmetries or regular structures. The motivation comes from the *Nyquist-Shannon sampling theorem* in signal processing [112]; if a continuous function, f, is bandwidth limited and contains no frequencies higher than f_{max}, then it can be exactly reconstructed from regularly spaced samples if the sampling frequency, f_s, is higher than the Nyquist rate, $2f_{max}$, that is:

$$f_s > 2f_{max}. \tag{87}$$

Otherwise, perfect reconstruction of the original function is not guaranteed as *aliasing* causes folding of high frequencies into lower frequencies. This can most easily be explained by looking at the problem in the frequency domain. When f is sampled at a frequency f_s, it can be shown that its frequency spectrum is replicated at a spacing f_s. Aliasing occurs if these replicated spectra overlap, that is, when $f_s/2 < f_{max}$, in which case it becomes impossible to recover the original function's spectrum from the sampled function.

The quantities being sampled in computer graphics are generally *not* bandwidth limited, e.g., there are sharp transitions due to geometrical edges and discontinuities in the shading, so some degree of aliasing almost always occurs. If a point set has obvious regularities along certain directions, high-frequency details are likely to be folded into distracting visible artifacts and moiré patterns [23]. The rationale for using irregular point sets is that, although aliasing still occurs, the artifacts are largely replaced by less distracting *random noise*.

To quantify these characteristics of a point set, i.e., both uniformity and irregularity, the discrepancy measures are generally not sufficient. Additionally, for stochastic points, firm bounds on the discrepancy may be hard to find. Instead, a *spectral analysis* of the point set is commonly performed [23]. The standard approach is to compute the power spectrum, or the estimated power density spectrum in case of stochastically generated points [125]. For a point set to be irregular, its power (density) spectrum should be radially symmetric. That is, there should not be any particular direction with a significantly different frequency spectrum. Additionally, it is desirable to have a low or zero energy at low frequencies (other than the DC peak), which indicates a lack of large gaps between points, as such gaps would cause low-frequency peaks. Stochastic point sets fulfilling these requirements are often loosely said to have *blue noise* properties, referring to the term “blue noise” used to describe random noise characterized by a power density that increases with increasing frequency. Figure 15 shows an example of well-distributed points and their associated spectrum.

Another, simpler measure of uniformity, is the *minimum point distance*. This is defined as the smallest distance between any two points in a point set. By maximizing the minimum point distance, it is ensured that the points are very well spread out without any clumps. However, this measure does not consider the irregularity of the points, as for example, a regular lattice has a very large minimum point distance, but also a high regularity.

3.5.4 Stochastic Sampling Methods

There are many stochastic methods for generating random points with a higher uniformity than independently drawn random points. The simplest approach is *stratified random sampling*, where the sampling domain is divided into a number of disjoint regions, or *strata*, in which random points are independently placed. In practice, the sampling domain $[0,1)^s$ is generally divided n times along each dimension, for a total of $N = n^s$ strata, and one point is placed at random in each stratum. The choice of n exactly determines the number of points, which limits the method to lower dimensions as n^s quickly grows large, or we achieve little stratification if n is small.

Several variants of stratified sampling exist. For example, in *latin hypercube* (or *n*-rooks) sampling, the sampling domain is similarly divided into n strata along each dimension, but only a single point is placed in each stratum for every dimension. That is, in 2D each row/column of the grid will have only a single random point. In practice, points can be placed in the strata along the diagonal of the sampling domain, and then the strata along each dimension are randomly permuted. This gives a total of $N = n$ points, but the overall stratification is rather weak. Although these stratification methods yield point sets with an increased uniformity, nothing prevents points in neighboring strata from being arbitrarily close.

To achieve better spectral properties, methods that enforce a strict certain minimum distance between any two points have been developed. Formally, assuming that each sample is placed in the center of a disk with radius r, a point set is said to follow a *Poisson-disk* distribution if no two such disks overlap. The distribution is said to have a *radius* of r. The name comes from the connection with the discrete

Poisson probability distribution, which models the number of samples that fall in any subset of a domain, if samples are placed at random with a certain rate. In Poisson-disk sampling, the constraint that points must fulfill the minimum distance requirement is added. The distribution generalizes to any dimension by considering a sphere or hypersphere of radius r at each point. Poisson-disk distributions were first studied in relation to the spatial distribution of trees [87].

The most straightforward method for generating points from the Poisson-disk distribution is a technique called *dart throwing* [23, 30], which is based on rejection sampling. Points are iteratively drawn from the uniform distribution (i.e., "throwing darts") and tested for inclusion using the Poisson-disk criteria. If a point is not within a distance $2r$ of any other point, it is added to the set and the process is repeated. The probability of finding a valid point in each draw is directly proportional to the area of the empty space between disks (of radius $2r$) placed at the current points. Therefore, the efficiency of the method quickly decreases as more points are generated, making it computationally expensive. It is also difficult to guarantee that a *maximal* distribution is generated, i.e., one where it is impossible to insert any further points without violating the Poisson-disk criteria.

Many faster algorithms have since been developed. A common theme is to exploit spatial data structures (e.g., grids, quadtrees, or scalloped sectors) to keep track of the unsampled space to guide the insertion of new points. Lagae and Dutré give a good overview in their survey [76]. Recent developments have focused on aspects such as efficient parallel implementation, higher dimensionality, and/or producing maximal and unbiased point sets [15, 35, 36, 42, 155]. The traditional argument against using Poisson-disk sampling – that it is too expensive – has thus largely disappeared. Very high-quality point sets can now be generated in empirically linear time, and practically in up to five or six dimensions [36].

As an alternative to Poisson-disk sampling, points with blue noise properties can be generated using a variety of other methods. For example, using models based on statistical mechanics [39], or particular geometric shapes (polyominoes) that can be combined into a large number of possible patterns [103]. The latter was first applied to rendering using the method presented in Paper IV. Another important category of algorithms is based on refining an initial point set to improve its blue noise properties. The classical approach is Lloyd *relaxation* [84], which iteratively moves each point to the center of its Voronoi cell. Modern approaches [8, 126] avoid some of the earlier drawbacks, e.g., convergence to a hexagonal grid. The initial seed distribution may be either a uniform random distribution, or preferably, a point set with approximate blue noise properties for faster convergence. It should be noted, however, that the process of refining a stochastically generated point set generally adds bias.

In summary, the quality of the sampling points used for (quasi-)Monte Carlo methods plays a vital role in reducing the variance or errors of the estimation. In our research, we have applied a variety of different methods, both deterministic low-discrepancy sequences and stochastic Poisson-disk points. In the next section, I will describe the contributions of each of our papers.

4 Contributions

In this section, I will try to succinctly summarize our research and the key contributions of each of the seven publications that form the basis of this doctoral dissertation.

The common goal throughout our work has been to develop advanced techniques for improving the efficiency of Monte Carlo simulation of light transport problems. In particular, we have focused mainly on variance reduction techniques for Monte Carlo integration where the integrand is a *product* with multiple terms. Such integrals occur frequently in physically-based rendering; at each ray–surface interaction, the outgoing radiance is given as a (hemi)spherical integral of the product of the incident illumination, the BRDF, a geometric term, and sometimes visibility (c.f., Equation 19 and 20).

Each of our studies focuses on different aspects of the above problem. Papers I–IV present complete systems for efficiently evaluating product integrals, with applications demonstrating rendering under distant direct illumination. This is followed by Papers V–VII, which focus on specific aspects of the larger problem.

The key contributions of my work can be summarized as:

- ▶ Several practical techniques for importance sampling of the product of lighting and reflectance for unbiased Monte Carlo rendering.
- ▶ Exploiting the triple product of lighting, reflectance, and visibility to further reduce the variance, using either importance sampling or control variates.
- ▶ Hierarchical sampling methods for efficiently transforming uniformly distributed points into any desired probability distribution.
- ▶ Automatic analysis of surface shaders to allow importance sampling, without knowledge of the mathematical formulation of the BRDF.
- ▶ An algorithm for sparse adaptive sampling and caching of the visibility function, for use in variance reduction schemes or direct pre-visualization.
- ▶ An optimized mapping between the (hemi)sphere and the square, which allows efficient implementation of our hierarchical algorithms.
- ▶ A method for generating well-distributed low-discrepancy points in three dimensions, with good two-dimensional projections.

I will start by describing the mathematical framework in which all of our work will be explained. Then, I will discuss our specific solutions in each of the above mentioned areas.

4.1 Mathematical Framework

In the following, we will assume a general form of the rendering equation [65], describing the reflected radiance, L_r, leaving a surface point $\mathbf{x}$ in direction ω_o, as an integral over the visible hemisphere:

$$L_r(\mathbf{x} \to \omega_o) = \int_\Omega L(\mathbf{y} \to \mathbf{x}) f_r(\mathbf{x}, \omega_i, \omega_o) V(\mathbf{x}, \mathbf{y}) \cos\theta_i \, d\omega_i, \tag{88}$$

where $L(\mathbf{y} \to \mathbf{x})$ is radiance leaving a surface at $\mathbf{y}$ in direction towards $\mathbf{x}$. Depending on the application, L may represent light reflected off another surface (i.e., indirect illumination) or emitted from a light source (i.e., direct illumination). In practice, we have focused mainly on the latter.

For convenience, we combine the cosine term with the BRDF to form a combined "reflectance" term, R, as is common practice:[8]

$$R(\mathbf{x}, \omega_i, \omega_o) = \begin{cases} f_r(\mathbf{x}, \omega_i, \omega_o)(\omega_i \cdot \mathbf{n}) & \text{if } \omega_i \cdot \mathbf{n} \geq 0, \\ 0 & \text{otherwise.} \end{cases} \tag{89}$$

Note, when integrating over the visible hemisphere, $\omega_i \cdot \mathbf{n} \geq 0$ always holds true. In some cases, it is more convenient to integrate over the full sphere, in which case the contribution is zero for directions below the horizon. It should also be noted that for practical purposes, we rewrite all integrals over the (hemi)sphere as integrals in $\mathbb{R}^2$ using appropriate mappings. This will be discussed in Section 4.6.

4.1.1 Product Importance Sampling

The basic n-sample Monte Carlo estimator of the integral in Equation 88, given a specific $\mathbf{x}$ and ω_o (omitted below), can be expressed as:

$$\widetilde{L}_{r,n} = \frac{1}{n} \sum_{j=1}^{n} \frac{L(\omega_j) R(\omega_j) V(\omega_j)}{f(\omega_j)}, \tag{90}$$

where $L(\omega_j) = L(\mathbf{y} \to -\omega_j)$ is the radiance arriving from a specific surface (e.g., light source) in direction ω_j, which is visible when V is nonzero. Note that ω_j are here random variables distributed according to $f(\omega)$ over the hemisphere.

This MC estimator is evaluated for each surface point, $\mathbf{x}$, at which we want to estimate the illumination due to a particular light source, L. In reality, when rendering a typical three-dimensional scene, there may be millions of such integrals to estimate. For each chosen sampling direction, ω_j, the three terms, L, R, and V have to be evaluated; L is typically inexpensive for the case of direct illumination, R may be of higher complexity if advanced reflection models are used, and the visibility, V, is evaluated by tracing a ray towards $\mathbf{y}$ to find out if the light source is visible, as seen from $\mathbf{x}$. Ray tracing involves traversing a spatial data

[8]In a few of the papers, the letter B has been used to denote this function, but we switch to R here to avoid confusion with other fields, where B sometimes denotes radiance.

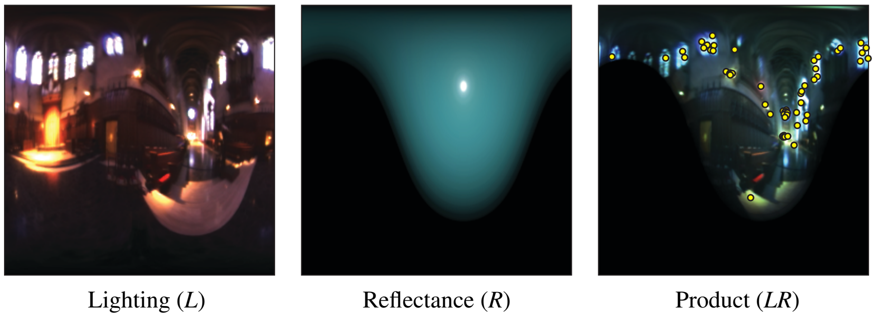


Figure 16: *We introduce several techniques for multiplying and sampling products of functions. One application is efficient Monte Carlo integration of direct illumination, where the product of the lighting, L, and reflectance, R, is sampled, as shown on the right.*

structure and performing multiple ray–primitive intersection tests. In addition, the exact probability density for each sample, $f(\omega_j)$, must be computed in order to achieve unbiased results. Therefore, at each integration point, we can only afford a small number of samples. The goal is to keep n in the range of *tens to hundreds* of samples, for the evaluation of high-frequency direct illumination.

To achieve low variance, we have developed several advanced *importance sampling* techniques for generating samples according to a product distribution. Figure 16 shows an example of sampling the product of distant environment map lighting and reflectance, which is an important application for our theory. Recall that, in order to minimize the variance, the probability distribution of the samples must accurately follow the shape of the integrand (c.f., Section 3.3). Most previous techniques have sampled according to only one of the terms involved, usually the lighting or reflectance (with or without the cosine term).

In Paper I, we present one of the first techniques for directly drawing samples from the following (double) product distribution:

$$f(\omega) = \frac{\widetilde{L}(\omega)\widetilde{R}(\omega)}{L_{ns}}, \qquad \text{with } L_{ns} = \int_{\Omega} \widetilde{L}(\omega)\widetilde{R}(\omega)d\omega, \tag{91}$$

where $\widetilde{L}$ and $\widetilde{R}$ represent accurate *approximations* of the exact illumination and reflectance. Both functions are positive, so the necessary condition that the probability density function is positive is fulfilled, i.e., $f(\omega) \geq 0,\ \forall\omega$. The normalization factor $1/L_{ns}$ is necessary for $f(\omega)$ to integrate to one. The physical interpretation of L_{ns} is that it represents the total contribution due to $\widetilde{L}$ and $\widetilde{R}$, assuming no shadowing (i.e., $V = 1$ everywhere). Combining Equation 90 and 91 gives the following *unbiased* estimator for the reflected radiance:

$$\widetilde{L}_{r,n} = \frac{L_{ns}}{n} \sum_{j=1}^{n} \frac{L(\omega_j)R(\omega_j)}{\widetilde{L}(\omega_j)\widetilde{R}(\omega_j)} V(\omega_j). \tag{92}$$

The variance of this estimator depends on two main factors. First, the approximation errors in the lighting and reflectance cause each of the terms $L/\widetilde{L}$ and $R/\widetilde{R}$ to fluctuate somewhat around one. Second, the visibility term, V, is unknown and each sample obtains a value in $[0,1]$ depending on the visibility between $\mathbf{x}$ and the light source. In the common case of only opaque occluders, V is a binary function. The samples $V(\omega_j)$ can then be seen as following a Bernoulli distribution, i.e., the discrete probability distribution that takes the value 1 with probability q, and 0 with probability $1-q$. The variance of this distribution is $q(1-q)$, which has its maximum at $q=0.5$, i.e., when half of our visibility samples fall in shadow.

In Paper II, the technique is refined so that the exact lighting term, L, can be used without approximation for importance sampling. Similarly, the alternative method for product sampling presented in Paper IV, allows the exact lighting to be used. Using these methods, $\widetilde{L}=L$ and the MC estimator simplifies to:

$$\widetilde{L}_{r,n} = \frac{L_{ns}}{n}\sum_{j=1}^{n}\frac{R(\omega_j)}{\widetilde{R}(\omega_j)}V(\omega_j), \qquad \text{where } L_{ns}=\int_{\Omega} L\widetilde{R}\,d\omega. \tag{93}$$

In this case, the only remaining variance stems from the visibility term and the approximation of the reflectance.

The machinery we have developed for product importance sampling will be further discussed below; in particular, Section 4.2 presents our hierarchical sampling methods, followed by details on the computation of the product distribution in Section 4.3, and approximations of the reflectance in Section 4.4.

4.1.2 Triple Product Importance Sampling

The variance due to the unknown visibility term, V, in Equation 92 and 93 can be significant. Indeed, this is often the main source of noise in rendered images [113]. Unfortunately, the visibility is also difficult to estimate and use for variance reduction. First, unlike the reflectance of a surface or the radiance emitted by a light source, the visibility is globally defined by the geometrical location of all objects in the scene. This means it can locally change quickly if there are nearby objects or complex occluders. Second, it is not a smooth function – each transition from visible to occluded or vice versa, represents a discontinuity. Therefore, the usual approach has been to try and take the other terms, i.e., lighting and/or reflectance, into account, but leave visibility to be evaluated using ray tracing. In two of our publications, we address this by approximating the visibility and including it in the Monte Carlo estimator in an unbiased way, using two different techniques.

The first attempt, in Paper IV, is to include an approximation of the visibility, $\widetilde{V}$, as a third term in the product distribution. We call this *triple product* importance sampling, as samples are drawn from the following probability distribution:

$$f(\omega) \propto L(\omega)\widetilde{R}(\omega)\widetilde{V}(\omega). \tag{94}$$

This is theoretically straightforward, but it is difficult in practice to find a useful approximation, $\widetilde{V}$. Recall that the probability distribution is never allowed to be zero when the integrand is non-zero (c.f., Section 3.3). Hence, visibility has to be conservatively estimated:

$$\widetilde{V}(\mathbf{x},\omega) \;\geq\; V(\mathbf{x},\omega), \quad \forall\, \mathbf{x},\omega. \tag{95}$$

Said another way, the visibility approximation must never report a direction as occluded ($V = 0$) if it is not, but the reverse is acceptable. The naïve solution would be to add a small epsilon to $\widetilde{V}$ to ensure it is never exactly zero. However, although unbiased, this method would lead to outliers whenever a visible direction is falsely classified as occluded, due to the division by a small value in $f(\omega)$.

The approach taken in Paper IV is to define $\widetilde{V}$ using *inner-conservative* bounding geometry, i.e., simple geometric primitives placed fully *inside* opaque objects. This guarantees that the relation in Equation 95 is fulfilled, as if a ray intersects the bounding primitive inside an object, then the ray is also guaranteed to be occluded by the object itself. The reverse is usually not true. The method is not without problems, however, as constructing the bounding geometry can be costly, and we are hence limited to a relatively small number of bounding primitives. The degree of variance reduction depends on how well the bounding geometry approximates the shape of the real objects, and in many cases, a close fit is difficult to achieve. Therefore, we have explored an alternative, more general solution based on control variates, which do not place the same strict requirements on $\widetilde{V}$.

4.1.3 Control Variate Estimator for Visibility

In Paper III, we address the visibility problem by constructing an accurate visibility approximation, $\widetilde{V}$, which is used to define a *control variate* term for reducing the variance. As opposed to triple product importance sampling, this method does *not* require the visibility to be conservatively estimated. Any approximation that is correlated with the true visibility function will do. However, the variance reduction is larger the stronger the correlation is, so in practice, we want $\widetilde{V} \approx V$.

The representation of $\widetilde{V}$ is chosen so that the analytical integral of the triple product $L\widetilde{R}\widetilde{V}$ can be efficiently computed, where L and $\widetilde{R}$ are the exact lighting and approximate reflectance, respectively. These are the same functions as we used for importance sampling in Paper II. This choice allows us to define a control variate estimator for the rendering integral (c.f., Equation 80), as follows:

$$\widetilde{L}_{r,n} = \frac{1}{n}\sum_{j=1}^{n}\left(\frac{LRV - \beta L\widetilde{R}\widetilde{V}}{f(\omega_j)}\right) \;+\; \beta \underbrace{\int_{\Omega} L\widetilde{R}\widetilde{V}d\omega}_{J}, \tag{96}$$

where the parameters ω_j and ω have been omitted for clarity. The integral J represents the reflected radiance, computed using approximate reflectance and visibility functions. The estimator effectively evaluates the difference between the exact radiance and this approximation using Monte Carlo integration, and then J is added back to keep the result unbiased. Since $L\widetilde{R}\widetilde{V}$ is strongly correlated to the function we want to integrate, LRV, it represents a good choice for use as a control variate.

The probability distribution of the samples, ω_j, used to estimate the difference can be chosen arbitrarily. To minimize the variance, we want to use importance sampling and choose $f(\omega) \propto L(\omega)\widetilde{R}(\omega)$, since this product is already available as part of the computation of J. Finally, the value of the coefficient β needs to be chosen. As we saw in Section 3.4, an optimal choice exists, but it depends on the unknown covariance between the terms involved. In our case, $\widetilde{V}$ is constructed using sparse random sampling (see details in Section 4.5), which is a consistent estimation with no strong bias towards either over- or underestimation of the occlusion. A value of $\beta = 1$ was therefore empirically found to work well. Intuitively, this centers the difference between the exact and approximated functions around zero, similar to the example shown in Figure 12. Now, the final estimator is found by inserting the expression for $f(\omega)$ and $\beta = 1$ in Equation 96, as follows:

$$\widetilde{L}_{r,n} = \frac{L_{ns}}{n} \sum_{j=1}^{n} \left(\frac{R(\omega_j)}{\widetilde{R}(\omega_j)} V(\omega_j) - \widetilde{V}(\omega_j) \right) + J, \tag{97}$$

where the normalization factor L_{ns} is the same as before (Equation 93). As we saw earlier, the approximation of reflectance causes some variation, but the variance due to the unknown visibility term is now reduced, often significantly.

Control variates is an attractive technique for many rendering problems, but so far it has not been widely used. The main difficulty is finding a suitable approximation that is both analytically integrable, and strongly correlated with the target function. Additionally, in our case we do not want to restrict ourselves to only static scenes, where long precomputation times can be afforded since the result will be reused over many frames. Hence, only little time can be spent on building the visibility approximation, and the added cost must be motivated by a large enough reduction in variance. The technique presented in Paper III is based on sparse sampling and caching of the visibility function, combined with an efficient binary encoding. These topics will be further discussed in Section 4.5.

4.2 Hierarchical Sampling

The algorithms for (double) product and triple product importance sampling that we have developed in Paper I, II, and IV can be divided into two main components: methods for computing the product distribution, and methods for sampling it, i.e., drawing random samples with the correct distribution. In reality, the two are tightly intertwined. I will start by discussing our novel sampling methods, i.e., how to generate random samples assuming the probability distribution is already known, before describing its computation in Section 4.3. The generated samples are finally used in the Monte Carlo estimators discussed above, in order to compute unbiased estimates of the reflected radiance.

Our goal has generally been to develop *explicit* methods for computing and sampling the product distribution, rather than using traditional statistical methods, where samples are often drawn from a simpler proposal distribution and then adjusted to follow the target distribution (c.f., Sections 3.3.2 and 3.3.3). For explicit sampling, we present an intuitive, efficient hierarchical approach that we call *hierarchical sample warping*. As an example of a statistically-based technique, we also present a novel *hierarchical rejection sampling* scheme. In both cases, hierarchical representations of the product distribution are exploited to make the sampling operations efficient. The two methods will be discussed below.

Preliminaries In the following, we assume that the sampling problem can be expressed in s-dimensional Euclidean space, $\mathbb{R}^s$. For our purposes, that means we transform the (hemi)spherical functions of the rendering integral to $\mathbb{R}^2$ using one of the mappings discussed in Section 4.6. We will limit the discussion to this two-dimensional case, but the methods generalize to higher dimensions.

Let $p(\mathbf{x}) \propto f(\mathbf{x})$ be an arbitrary multi-dimensional probability distribution we want to sample. For example, in our case $f(\mathbf{x})$ may be the (unnormalized) product of lighting and approximated reflectance, mapped to $\mathbb{R}^2$:

$$f(\mathbf{x}) = L(\mathbf{x})\widetilde{R}(\mathbf{x}). \tag{98}$$

We assume that f can be expressed as a two-dimensional discrete *image*, $F(\mathbf{x})$, i.e., a function that is piecewise constant over a regular grid. For practical purposes, we let $\mathbf{x} \in [0,1)^2$, and assume the resolution of the image is a power of two in each dimension. Hence, the image has $2^m \times 2^m$, $m \geq 0$ grid cells or *pixels*.[9] Each pixel with integer coordinates $\mathbf{t} = (x,y)$ for $0 \leq x,y < 2^m$, occupies a little square in the grid. We introduce $\mathbf{s} = (m,\mathbf{t})$ to denote this region of the unit square:

$$\mathbf{s} \;=\; 2^{-m}[x,x+1) \times 2^{-m}[y,y+1) \qquad \subseteq [0,1)^2. \tag{99}$$

Also, note that the area of a pixel is $A(\mathbf{s}) = 2^{-2m}$. The value of a discrete pixel is denoted $F^m_{\mathbf{t}} = F^m_{x,y}$, which is assumed to represent the *average* of the original function over $\mathbf{s}$, which is given by the integral:

$$F^m_{\mathbf{t}} \;= \frac{1}{A(\mathbf{s})}\int_{\mathbf{s}} f(\mathbf{x})\,d\mathbf{x}. \tag{100}$$

Hence, as a consequence, $\int F(\mathbf{x})d\mathbf{x} = \int f(\mathbf{x})d\mathbf{x}$. Note that we can represent $f(\mathbf{x})$ up to any level of accuracy by choosing a high enough resolution, m, but there are practical limits due to memory consumption and computational efficiency. In some cases, the functions we are interested in sampling are already represented as discrete images, in which case F is exact, i.e., $F(\mathbf{x}) = f(\mathbf{x})$. For example, the lighting, L, from an environment map is already discretized to the resolution of the environment map.

To describe our sampling methods, we define an *image hierarchy*, which consists of the full series of images of resolution $2^l \times 2^l$ pixels, where $0 \leq l \leq m$, i.e., all images of size 1×1 to $2^m \times 2^m$, with a power of two number of pixels along each

[9]Note that we use capital letters to denote discrete images, which should not be confused with the cdf of a random variable. Also, note that $\mathbf{x}$ here refers to positions in the unit square.

dimension. The pixels in each image are denoted, $F^l_{x,y}$, where l is called the *level* of an image, ranging from the coarsest resolution ($l = 0$) to the finest ($l = m$). A pixel's integer coordinates are $0 \leq x,y < 2^l$. The pixels at each level represent the average image values over the corresponding squares, i.e., the integrals scaled by the inverse pixel areas, 2^{2l}, c.f., Equation 100. Due to the 2×2 scale between each level, this means that a pixel at a level $l < m$ is given as the average over the four pixels under its support at the next finer level, $l+1$. We have:

$$F^l_{x,y} \;=\; 2^{2l}\int_{\mathbf{s}} F(\mathbf{x})\,d\mathbf{x} \;=\; \frac{1}{4}\sum_{0\leq i,j\leq 1} F^{l+1}_{2x+i,2y+j}, \qquad 0 \leq l < m. \tag{101}$$

Note that according to this definition, the image hierarchy is equivalent to a full mipmap hierarchy [161] computed using a box filter. Methods for sparsely computing suitable image hierarchies for the case where F is a *product* of multiple terms, will be discussed in Section 4.3.

4.2.1 Hierarchical Sample Warping

In our first sampling method, we exploit an exact hierarchical representation of the image, F, to efficiently generate samples from the probability distribution $p(\mathbf{x}) \propto F(\mathbf{x})$. The sampling method is introduced in Paper I and briefly mentioned in Paper II (with pseudocode), but I will give a more formal description here.

Hierarchical Probability Tree We first note that the single pixel at the coarsest level, $F^0_{0,0}$, represents the integral over the entire image. Hence, our probability distribution is defined (with correct scale) as:

$$p(\mathbf{x}) = \frac{F(\mathbf{x})}{F^0_{0,0}}, \qquad \text{since } F^0_{0,0} = \int\limits_{[0,1)^2} F(\mathbf{x})d\mathbf{x}. \tag{102}$$

The marginal probability of a random sample, X, being placed in a region $\mathbf{s} = (l,\mathbf{t})$, is found by integrating the pdf over $\mathbf{s}$ (c.f., Equation 34), as follows:

$$P(X\in\mathbf{s}) \;=\; \int_{\mathbf{s}} p(\mathbf{x})d\mathbf{x} \;=\; \frac{1}{F^0_{0,0}}\int_{\mathbf{s}} F(\mathbf{x})d\mathbf{x} \;=\; 2^{-2l}\,\frac{F^l_{\mathbf{t}}}{F^0_{0,0}}, \tag{103}$$

where we have used Equation 101 in the last step. The expression for $P(X\in\mathbf{s})$ applies at any level in the hierarchy. This can be exploited in sampling strategies that start at the coarsest level and hierarchically place samples according to F. At each step in the hierarchy, the *conditional probabilities* of sampling each of the four pixels at the next finer level are computed. The conditional probability of sampling $\mathbf{s}' = (l',\mathbf{t}')$, under the support of $\mathbf{s}$, is given by:

$$P(X\in\mathbf{s}' \mid X\in\mathbf{s}) \;=\; \frac{P(X\in\mathbf{s}')}{P(X\in\mathbf{s})} \;=\; 2^{2(l-l')}\,\frac{F^{l'}_{\mathbf{t}'}}{F^l_{\mathbf{t}}}, \qquad \text{for } \mathbf{s}'\subseteq\mathbf{s}. \tag{104}$$

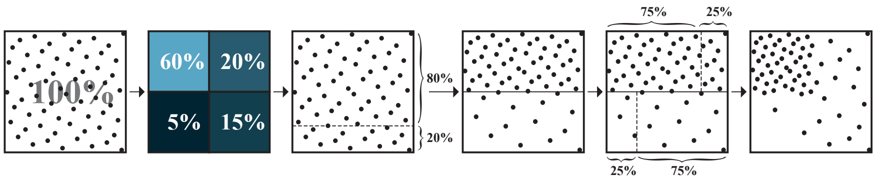


Figure 17: *With our hierarchical sample warping algorithm, a uniformly distributed point set is recursively warped (rescaled) into the desired distribution. At each step in the recursion, the conditional probabilities for the next finer level controls the placement of splitting planes. By warping the point set based on the splitting planes, its density is constantly adjusted to maintain the correct marginal probability density in all parts of the domain. The illustration shows the warping for one full level in the hierarchy in two dimensions.*

In particular, when going from one level to the next, the conditional probabilities for each of the four pixels, $\mathbf{s}' = (l+1, \mathbf{t}')$, are simply:

$$P(X \in \mathbf{s}' \mid X \in \mathbf{s}) = \frac{1}{4}\frac{F^{l+1}_{\mathbf{t}'}}{F^{l}_{\mathbf{t}}}, \qquad \text{for } \mathbf{s}' \subseteq \mathbf{s},\ l' = l+1. \tag{105}$$

Previous work has explored the use of similar hierarchies of conditional probabilities as *decision trees* for random thresholding [20, 22, 78]. In those methods, new random variables are drawn at each level in the hierarchy, in order to randomly select which of the four children nodes to sample. This makes it difficult to achieve high-quality sampling points, as any blue noise properties of the input samples are effectively lost.

Sample Warping We propose an alternative strategy, called *hierarchical sample warping*, which has the advantage that it largely preserves the spectral qualities of the input samples. The method transforms a uniformly distributed point set into samples distributed according to the target distribution $p(\mathbf{x}) \propto F(\mathbf{x})$, by recursively splitting and rescaling, the point set along each dimension. The technique is illustrated in Figure 17. Starting with the coarsest level and the second dimension, y, the point set is split into two halves along a splitting plane at y_s. The plane is positioned so that the probability of a uniformly distributed sample falling in each half, is equal to the corresponding conditional probabilities of sampling each half of the current region.

The points in both halves are then linearly scaled, or *warped*, so that y_s moves back to the center of the interval. Linearly scaling a uniformly distributed point set changes its density, but the points remain uniformly distributed within each half. The process is repeated for each half along the first dimension, x, using the conditional probabilities of each pixel. This completes one step of the warping algorithm. At this point, the point set has a density function that is proportional to

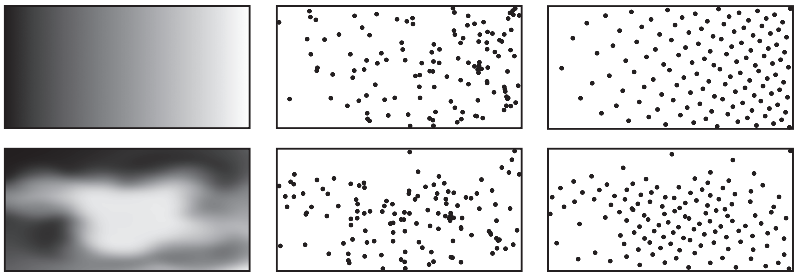


Figure 18: *Two examples that illustrate that the quality of an input point set is largely preserved through our hierarchical sample warping algorithm. In this case, the probability distributions are defined over a rectangular area.*

the 2×2 image at level $l = 1$, i.e., the samples within each of the four pixels are locally uniformly distributed, but with the correct marginal probability density.

The process is recursively repeated for each square that has *one or more samples*, while for empty squares, the recursion can be immediately terminated. Hence, due to induction, once samples have been warped down to the finest level, $l = m$, the overall distribution of the warped points is $p(\mathbf{x}) \propto F(\mathbf{x})$ (Equation 102) as desired. The early termination for empty squares is an important practical aspect, as it allows $F(\mathbf{x})$ to be *sparsely* computed only where needed for a given input point set (we will look at methods for this in Section 4.3). In our application, this greatly reduces the cost of sampling. Since F is a product of multiple terms and may very well be of, e.g., 1024^2 pixels resolution or higher, computing the full product would be prohibitively expensive.

Our algorithm can be seen as a variant of inversion sampling (Section 3.3.1). Similar to sampling by analytic inversion of the cumulative distribution function (cdf), our method transforms uniformly distributed points into the desired distribution. However, this is done hierarchically in multiple steps rather than in a single step. The benefit is that we never have to explicitly compute and invert the full cdf. The main advantage over previous hierarchical approaches using conditional probabilities, is that our method tends to preserve the spectral properties of an input point set; two points that are far apart in the input point set, are likely to be far apart in the final distribution. Figure 18 shows an example of this for two different probability distributions. Another advantage is that our method does not require generating any other random variables than the input point set, which makes the implementation efficient. It should be noted that a minimum point distance is not guaranteed, however, as two points may end up arbitrarily close due being warped differently in neighboring squares. The non-uniform scaling of the point set in each step of the recursion also changes the anisotropy of the point distribution. This effect has been studied by Wei and Wang [156].

4.2.2 Hierarchical Rejection Sampling

Paper IV introduces another hierarchical sampling method, which builds on rejection sampling. Recall that rejection sampling generates random samples by drawing samples from a proposal distribution, and then randomly accepting or rejecting (thresholding) them against the target distribution (c.f., Section 3.3.1). Our method uses a hierarchically computed *upper bound* of $F(\mathbf{x})$ as the proposal distribution, and tests each generated sample against $F(\mathbf{x})$.

We use the notation $\overline{F}^l_{\mathbf{t}}$ to denote the upper bound of F over a region $\mathbf{s} = (l, \mathbf{t})$. The only requirement on the upper bound is that it is conservative, that is:

$$\overline{F}^l_{\mathbf{t}} \geq \max_{\mathbf{x} \in \mathbf{s}} F(\mathbf{x}). \tag{106}$$

In practice, we use a bounding function that gets progressively closer to F as the level, l, increases. To minimize the number of candidate samples, we use hierarchically constructed low-discrepancy points (Section 3.5.2) to gradually fill up the volume under the graph of the bounding function with uniformly distributed points. As soon as a point in any region of the function exceeds the locally computed upper bound, the generation of new candidate points in that region is terminated, as those are guaranteed to be rejected. Regions where points are accepted are, on the other hand, recursively subdivided to refine the upper bound.

The method is motivated by the fact that an *exact* hierarchical representation of $F(\mathbf{x})$, as required for sample warping, is often difficult to compute when F is a product of multiple terms, i.e., $F = F_1 F_2 \dots F_n$. However, an upper bound is easier to compute. In our paper, we conservatively estimate the *maximum* of F over any region, $\mathbf{s}$, as the product of the individual terms' maxima:

$$\overline{F}^l_{\mathbf{t}} = \prod_i \left(\max_{\mathbf{x} \in \mathbf{s}} F_i(\mathbf{x}) \right) \geq \max_{\mathbf{x} \in \mathbf{s}} F(\mathbf{x}). \tag{107}$$

The reason for this being an (over-)conservative estimate and not the exact maximum, is that the peaks of the individual terms, F_i, do not usually align. However, as we go to finer resolutions the estimate is refined, until at the highest resolution, $l = m$, the upper bound is exactly equal to F, i.e., $\overline{F}^m_{\mathbf{t}} = F^m_{\mathbf{t}}$. The underlying assumption is that F is locally smooth. This is generally the case in our applications, even though locally sharp peaks or discontinuities may exist, e.g., due to strong lights in the environment map. Around such peaks, the sampling method will locally refine the upper bound until it reaches the full resolution image.

Properties of the Samples The choice of input samples is an important practical aspect, and the algorithm imposes a few constraints for good results. In order to sample according to an s-dimensional probability distribution, we generate $(s+1)$-dimensional points and use the first dimension for the rejection test. The projection of the remaining s dimensions of the surviving points make up the coordinates of the final samples. For best results, the input points should ideally have a well-distributed projection in these dimensions. Additionally, to allow early termination of the hierarchical refinement and sampling process, which is key for efficiency, the points must be sequentially ordered along the first dimension. Many deterministic low-discrepancy points based on digital nets and sequences (Section 3.5.2) fulfill the latter requirement, while also having reasonably good projections in the other dimensions. In Section 4.7, we will look a novel method for generating three-dimensional point sets, that is specifically designed to improve the projection of the non-ordered dimensions.

For example, in two-dimensional sampling problems (such as our example applications in Section 4.1), we can use standard three-dimensional *Hammersley* points [101], $S = \{\mathbf{x}_0, \dots, \mathbf{x}_{N-1}\}$, with each point defined as:

$$\mathbf{x}_i \;=\; \left(\frac{i}{N},\, \Phi_{p_1}(i),\, \Phi_{p_2}(i)\right) \;\in [0,1)^3, \tag{108}$$

where $\Phi_{p_1}(i)$ and $\Phi_{p_2}(i)$ are the *radical inverses* of the index i in two different prime bases p_1 and p_2, i.e., its digits in the chosen bases reversed around the decimal point. To reduce the bias from deterministically generated points, we add a random jitter in the range $[0, 1/N)$ to the first component, and use random scrambling [104] in the assignment of samples to regions in the paper.

As a rejection sampling-based method, the final number of samples is not known in advance and cannot be exactly controlled, as it depends on how many samples survive the rejection test. Assuming the input points, $\mathbf{x}_i = (x_i^{(1)}, x_i^{(2)}, x_i^{(3)})$, are defined over the unit cube, $[0,1)^3$, the rejection test is given as (c.f., Equation 66):

$$\text{if } \; x_i^{(1)} \leq cF\left(x_i^{(2)}, x_i^{(3)}\right) \;\text{ then }\; \mathbf{y}_j = (x_i^{(2)}, x_i^{(3)}) \;\; \text{(accept)}, \tag{109}$$

where $\{\mathbf{y}_j\}$, $j = 1, \dots, n$ is the set of accepted (two-dimensional) samples. The coefficient c must be chosen so that:

$$cF(\mathbf{x}) \leq 1, \quad \forall \mathbf{x} \qquad \Longleftrightarrow \qquad c \leq \frac{1}{\max F(\mathbf{x})}. \tag{110}$$

The exact maximum of $F(\mathbf{x})$ is often unknown, but in practice, we can use the upper bounds $\overline{F}^l_{\mathbf{t}}$ a few levels down in the hierarchy, to quickly get a conservative, reasonably tight estimate. The expected number of accepted samples, $E(n)$, is then equal to the volume under $cF(\mathbf{x})$, scaled by the total number of input points, i.e.:

$$E(n) = Nc \int\limits_{[0,1)^s} F(\mathbf{x}) d\mathbf{x}. \tag{111}$$

Since the integral of F varies, the value of c must be adjusted accordingly to achieve a number of samples that is approximately constant. For our application of triple product importance sampling in Paper IV, we have designed a heuristic that allows N and c to be iteratively chosen within a small number of iterations (1.3 on average), but this may not always be possible.

In summary, the main advantage of hierarchical rejection sampling compared to sample warping is that the method does *not* require a hierarchical representation of $F(\mathbf{x})$ to be known. It is sufficient that an upper bound can be computed. This allows sampling in cases where F is to difficult or expensive to compute. The price for this flexibility, however, is that the method puts stricter requirements on the input samples. Also, it cannot guarantee a fixed or predictable number of samples, which may be a problem in some applications.

4.3 Sparse Product Evaluation

In the previous section, we have seen how a product distribution described as an image, $F(\mathbf{x})$, can be sampled using two novel hierarchical techniques. Both sampling methods are based on hierarchically evaluating the product, either exactly or using an upper bound, starting at the coarsest level and recursively refining the computation where necessary. The prime applications are double and triple product importance sampling (Section 4.1), where F is defined as a product of lighting, reflectance, and in some cases, visibility. We will look at different methods for approximating these terms later, but for now, let us discuss how the product distribution of two or more arbitrary terms can be evaluated efficiently.

Our primary contributions are in the area of computing the product distribution exactly, i.e., without further approximations than what its individual terms may bring. This is desirable as it allows sampling through hierarchical warping, which is both efficient and flexible. As an alternative for cases where the exact product is too expensive to compute, I will also briefly describe an approximate technique, which was used in Paper IV.

4.3.1 Haar Wavelet Products

Our first approach transforms the problem into the wavelet domain in order to reduce the complexity. I will start by motivating the approach, and then discuss the theory behind it. As before, the focus is on the two-dimensional case for illustrative purposes, although the presented methods generalize to higher dimensions. In the following, consider the case where F is a product of two terms:

$$F(\mathbf{x}) = G(\mathbf{x})H(\mathbf{x}). \tag{112}$$

First, note that although sample warping usually only requires a small subset of the terms in the hierarchy, $F^l_{\mathbf{t}}$, to be computed, the initial coarsest coefficient, $F^0_{0,0}$, is always needed. This represents the integral over the entire image, which is used for normalizing the probability distribution and computing conditional probabilities at the start of the recursion (Equation 105). Computing $F^0_{0,0}$ naïvely is too costly, since:

$$F^0_{0,0} = \int_{[0,1)^2} G(\mathbf{x})H(\mathbf{x})d\mathbf{x} = \frac{1}{2^{2m}} \sum_{0 \le x,y < 2^m} G_{x,y}H_{x,y}, \tag{113}$$

i.e., it requires per-pixel multiplication of the images G and H at their full resolution (note that $F^0_{0,0} \neq G^0_{0,0} H^0_{0,0}$).

The rationale for our work is that, by using appropriate *compressed* representations of G and H, it is possible to significantly reduce the cost of evaluating their product. The compressed representations must be both hierarchical and *compact*, i.e., being able to accurately approximate the functions using a small number of coefficients. In addition, their multiplication should ideally be *sparse*, meaning that only a small number of nonzero terms exist. It turns out that the *Haar wavelet basis* fulfills all these requirements. This realization, together with the development of the hierarchical sample warping method, are the key contributions in Paper I.

The Haar Wavelet Basis In general terms, a *wavelet* function, $\psi(t)$, is a function that describes a localized wave-like oscillation. As such, a wavelet is defined both in terms of its frequency and its location. Transforming a signal into the wavelet domain, i.e., expressing it as a sum of wavelet functions, allows it to be analyzed in both time *and* frequency. Wavelets are therefore widely used in signal processing and analysis applications. The method is related to the Fourier transform, which decomposes a signal into a sum of sinusoids and hence only gives information about its frequency content. For an overview of wavelet analysis, we refer to Mallat's book [86].

For practical purposes, the wavelet functions are usually defined as translations and scalings of a *mother wavelet*, $\psi(t)$, which is designed to have compact support and fulfill the following properties:

$$\int_{-\infty}^{\infty} \psi(t)dt = 0 \quad \text{and} \quad \int_{-\infty}^{\infty} |\psi(t)|dt = 1. \tag{114}$$

Each scaled wavelet function essentially represents a bandpass filter. In order to represent the full range of frequencies, the wavelet function is combined with a so-called *scaling function* [85], $\phi(t)$. The scaling function should behave as a lowpass filter, and is usually chosen so that:

$$\int_{-\infty}^{\infty} \phi(t)dt = 1 \quad \text{and} \quad \int_{-\infty}^{\infty} \phi(t)\psi(t)dt = 0. \tag{115}$$

The continuous wavelet transform considers the set of all possible scaling and translations, which is highly redundant. For practical implementation, the analysis is usually limited to a set of wavelet functions that are defined at discrete steps in time and frequency, i.e., using a *discrete wavelet transform* (DWT).[10] A common choice is:

$$\psi_{l,k}(t) = 2^{l/2}\psi\left(2^l t - k\right), \qquad l \geq 0, \tag{116}$$

where $l, k \in \mathbb{Z}$ are the integer scale and translation, respectively, that are applied to the mother wavelet function. By appropriate choice of scaling and wavelet functions, an *orthonormal* basis is formed, in which any real continuous function can

[10]Note that the discrete wavelet transform still works with continuous signals; it is only the wavelet functions that occur at discrete steps in time and frequency.

be expressed. Each coefficient in the wavelet decomposition represents the localized energy content of the function at a specific frequency. In effect, this property concentrates the energy to a small number of coefficients that accurately represent the overall shape of the function. Minor details are added through wavelets at finer resolutions, but those coefficients tend to be small. This is exploited for *wavelet compression*, where small coefficients can be truncated or heavily quantized with minimal impact on the reconstruction.

The *Haar wavelet basis* is the simplest possible orthonormal wavelet basis, proposed already in 1909 by Alfréd Haar. It is defined by the following scaling and mother wavelet functions in one dimension:

$$\phi(t) = \begin{cases} 1, & 0 \le t < 1, \\ 0, & \text{otherwise}, \end{cases} \qquad \psi(t) = \begin{cases} 1, & 0 \le t < 1/2, \\ -1, & 1/2 \le t < 1, \\ 0, & \text{otherwise}. \end{cases} \tag{117}$$

In the following, we limit ourselves to the unit square, $[0,1)^2$, and use the common *non-standard decomposition* [136] to extend the Haar wavelet basis to two dimensions:

$$\phi(x,y) = \phi(x)\phi(y), \quad \text{and} \quad \begin{cases} \psi^{(1)}(x,y) &= \psi(x)\phi(y), \\ \psi^{(2)}(x,y) &= \phi(x)\psi(y), \\ \psi^{(3)}(x,y) &= \psi(x)\psi(y). \end{cases} \tag{118}$$

The scaled and translated Haar wavelet functions over the unit square, are given by applying Equation 116 along each dimension, as follows:

$$\psi^{(i)}_{l,\mathbf{t}}(\mathbf{x}) = 2^l\, \psi^{(i)}\left(2^l\, \mathbf{x} - \mathbf{t}\right), \quad \text{for} \begin{cases} l \ge 0, \\ 0 \le \mathbf{t} < 2^l. \end{cases} \tag{119}$$

We note that each such wavelet function has square support of size $2^{-l} \times 2^{-l}$ in the unit square. The three types of wavelet functions, $\psi^{(i)}$, $i = \{1,2,3\}$, represent the horizontal, vertical and diagonal differences, respectively. Since each wavelet function is split by its center, we note that the coefficients up to (and including) a scale l, are enough to *exactly* represent a discrete image, $F(\mathbf{x})$, at level $l+1$. Also note that the coefficient for the scaling function represents the integral of F over the unit square, which is the same as $F^0_{0,0}$ before. The Haar wavelet basis is therefore perfectly suited for hierarchical sample warping, as it allows efficient compression of the involved functions, as well as hierarchical reconstruction. The only piece missing is the theory for multiplication of two compressed wavelet representations, which we will look at next.

General Wavelet Product We use Ψ_i to denote the Haar basis functions up to a level $m-1$, which are enough to represent the full-resolution image at $l = m$:

$$F(\mathbf{x}) = \sum_i F_i\, \Psi_i(\mathbf{x}), \quad \text{where} \quad \Psi = \left\{\phi, \psi^{(1)}_{(0,\mathbf{0})}, \ldots, \psi^{(i)}_{(l,\mathbf{t})}, \ldots\right\}, \tag{120}$$

where F_i are the coefficients of the wavelet representation of F. There are the same number of coefficients and basis functions as there are pixels in the full-resolution image, i.e., 2^{2m}, and we number these $i = 0, \ldots, 2^{2m}-1$. In our case F is a product of two terms, $F = GH$ (Equation 112), which are represented in the same wavelet basis. The product can be written:

$$F(\mathbf{x}) = G(\mathbf{x})H(\mathbf{x}) = \left(\sum_j G_j \Psi_j(\mathbf{x})\right)\left(\sum_k H_k \Psi_k(\mathbf{x})\right). \tag{121}$$

The inner product of $F(\mathbf{x})$ and a basis function gives the corresponding coefficient, which using Equation 121 expands to:

$$\begin{aligned} F_i &= \int \Psi_i(\mathbf{x}) F(\mathbf{x}) d\mathbf{x} \\ &= \int \Psi_i(\mathbf{x}) \left(\sum_j G_j \Psi_j(\mathbf{x})\right)\left(\sum_k H_k \Psi_k(\mathbf{x})\right) d\mathbf{x} \\ &= \sum_j \sum_k C_{ijk} G_j H_k \quad \text{where} \quad C_{ijk} = \int \Psi_i(\mathbf{x}) \Psi_j(\mathbf{x}) \Psi_k(\mathbf{x}) d\mathbf{x}. \end{aligned} \tag{122}$$

The set of coefficients C_{ijk} are called *tripling coefficients*. Note that these equations hold in the general case, for any dimensionality and choice of basis.

Ng et al. [97] analyzed the two-dimensional case, and showed that the set of tripling coefficients is sparse if the two images G and H have sparse representations in the Haar wavelet basis. Their Haar tripling coefficient theorem showed that it is possible to directly compute the coefficients for the wavelet representation of the product, F, using linear programming and a simple set of rules. The sparsity comes from the fact that if G and H are each compressed to a small set of nonzero coefficients, the majority of the tripling coefficients will be zero. I will not reproduce the theorem here, but for further details, refer to the original paper [97] and my Master's thesis [20]

In the technique presented in Paper I, we apply the two-dimensional wavelet product, which was originally developed for interactive relighting applications, to our application of Monte Carlo rendering. In particular, we let G and H be precomputed, compressed representations of environment map lighting and the reflectance function, respectively. The sampling proceeds by hierarchically computing the wavelet coefficients, F_i, for the product and at each step, using these to locally reconstruct the product distribution. This is used to evaluate the conditional probabilities necessary for sample warping, as explained in Section 4.2.1. Figure 19 shows an image rendered using our technique.

In the paper, we also present a *generalized tripling coefficient theorem*, which allows F and G to have n dimensions each, while H has m dimensions, where $0 \leq m \leq n$. Note that the original theorem only worked for $n = m = 2$. We show that the tripling coefficients, C_{ijk}, can be expressed as products of *one*-dimensional tripling coefficients and new so-called *coupling* coefficients. We also show a proof-

Figure 19: *Image rendered using the technique presented in Paper I, which uses the wavelet product of lighting and reflectance for importance sampling.*

of-concept implementation of the generalized theorem for the case $n = 4, m = 2$, i.e., an 4D = 4D $\times$ 2D wavelet product. For details, refer to Paper I.

4.3.2 Fast Wavelet Product

In Paper II, we note that the general framework for wavelet products is unnecessarily flexible for our purposes. Equation 122 allows the wavelet coefficient at any place in the hierarchy to be computed, but in practice, we are only interested in these for hierarchical reconstruction of the product function. The goal is to hierarchically evaluate $F^l_{\mathbf{t}}$, i.e., the average of the product over a square $\mathbf{s} = (l, \mathbf{t})$. By inserting the wavelet expansion of the product $F(\mathbf{x}) = G(\mathbf{x})H(\mathbf{x})$ (Equation 121) into the expression for the image average, $F^l_{\mathbf{t}}$, at the coarsest level (c.f., Equation 101), we note that:

$$\begin{aligned} F^0_{0,0} &= \int F(\mathbf{x})d\mathbf{x} \\ &= \int \left(\sum_i G_i \Psi_i(\mathbf{x})\right)\left(\sum_j H_j \Psi_j(\mathbf{x})\right) d\mathbf{x} \\ &= \sum_i \sum_j G_i H_j \int \Psi_i(\mathbf{x}) \Psi_j d\mathbf{x} \;=\; \sum_i G_i H_i. \end{aligned} \tag{123}$$

The last step is due to the orthonormality of the Haar basis, i.e., the inner product of any two basis functions with different indices is zero. Hence, the image at the coarsest level is computed simply by summing up the products of all pairs of wavelet coefficients with the same index. The set of nonzero terms is sparse, as it is only when both G_i and H_i are nonzero that their product has an influence. The above equation can alternatively be written as:

$$F_{0,0}^{0} = G_{0,0}^{0} H_{0,0}^{0} + \sum_{i=1}^{2^{2m}} G_i H_i, \qquad (124)$$

i.e., a product of the image averages plus the sum of all wavelet coefficients for basis functions under the support of these.

Intuitively, this relationship generalizes to any smaller region, $\mathbf{s} = (l, \mathbf{t})$, of the domain due to the hierarchical nature of the wavelet basis; a localized basis over $\mathbf{s}$ is formed by appending a scaling function, $\phi_{l,\mathbf{t}}$, and considering all wavelet functions under the support of $\mathbf{s}$. For details, refer to the derivation in the paper. The result is that the product image can be computed directly at any level as:

$$F_{\mathbf{t}}^{l} \; = \; G_{\mathbf{t}}^{l} H_{\mathbf{t}}^{l} \; + \; 2^{2l} \sum_{\{i \,|\, \Psi_i \in \mathbf{s}\}} G_i H_i, \qquad (125)$$

where the sum is over the indices of all wavelet functions that exist at a level $l' \geq l$, and are under the support of the current square, $\mathbf{s}$. The scale factor 2^{2l} comes from the area of $\mathbf{s}$ (c.f., Equation 101). We refer to this simplification as the *fast wavelet product*, as it represents a specialized form of the general wavelet product that is better suited for fast hierarchical evaluation of the product image.

To generate samples distributed according to $F(\mathbf{x})$, we start at the coarsest level and recursively evaluate Equation 125 where needed. In practice, looking at a level l, the terms G and H are *separately* reconstructed from their respective wavelet coefficients to find the averages $G_{\mathbf{t}'}^{l+1}$ and $H_{\mathbf{t}'}^{l+1}$ for the four quadrants at the next level, $l+1$. To each of these, the sums of corresponding wavelet coefficients are added in order to compute $F_{\mathbf{t}'}^{l+1}$. These are then used to evaluate the conditional probabilities for sample warping (Equation 105) as before. The implementation of these operations can be made very efficient as the wavelet coefficients are naturally arranged in a hierarchical sparse *tree structure*. Evaluating a partial sum $\sum G_i H_i$ at any node in the tree, becomes a process of multiplying and accumulating all nonzero wavelet coefficients over the children nodes. The same wavelet coefficients are used throughout, just different subsets of them, so all partial sums can be computed and cached at startup in a single sparse tree traversal.

4.3.3 Sparse Quadtree Product

The fast wavelet product introduced in the previous section, is an efficient way of sampling the product of two functions stored in the Haar basis. In practice, however, in order to use this method, all the involved data must be precomputed and transformed into the wavelet basis prior to rendering. This can be a severe limitation. For instance, precomputing wavelet representations of the lighting and all reflectance functions in a realistic scene is impractical.

To avoid these problems, we suggest a method in Paper II that allows a hierarchical approximation of the reflectance function to be constructed on-the-fly (see Section 4.4). The approximation is represented in the form of a *sparse* image, i.e., a hierarchical image with some larger regions ($l < m$) being piecewise constant. We have already seen examples of such sparse images in the form of the product function reconstructed from compressed Haar wavelet representations. In that case, the compression removes details in areas of low variation, causing unimportant regions to be piecewise constant. The difference is that we now directly compute the sparse image, without going via a wavelet basis.

Sparse images are naturally encoded in a sparse *quadtree* structure, i.e., the image averages, $F^l_{\mathbf{t}}$, are hierarchically arranged in a sparse tree, where the leaves represent squares of constant value. We note that the product of two sparse quadtrees, $F(\mathbf{x}) = G(\mathbf{x})\,H(\mathbf{x})$, can be efficiently computed by traversing the trees of G and H in parallel. Assume $G(\mathbf{x})$ is constant over a square $\mathbf{s} = (l, \mathbf{t})$ with a value $G^l_{\mathbf{t}}$. Then, the same square in the product image can be computed as (c.f., Equation 101):

$$\begin{aligned} F^l_{\mathbf{t}} &= 2^{2l} \int_{\mathbf{s}} G(\mathbf{x}) H(\mathbf{x}) d\mathbf{x} \\ &= 2^{2l} G^l_{\mathbf{t}} \int_{\mathbf{s}} H(\mathbf{x}) d\mathbf{x} \;=\; G^l_{\mathbf{t}} H^l_{\mathbf{t}}, \qquad \text{if } G(\mathbf{x}) = G^l_{\mathbf{t}} \text{ for } \mathbf{x} \in \mathbf{s}. \end{aligned} \tag{126}$$

The same applies if H is locally constant. Hence, we note that the image averages can be directly multiplied whenever *at least one* of the two terms is a leaf node (and hence piecewise constant). This conclusion is not groundbreaking, but it leads to a very practical algorithm for computing the product distribution from sparse images. Essentially, the two quadtrees are recursively traversed in parallel, and all pairs of nodes with at least one leaf are multiplied. The final product image is hierarchically computed at the back of the recursion by averaging over the results for the four children nodes at each step (Equation 101).

In our implementation of the algorithm in Paper II, the evaluated sparse product coefficients are cached along the way and used for hierarchical sample warping in a second step. Additionally, our sparse approximation of the reflectance function is constructed alongside the evaluation of the product in a single traversal. For details, we refer to Section 4.4.2 and the pseudocode in our paper.

The complexity of the sparse quadtree product depends on the number of pairs of leaf-level nodes in the two quadtrees. If one of the trees is very sparse, we can afford the other to be non-sparse. This fact is exploited in our papers by the use of full-resolution, unapproximated lighting together with a very sparse approximation of the reflectance, allowing sampling according to $p(\omega) \propto L(\omega)\widetilde{R}(\omega)$, as mentioned in Section 4.1.1. In other applications, it may be more appropriate to let both terms be moderately sparse.

The sparse quadtree representation is generally less sparse than an equivalent wavelet representation, so the best choice of algorithm depends on the use case. If the data is available beforehand, it is preferable to compress it in the Haar wavelet basis and use the fast wavelet product. For rendering problems, however, we are often interested in evaluating a large number of integrals with different integrands. In this case, extensive precomputation is not efficient. Another advantage of our sparse quadtree product is that it can be easily extended to handle products of more than two terms. Equation 126 generalizes to products on the form $F = F_1 F_2 \dots F_n$, as long as *at most one* term is not a leaf. In Paper III, we show an example of this by computing the *triple* product of lighting, approximate reflectance, and approximate visibility, i.e., $L\widetilde{R}\widetilde{V}$, for use as a control variate term. It should be noted that this example is a somewhat favorable case, as $\widetilde{V} = 0$ in fully occluded regions, which allows the tree traversal to be efficiently pruned in those regions.

4.3.4 Approximate Product

Finally, in Paper IV we use the hierarchical rejection sampling method (see Section 4.2.2) instead of sample warping. To make the rejection test faster, each generated candidate sample is tested against a hierarchical *approximation* of the product distribution, rather than the exact product. We consider the case of a product with multiple terms, $F = F_1 F_2 \dots F_n$, and test samples against an approximation $\widetilde{F}(\mathbf{x}) \approx F(\mathbf{x})$. In the paper, hierarchical representations of each of the terms F_i are precomputed. Then, at sample generation time, the product over a specific region, $\mathbf{s} = (l, \mathbf{t})$, is simply approximated as the product of the averages over $\mathbf{s}$ of the individual terms, i.e., $F^l_{\mathbf{t},i}$ for the i^{th} image, as follows:

$$\widetilde{F}^l_{\mathbf{t}} = \prod_i F^l_{\mathbf{t},i}. \tag{127}$$

This is an approximation, since in general:

$$\widetilde{F}^l_{\mathbf{t}} = \prod_i \left(2^{2l} \int_{\mathbf{s}} F_i(\mathbf{x}) d\mathbf{x} \right) \neq 2^{2l} \int_{\mathbf{s}} \prod_i F_i(\mathbf{x}) d\mathbf{x} = F^l_{\mathbf{t}}, \quad \text{if } l < m. \tag{128}$$

However, the error of the approximation in Equation 127 gets smaller at finer resolutions, under the assumption that the terms F_i are locally smooth. Hence, we want to evaluate the approximation at finer resolutions in regions where $F(\mathbf{x})$ is large, while in low-importance regions, a cruder approximation (lower l) is acceptable.

A couple of different sampling strategies are discussed in the paper. In the main algorithm, candidate samples are hierarchically generated and tested against the locally estimated upper bound, $\overline{F}^l_{\mathbf{t}}$. If a sample passes this test, the node is subdivided and the upper bound is refined by recomputing it at the finer level, $l+1$. At each step, the sample is placed in one of the four children nodes, and new candidate samples are generated for the remaining nodes. The samples are tested against the refined upper bounds, and the process is repeated recursively. Whenever a single sample remains, i.e., when no new candidate samples can be generated without exceeding the local upper bound, the recursion is terminated. At that point, the product approximation $\widetilde{F}^l_{\mathbf{t}}$ is evaluated, and the final rejection test performed.

In effect, the product distribution is more accurately approximated in regions of high importance, i.e., regions that are subdivided to a fine level. In other regions, where the recursion terminates early, the approximation is evaluated at coarser nodes. The final $\widetilde{F}(\mathbf{x})$ from which samples are drawn, is thus made up of leaf nodes of different size and accuracy. During the sampling process, its integral $\int \widetilde{F}(\mathbf{x})d\mathbf{x}$ is additionally computed, and later used for normalizing the probability associated with each sample.

Discussion The approximate product method is attractive since it automatically adjusts the accuracy of the product approximation to important regions. Additionally, since the approximation is computed on-the-fly, the full-resolution representations of the individual terms, F_i, can be used. However, it should be noted that, although the approximation error is upper bounded, the method gives no other guarantees on its magnitude. The need for precomputing each product term, F_i, is also a drawback in certain applications.

In comparison, the wavelet and quadtree products rely on sparse, approximate representations of F_i, while the product is evaluated exactly. It may therefore be easier to predict the accuracy of the final probability distribution. The exact product also makes it possible to rely on sample warping. Hence, these methods do not need to generate and test a large number of candidate samples. However, some amount of precomputation is necessary, even though we show examples of sparse product terms that are created on-the-fly (i.e., reflectance and visibility). In all algorithms, the evaluation of the product is automatically directed towards regions where it is needed, avoiding unnecessary work in unsampled parts of the function.

For results and further comparisons, refer to our original publications (Paper I–IV). We will now look at different ways of computing the individual terms, F_i, in our example applications.

4.4 Analysis and Approximation of Reflectance

In the previous sections, we have introduced methods for sampling and evaluating a product distribution. The algorithmic innovations that allow this are not tied to any particular use case, but the motivating application in our work has been the evaluation of the rendering equation. In our formulation, one of the terms in the product is the *reflectance function*, R, defined in Equation 89. Each surface position and outgoing light direction, ω_o, picks out a two-dimensional *slice* of the reflectance function, $R(\omega_i)$, which describes the BRDF times the cosine term over all incident directions, ω_i. For convenience, we map directions on the (hemi)sphere to the unit square. The goal is thus to find the slice $R(\mathbf{x})$, where $\mathbf{x} \in [0,1)^2$, for a given surface point and outgoing direction. In this section, we will describe the different strategies that we have developed for computing appropriate approximations, $\widetilde{R}(\mathbf{x})$, for this purpose.

Note that even though the product distribution is a scalar function, we generally want to use full color representations of its individual terms. The reason is that, for example, multiplying a red material with green lighting should result in a low or zero contribution, and a low sampling density as a result. The product thus has to be computed in color, and its luminance is used to define the final probability distribution. We are therefore interested in RGB representations of $R(\mathbf{x})$.

4.4.1 Precomputed Tabulated Representations

For the wavelet-based sampling approach used in Paper I, we precompute tabulated reflectance functions in the Haar wavelet basis. The outgoing direction, ω_o, is discretized into a 2D set of fixed directions. For each such outgoing direction, a two-dimensional reflectance function is stored in the wavelet basis. These are compressed by discarding all coefficients with an absolute value below a certain threshold. The threshold is chosen so that a fixed percentage of the coefficients are retained. In our research, we found that most materials can be accurately represented using only around 2% of the wavelet coefficients. This shows that the Haar basis efficiently concentrates the information to a small number of coefficients. Due to the efficient compression achieved, we could use realistic measured materials [88], many of which were anisotropic. For the results presented in the paper, we tabulate the reflectance function over 16^2–32^2 discrete outgoing directions, using a resolution of 64^2–128^2 for each slice (i.e., 82–328 wavelet coefficients per slice at 2% sparsity). The total memory consumption is only 0.3–5.0 MB per material, but the reported precomputation times are the range of a few minutes (using extensive super-sampling to improve the quality).

In the method proposed in Paper IV, we take a similar approach, but store *uncompressed* tabulated reflectance functions. For each two-dimensional slice, we store an image hierarchy of averages, $\widetilde{R}^l_{\mathbf{t}}$, alongside the maxima, $\overline{R}^l_{\mathbf{t}}$, which are needed for hierarchical rejection sampling. To reduce the memory consumption, we limit the implementation to support only isotropic BRDFs (c.f., Section 2.3.1) by storing a tabulated reflectance function for each of a discrete set of outgoing polar angles, θ_o. Each slice of the reflectance function is stored in the local hemispherical frame centered around the surface normal, using the HEALPix mapping (Section 4.6.1) with a resolution of 1536 to 24576 data points per slice, depending on the specularity of the material. The precomputation times are shorter than in the method of Paper I, but still in the range of 0.4 to 5.3 seconds per material. Additionally, each tabulated (isotropic) reflectance function occupies 2.6–38.5 MB of memory (in full floating-point RGB values).

The main advantage of using precomputed representations of the reflectance function is that very well-distributed samples can be achieved. The error introduced by the quantization and compression of $R(\mathbf{x})$ is easy to measure, and it can be made arbitrarily small by increasing the resolution of $\widetilde{R}$ and/or reducing the compression. For applications where the same materials can be reused over many frames, such as product visualization or rendering of animations, precomputing tabulated materials may be a good idea. However, the long precomputation times and the memory consumption present major drawbacks, as a realistic scene may contain hundreds or thousands of different materials. The reported timings could be sig-

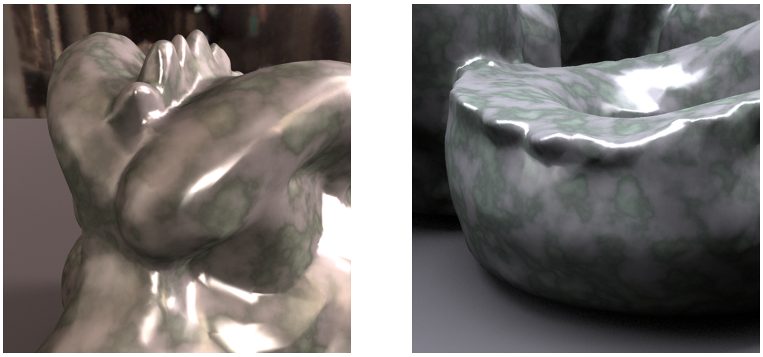


Figure 20: *Example of a reflectance function that is procedurally defined by a shader program. As many shader inputs may not be known beforehand, precomputing a tabulated representation of the reflectance is difficult. The images are rendered using the technique presented in Paper II.*

nificantly reduced by using modern parallel implementations running on a multi-threaded CPU or graphics processor, but even then, the precomputation needed for a realistic scene is non-negligible.

Additionally, in real applications, the reflectance functions are often programmatically defined by *shader* programs. Figure 20 shows a simple example of this. Depending on the number of shader inputs used, the shaders may be of too high dimensionality to be realistically precomputed. Changing the material parameters or editing the BRDF, usually also requires the tabulated representations to be re-computed, making interactive changes difficult. Therefore, finding methods that do *not* require preprocessing has been one of our primary goals. Next, we will look at two different approaches.

4.4.2 Dynamically Sampled Approximation

In the method introduced in Paper II, we dynamically construct an approximation of the reflectance function, $\widetilde{R}(\mathbf{x})$, in order to avoid the problems associated with precomputation-based approaches. Our approximation is based on a small set of point samples, $R(\mathbf{x}_i),\ i = 1,\ldots,N$, which are computed on-the-fly. Conceptually, the point samples are used to reconstruct a continuous representation of the reflectance function. This is then discretized to a sparse quadtree representation, i.e., a piecewise constant function with nodes of varying size, which is used for sampling purposes.

The reconstruction of a continuous reflectance function is a scattered data interpolation problem, for which many advanced numerical techniques exist [157]. Our application is unusual in that we can only afford to spend a minimal amount of time on the reconstruction, and the result is to be represented as a sparse quadtree.

The approach we describe in the paper combines the reconstruction and discretization steps. In practice, we hierarchically subdivide the set of samples, $R(\mathbf{x}_i)$, until only a *single* sample per node remains. These nodes define the leaves of the hierarchical approximation, $\widetilde{R}^l_{\mathbf{t}}$, which are averaged at the back of the recursion to form the complete sparse quadtree representation. In doing this, we additionally evaluate the sparse quadtree product (Section 4.3.3) in the same traversal, multiplying $\widetilde{R}$ with an uncompressed lighting term, L. The results are cached along the way, which makes the subsequent sampling using sample warping extremely simple. This process is performed on-the-fly just prior to the sampling of the product distribution. Hence, all shader inputs and material parameters are known, which means we can support arbitrary shaders and spatially varying materials.

A number of factors affect the accuracy of the resulting approximation, $\widetilde{R}(\mathbf{x})$, and hence the amount of variance reduction achieved. First, if a large number of point samples from R are taken (i.e., a large N), the probability of finding important features is increased. However, there is a tradeoff between how much effort to spend on constructing the approximation, versus how many samples can be afforded in the final sampling step, where $\widetilde{R}(\mathbf{x})$ is used for product sampling. The goal is to optimize the overall quality for a given fixed total computation time. In our paper, we describe a simple variational analysis we performed to find a good work distribution between the two tasks.

Second, the choice of samples, $R(\mathbf{x}_i)$, has a large impact on the result. For diffuse materials, a uniform or cosine-weighted sampling over the hemisphere is adequate, but for glossy materials it is important to accurately capture the specular peak(s). For this reason, we aim to use importance sampling to draw a set of random sampling directions distributed according to $p(\omega) \propto R(\omega)$. The reflectance function is evaluated for the chosen directions, and the result is mapped to the unit square to give $R(\mathbf{x}_i)$. Sampling exactly according to $R(\omega)$ may not be possible depending on the BRDF model used, but an approximate probability distribution suffices if it accurately captures the peaks of the BRDF. Note that we do *not* need to compute the probability associated with each of these samples, as they are never used in a Monte Carlo estimator, but only for approximating the (unknown) reflectance function. In our implementation, we rely on existing methods for numerical inversion of the cdf for some well-known BRDFs, e.g., the Phong and Ward models (see Section 2.3.2). Hence, our method can be said to be *semi-automatic*; it handles procedurally or programmatically defined shaders, but we rely on the rendering system or the user to implement a (possibly approximate) importance sampling strategy for the reflectance function. The generated samples are then used to dynamically build the quadtree approximation that is required for efficient product sampling and control variate techniques.

Although most rendering systems already use BRDF importance sampling in some form, we would ultimately like to remove this requirement in order to support completely arbitrary shaders. In the work presented in Paper VI, we have explored one such technique, which will be described next.

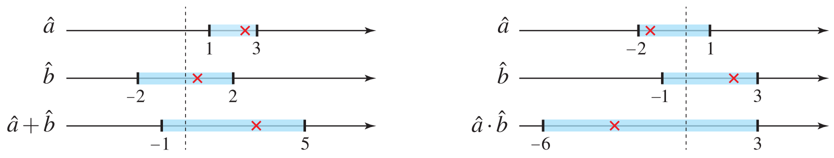


Figure 21: *In interval analysis [94], the value of each variable is represented by a conservative interval, and the arithmetic operations are defined to operate on such intervals. Here, two simple examples are shown (addition and multiplication). The red marks indicate one potential value of each variable.*

4.4.3 Optimized Interval Analysis

In Paper VI, we assume the user specifies a completely arbitrary reflectance function by writing a shader program. The program consists of a sequence of instructions, which compute the reflectance, R, given a specific surface point, incident and outgoing directions (c.f., Equation 89). A real scene may consist of hundreds or thousands of such shader programs. Manually computing useful approximations or upper bounds of their generated reflectance functions, as required for importance sampling, is both tedious and error-prone. Additionally, it requires advanced mathematical skills. Therefore, we want to apply automatic methods for *analyzing* the shader program, in order to construct appropriate importance functions.

Bounded arithmetics, such as *interval analysis* [94], is an example of one such tool. In our paper, we present several important optimizations that allow interval analysis to be applied to shader analysis much more efficiently than before. Before introducing our optimizations, let me give a brief review of interval analysis and discuss how it applies to our hierarchical sampling methods.

Interval Analysis of Shaders The idea behind bounded arithmetics is to compute conservative bounds on the result of a computation, instead of a single value. Using interval analysis, the bounds are represented as a conservative interval, which is guaranteed to enclose the exact result. We use the following notation to denote an interval:

$$\widehat{a} = [\underline{a}, \bar{a}] = \{x \in \mathbb{R} \mid \underline{a} \leq x \leq \bar{a}\}, \tag{129}$$

where $\underline{a}$ and $\bar{a}$ are the lower and upper boundaries of the interval $\widehat{a}$, respectively.[11] In order to bound a computation, each arithmetic operation is redefined to operate on intervals. As a simple example, consider the addition of two intervals:

$$\widehat{a}+\widehat{b} = \left[\underline{a}+\underline{b},\ \bar{a}+\bar{b}\right]. \tag{130}$$

The result of each mathematical operation is a new interval, which is given as input to the next operation, and so on. Figure 21 shows two examples of simple

[11]The values $\pm\infty$ are allowed and specify open-ended intervals.

interval operations. The bounds grow as they are propagated, and the end result is conservative bounds for the result of a chain of computations. Interval analysis is thus widely used in scientific and engineering applications, for example, to bound the rounding errors in numerical computations. A few published methods have similarly applied bounded arithmetics to compute bounds on the result of shader evaluations for various purposes [48, 55, 56, 59]. Recently, Velázquez-Armendáriz et al. [149] applied the same machinery for analyzing shaders in Monte Carlo rendering, which is closely related to our work.

There are several different ways in which shader analysis based on interval analysis can be applied to our sampling methods presented in Section 4.2 and 4.3. Given a range of incident directions, $\widehat{\omega}_i$, mapped to the square, and bounds for all varying shader inputs over this set of directions, one can compute conservative bounds on the reflectance function over any region:

$$\widehat{R}^l_{\mathbf{t}} = \left[\underline{R}^l_{\mathbf{t}},\, \overline{R}^l_{\mathbf{t}}\right], \qquad \text{where} \left\{ \begin{array}{lcl} \underline{R}^l_{\mathbf{t}} & \leq & \min\limits_{\mathbf{x}\in\mathbf{s}} R(\mathbf{x}), \\ \overline{R}^l_{\mathbf{t}} & \geq & \max\limits_{\mathbf{x}\in\mathbf{s}} R(\mathbf{x}). \end{array} \right. \tag{131}$$

In practice, a *bounding shader* [55, 59] is compiled based on the original shader, where each instruction has been replaced by a sequence of instructions performing the equivalent interval operation.

The range of $\widehat{R}^l_{\mathbf{t}}$ for a specific shader and region $\mathbf{s} = (l, \mathbf{t})$, i.e., how tight the computed interval is over $\mathbf{s}$, depends on the extent of the region and the ranges of the shader inputs over this region. The bounds will generally be very loose at coarse levels, but they get progressively tighter at finer levels. Hence, the bounds may be used to construct a hierarchical approximation of the reflectance, e.g., using the midpoints of the intervals:

$$\widetilde{R}^l_{\mathbf{t}} = \left(\underline{R}^l_{\mathbf{t}} + \overline{R}^l_{\mathbf{t}}\right)/2, \tag{132}$$

and refining the approximation wherever the width of the interval, i.e., $\overline{R}^l_{\mathbf{t}} - \underline{R}^l_{\mathbf{t}}$, is large. Interval analysis thus gives a fully *automatic* way of determining which parts of the reflectance function need to be approximated at finer resolutions (as opposed to relying on existing importance sampling techniques as in Section 4.4.2). Alternatively, we may compute the upper bounds only, $\overline{R}^l_{\mathbf{t}}$, and use these together with the hierarchical rejection sampling method introduced in Section 4.2.2. This method will also automatically refine the function where needed. It can be used for sampling product distributions if the upper bounds of the other terms are also available, which should not be a problem for the case of environment map lighting.

Compiler Optimizations There are two main culprits of the general approach outlined above. First, the bounds computed using interval analysis, quickly grow very large under certain circumstances. This can be addressed through the use of higher-order bounded arithmetics, which track the dependencies between variables. We have left this extension for future work. Second, the cost of executing a bounding shader is often considerably higher than executing the original shader, as each instruction has been replaced by several others. This causes the hierarchical construction of importance functions to be quite expensive, which partially defeats the purpose. To lower the cost, we have developed several advanced compiler optimizations targeting interval analysis, which are presented in more detail in Paper VI. The core idea is to extract and track *compile-time* bounds information, and use this to optimize the generated code.

The approach is most easily explained by an example. Consider the multiplication of two intervals, which in the general case is given by:

$$\widehat{a}\cdot\widehat{b} \;=\; \left[\min(\underline{a}\underline{b},\underline{a}\bar{b},\bar{a}\underline{b},\bar{a}\bar{b}),\ \max(\underline{a}\underline{b},\underline{a}\bar{b},\bar{a}\underline{b},\bar{a}\bar{b})\right], \tag{133}$$

i.e., since one or both of the intervals $\widehat{a}$ and $\widehat{b}$ may be negative or overlap zero, we have to take all combinations of their lower and upper boundaries into account. For instance, $[-3,2]\cdot[-4,3] = [-9,12]$. In the general case, this leads to an arithmetic cost of 10 scalar instructions (4 mul and 6 min/max) for one interval multiplication. However, if we can determine the *maximum* possible range of each operand beforehand, e.g., that an interval is positive, $\widehat{a}\in[0,\infty]$, then more efficient implementations are possible. For instance:

$$\widehat{a}\cdot\widehat{b} = \begin{cases} \left[\min(\underline{a}\underline{b},\bar{a}\underline{b}),\max(\underline{a}\bar{b},\bar{a}\bar{b})\right] & \text{if } \widehat{a} \geq 0, \\ \left[\underline{a}\underline{b},\bar{a}\bar{b}\right] & \text{if } \widehat{a},\widehat{b} \geq 0. \end{cases} \tag{134}$$

The same can be applied if we know that one of the operands can only take scalar values. Similar optimizations are possible for other instructions. Refer to Paper VI for further examples.

Using a technique we call *static bounds analysis*, bounds information is propagated through the shader code at compile time, in order to find the largest possible range each instruction operates on. In practice, this is done by "executing" the code at compile time using interval analysis, applying the widest possible interval to each input. The result is a conservative range for the possible values of each variable in the program, which allows the compiler to choose the most inexpensive implementation of each interval operation in the code generation phase.

The static bounds can come from several different sources. One example is the data type of variables, e.g., shading languages often support signed/unsigned normalized types, which are limited in range. Another example is domain-specific knowledge, e.g., that colors are positive or that the length of a normalized vector is one. In addition, user annotations may be used to give the compiler additional information about computations or shader parameters. Many instructions also naturally limit the range of the result. For example, the outcome of an absolute value or square root operation is always positive (if it exists). Such information is propagated forward, and allows subsequent interval operations to be optimized.

A related technique that we have developed is *valid range analysis*. Here, the assumption is made that, at runtime, each instruction will receive valid input. For example, the input to a square root is assumed to be positive for the shader to generate valid results. This type of bounds information is propagated *backwards* through the code to allow further optimizations. Note that handling this correctly, and interval analysis in general, requires robust treatment of floating-point exceptions, such as Not-a-Number (NaN). This is discussed in detail in Paper VI, together with many other practical aspects. The final result is that, for computation of single-sided (upper) bounds, 42–44% of the instructions can on average be removed using our optimizations. Note that this is *in addition* to those normally eliminated by standard compiler optimizations. For double-sided intervals (lower and upper bounds), the savings are 24–27% on a range of example shaders of varying complexity. This is an important step towards using interval analysis for Monte Carlo sampling, as it significantly reduces the cost of hierarchically bounding and approximating unknown functions.

4.5 Estimation of Visibility

In many rendering problems, we are interested in computing the reflected radiance due to light arriving from a specific surface or light source in the scene. Under this assumption, the rendering equation involves a *visibility* term, $V \in [0,1]$, which denotes the visibility between our integration point and the other surface (c.f., Equation 88). The prime example is the evaluation of direct illumination under distant environment map lighting (Section 2.4.3), which is an important application of our research.

We have explored two different ways of approximating the visibility function, in order to use it for variance reduction purposes. Paper IV presents a method using an inner-conservative visibility approximation, while Paper III introduces a more general solution. I will now briefly describe the two methods.

4.5.1 Conservative Bounding Geometry

As mentioned in Section 4.1.2, the method presented in Paper IV performs *triple* product importance sampling, including all three of the lighting, reflectance, and visibility terms in the sampling. In this case, the visibility approximation, $\widetilde{V}(\omega_i)$, must be inner-conservative, i.e., a direction must never be reported occluded if there is a chance it is visible.

The approach we take is to compute a set of bounding spheres, which are placed so that they are fully inside the opaque scene geometry. The bounding spheres are computed as a preprocess before the rendering starts, using a method similar to Wang et al.'s [151] work. To make the visibility queries faster, we also aggregate the bounding spheres into a bounding volume hierarchy (BVH). The method is not limited to using bounding spheres, but we chose to work with this shape in our proof-of-concept implementation, as the sphere has a very simple geometry and existing tools could be used for computing the bounding geometry.

At runtime, the hierarchy of bounding spheres is used to quickly construct a conservative visibility approximation, $\widetilde{V}(\mathbf{x})$, where $\mathbf{x} \in [0,1]^2$ represents directions on the sphere mapped to the unit square. Note that the visibility changes throughout the scene, so this visibility approximation has to be recomputed at each new integration point in order to be conservative. To make this reasonably efficient, we traverse the image hierarchy and the sphere BVH in parallel. Each square, $\mathbf{s} = (l, \mathbf{t})$, in the image represents a frustum of directions, which is hierarchically tested against the BVH. If the frustum does not intersect any bounding spheres, the square is directly marked as visible ($V = 1$). On the other hand, if it is fully blocked by a *leaf* node in the BVH, then all directions in $\mathbf{s}$ are guaranteed to be occluded ($V = 0$). In case the visibility cannot be determined, the square is recursively subdivided, and the process repeated.

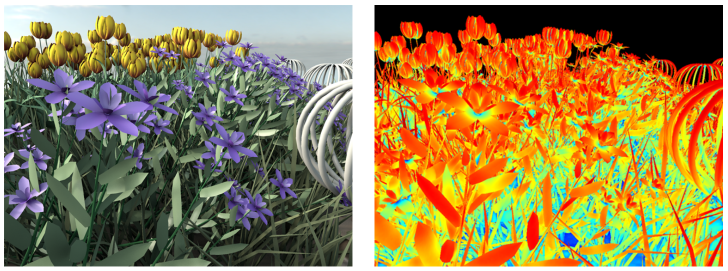
In practice, since $\widetilde{V}(\mathbf{x})$ is recomputed for *each* new surface point, the method is limited to using low resolution and relatively few bounding spheres. In addition, it only works well in cases where the scene geometry can be accurately approximated using bounding spheres, or other simple shapes. These concerns are addressed in our second method, which will be described next.

4.5.2 Visibility Caching

In Paper III, we use the visibility approximation as a *control variate* term, as described in Section 4.1.3. This adds a lot of flexibility, as the visibility does not have to be conservatively approximated – any approximation that is correlated to the true function will work. Therefore, it is possible to cache and reuse $\widetilde{V}(\mathbf{x})$ over many surface points. Our approach is conceptually similar to previous techniques for sparse sampling and caching of irradiance [153] and radiance [70]. However, instead of storing relatively low-frequency indirect illumination, we sparsely sample and cache the visibility function, which contains sharp discontinuities and high-frequency effects. We call this *visibility caching*.

Apart from the sampling method itself, i.e., using control variates, one of our main contributions is a thorough analysis of the visibility function. This was used to find an empirically-based heuristic for controlling the weighting and insertion of new cache records. Each cache record is stored as a bit mask with 2^{2m} bits, which encodes a discrete image of $2^m \times 2^m$ pixels, representing the binary visibility function. In practice, resolutions of $m = 5$ or $m = 6$ were used, i.e., 1024–4096 visibility samples per cache record. To allow hierarchical lookups, the bits are stored along a space-filling curve in the Morton order [95]. Hence, the visibility approximation at coarser levels, $l < m$, can be found by counting bits in consecutive segments of $2^{2(m-l)}$ bits, without having to explicitly store the hierarchical representation. For details, refer to our paper.

In our analysis, we view the sampled approximations, $\widetilde{V}$, as dependent random observations of the underlying unknown visibility function. The statistical *correlation* between a large number of such approximations was measured by computing the correlation coefficient (Equation 44) between each pair. The results are presented both in terms of the spatial variation for visual inspection, and as functions of distance and normal directions. Examples are shown in Figure 22. Based on



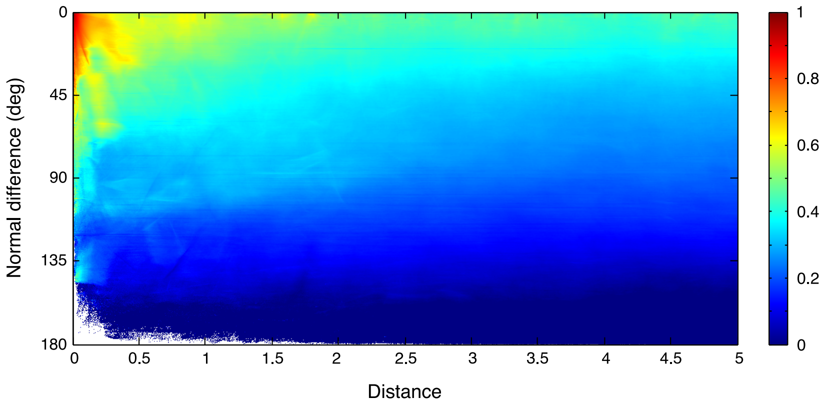


Figure 22: *In order to design good heuristics for our visibility caching algorithm, measurements of the correlation of the visibility function were performed on a range of different scenes. One example is shown here, with the spatial variation of the correlation shown on the top right. The bottom chart shows the average correlation as a function of distance and difference in normals, which forms the basis for the final heuristic.*

these measurements, a robust heuristic for computing the relative importance of different cache records was designed. The heuristic takes several different factors into account, e.g., the distance between visibility approximations, the difference in local surface normal, and a geometric term adjusting for the relative geometries between two cache records.

At each surface point during rendering, we locate a small number of nearby cache records, $\widetilde{V}_i$, and evaluate the heuristic to find a weight, ξ_i, for each record. The linear combination forms the final visibility approximation, that is:

$$\widetilde{V}(\mathbf{x}) = \sum_i \xi_i \widetilde{V}_i(\mathbf{x}). \tag{135}$$

An important advantage of our approach is that since $\widetilde{V}$ is used as a control variate, errors in the approximation do *not* introduce artifacts or bias in the final rendering, only increased variance. Hence, the visibility cache records can be temporally reused over multiple frames for rendering animations efficiently. During render-

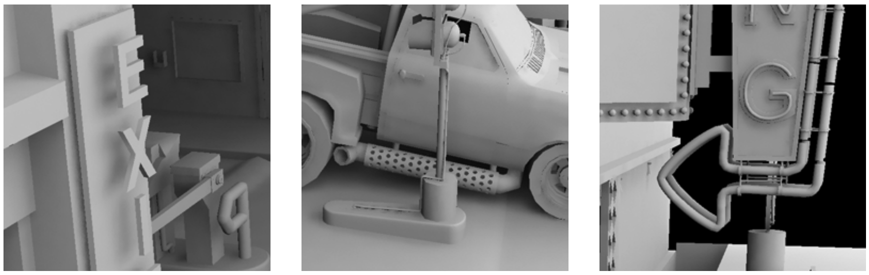


Figure 23: *The visibility approximation computed using our visibility caching algorithm presented in Paper III can also be used for other purposes than Monte Carlo rendering. One such example is ambient occlusion, as shown here.*

ing, we detect when the local visibility approximation is inaccurate compared to the final exact visibility samples used to evaluate the Monte Carlo estimator (Equation 97). When this occurs, a new cache record is inserted, and outdated records are removed. Hence, the local density of cache records is automatically adjusted based on the properties of the visibility function, both spatially and temporally.

The visibility approximation created through visibility caching proved to be accurate enough also for other purposes than using it as a control variate term. In our paper, we show an example where the integral of $\widetilde{V}(\mathbf{x})$ is directly visualized as ambient occlusion (see Figure 23). Another use case is the direct visualization of the control variate term, $J = \int L\widetilde{R}\widetilde{V}d\omega_i$, which represents an accurate approximation of the outgoing radiance. This is useful for pre-visualization of the final rendering.

4.6 Efficient Mapping of the Sphere

As we have seen, integrals in light transport simulation are commonly expressed in terms of integrating some unknown directional quantity, $f(\omega)$, over differential solid angle, $d\omega$, on the upper hemisphere, Ω, or sometimes the entire sphere. To make the implementation easier, it is desirable to rewrite such integrals to be evaluated over the two-dimensional unit square, $(s,t) \in [0,1]^2$. Working in the plane, as opposed to on the (hemi)sphere, enables fast construction of hierarchical data structures (e.g., quadtrees) to approximate the functions and guide the sampling. Therefore, we seek a mapping from the unit square to spherical coordinates, with known simple inverse and other desirable properties (see below).

Several different such mappings have been used throughout our work. In particular, in Paper V we extend an existing square-to-hemisphere mapping with recognized useful properties, to work over the full sphere of directions. An important contribution of our work in that paper is the highly optimized *single instruction, multiple data* (SIMD) implementations of the transforms to the (hemi)sphere and their inverses. I will give a brief overview here.

4.6.1 Mapping Functions

Directions on the unit sphere are most intuitively described by spherical coordinates $\omega = (\theta, \phi)$, where $\theta \in [0, \pi]$ is the polar angle (for the full sphere), and $\phi \in [0, 2\pi]$ is the azimuthal angle (c.f., Figure 4 in Section 2.1.2). The relationship between solid angle and spherical coordinates is given by:

$$d\omega = \sin\theta d\theta d\phi, \tag{136}$$

which is easy to realize geometrically, as a differential patch at a polar angle θ lies on a circle with radius $\sin\theta$. Hence, an integral over the *hemisphere* can be written:

$$\int_\Omega f(\omega)d\omega = \int\limits_{\phi=0}^{2\pi} \int\limits_{\theta=0}^{\pi/2} f(\theta, \phi) \sin\theta d\theta d\phi. \tag{137}$$

To make Monte Carlo integration over the (hemi)sphere practical, we want to express $f(\omega)$ as a two-dimensional function in the plane, $f'(s,t)$, by applying a mapping $M : (s,t) \to (\theta, \phi)$. The mapping transforms positions in the unit square, $\mathbf{u} = (s,t) \in [0,1]^2$, to directions on the (hemi)sphere. Using standard multivariate calculus, the integral in Equation 137 is rewritten as an area integral by the substitution:

$$\int_\Omega f(\omega)d\omega \underset{[\omega=M(\mathbf{u})]}{=} \int\limits_{[0,1]^2} f(M(\mathbf{u})) \left|\det(J_M(\mathbf{u}))\right| \sin\theta dA, \tag{138}$$

where J_M is the Jacobian matrix of the mapping, M, i.e., its partial derivatives with respect to $\mathbf{u}$. Its determinant is:

$$\det(J_M(\mathbf{u})) = \begin{vmatrix} \frac{\partial\theta}{\partial s} & \frac{\partial\theta}{\partial t} \\ \frac{\partial\phi}{\partial s} & \frac{\partial\phi}{\partial t} \end{vmatrix} = \frac{\partial\theta}{\partial s}\frac{\partial\phi}{\partial t} - \frac{\partial\theta}{\partial t}\frac{\partial\phi}{\partial s}. \tag{139}$$

By appropriate choice of M, we can ensure an *equal-area* mapping, which has:

$$d\omega \propto dA, \tag{140}$$

i.e., a differential area in the square represents the same differential solid angle, independent of the direction. This is a useful property, as it allows us to carry out all sampling operations on the unit square, without adjusting for a potential non-uniformity of the mapping. We will now look at two examples of equal-area mappings that we have used, before introducing our optimized version.

Cylindrical Equal-Area Projection For implementing the sampling method described in Paper I, we used the classical cylindrical equal-area projection, which is essentially a cosine-weighted latitude–longitude mapping. Figure 16 shows examples of spherical functions represented in the cylindrical equal-area mapping.

The mapping is derived from the simple projection of the sphere outwards onto a cylinder. For the hemispherical case, the transform is:

$$M: \begin{cases} \theta = \cos^{-1} t \\ \phi = 2\pi s \end{cases} \iff M^{-1}: \begin{cases} s = \frac{\phi}{2\pi} \\ t = \cos\theta. \end{cases} \tag{141}$$

This mapping is widely used in computer graphics because of its simplicity. It is easy to show that it is area-preserving. First, note that its Jacobian determinant is:

$$\det(J_M) = \begin{vmatrix} 0 & -\frac{1}{\sqrt{1-t^2}} \\ 2\pi & 0 \end{vmatrix} = \frac{2\pi}{\sqrt{1-t^2}} = \frac{2\pi}{\sin\theta}, \tag{142}$$

due to $1-t^2 = 1-\cos^2\theta = \sin^2\theta$. Inserted into Equation 138, we find that:

$$\int_\Omega f(\omega)d\omega = 2\pi \int\limits_{[0,1]^2} f(M(\mathbf{u}))\,dA = 2\pi \int\limits_{[0,1]^2} f'(s,t)\,dA, \tag{143}$$

as expected. Note that the 2π scale factor comes from the difference in area between the hemisphere and the unit square.

The main drawback of the cylindrical equal-area projection is that its maximum anisotropy, or *distortion*, tends to infinity towards the pole(s). A differential solid angle maps to a region in the square that gets increasingly stretched out along the s-axis, and compressed along t, as we get closer to $\theta = 0$ or $\theta = \pi$. This is undesirable, as a large anisotropy reduces the benefit of having well-distributed sampling points.

The HEALPix Mapping In the method proposed in Paper IV, we use the so-called *hierarchical equal area isolatitude pixelization* (HEALPix) mapping [45] for encoding precomputed lighting and reflectance data, as well as for representing dynamically generated visibility approximations.

The mapping was originally developed for applications in astronomy, and has low, well-controlled distortion. Different configurations are possible, but in the most common variant, the sphere is divided into 12 square facets. Each facet is a curvilinear quadrilateral (quad) on the surface of the sphere, which can be hierarchically subdivided into pixels of equal area. The mapping is designed so that the pixel centers at any level of subdivision, lie at a discrete number of concentric rings of constant latitude. These properties make it efficient to transform precomputed data into a HEALPix representation. The number of rings is determined by the level of subdivision, but the number of pixels in each ring is also different at different latitudes. For details, we refer to the original paper by Gorski et al. [45], and the official open-source implementation by NASA.[12]

For sampling purposes, the division into 12 distinct square facets presents a drawback compared to mappings that use only a single square, as it is difficult to ensure

[12]The official HEALPix web page is: `http://healpix.jpl.nasa.gov/`

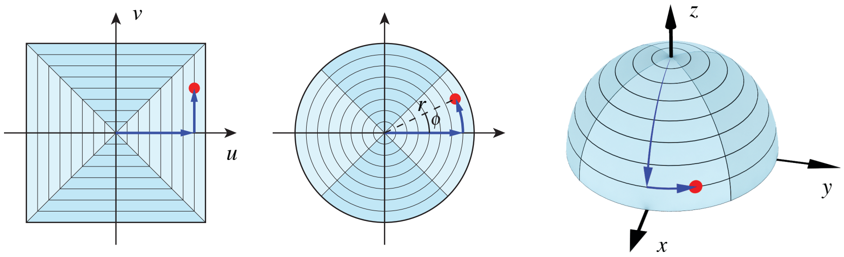


Figure 24: *The concentric map from the square to the hemisphere [128] is shown here. We extend the map to the sphere, and present highly optimized implementations of the forward and inverse transforms for both cases.*

that samples are well-distributed across the boundaries. For example, in our implementation, each facet was individually sampled using well-distributed points, but samples in neighboring facets could end up arbitrarily close to each other. It should also be noted that evaluating the HEALPix mapping function is relatively expensive compared to some simpler alternatives. This is an issue in rendering applications, where a large number of directions need to be mapped back and forth between the (hemi)sphere and the plane.

4.6.2 Efficient Low Distortion Mapping of the (Hemi)Sphere

To address the deficiencies of both the simple cylindrical equal-area projection and the HEALPix mapping, we devote Paper V to describe an efficient mapping of the sphere and hemisphere, which has low distortion and operates over a single unit square. The work builds on the so-called *concentric map* by Shirley and Chiu [128], but extends it in several important ways discussed below.

Hemispherical Case In the original paper [128], points are mapped from the square to the disc or hemisphere by mapping concentric squares to concentric circles. We will briefly review the algorithm, and prove that it is an equal-area mapping. Looking at the rightmost triangular sector in the square shown in Figure 24, the mapping to the corresponding sector of the hemisphere is given by (similar formulas apply in the other sectors):

$$M : \begin{cases} \theta = \cos^{-1}(1-u^2) \\ \phi = \frac{\pi}{4}\frac{v}{u} \end{cases} \quad \text{where} \quad \begin{cases} u = 2s-1, \\ v = 2t-1. \end{cases} \tag{144}$$

The proof of area preservation follows along similar lines as the proof for the cylindrical mapping (Equation 142). We take the Jacobian determinant with respect to (u,v), and note that the initial transform from the unit square to $(u,v) \in [-1,1]^2$ gives an extra scaling by four due to the chain rule. Also, note that $\cos\theta = 1-u^2$

(see the definition of M above), which gives:

$$\det(J_M) = 4\begin{vmatrix} -\frac{-2u}{\sqrt{1-(1-u^2)^2}} & 0 \\ -\frac{\pi}{4}\frac{v}{u^2} & \frac{\pi}{4u} \end{vmatrix} = 4\frac{2u}{\sin\theta}\frac{\pi}{4u} = \frac{2\pi}{\sin\theta}, \tag{145}$$

i.e., we have $d\omega = 2\pi\, dA$ as before.

For practical purposes, we often need to compute the Cartesian coordinates, (x,y,z), of the point on the sphere that corresponds to a direction (θ,ϕ). Converting spherical coordinates to points on the hemisphere, $\Omega = \{(x,y,z) \mid x^2+y^2+z^2 = 1, z \geq 0\}$, is straightforward. For the concentric map, we have [128]:

$$\begin{aligned} x &= \cos\phi\sin\theta &&= \cos\phi\cdot r\sqrt{2-r^2} \\ y &= \sin\phi\sin\theta &&= \sin\phi\cdot r\sqrt{2-r^2} \\ z &= \cos\theta &&= 1-r^2, \end{aligned} \tag{146}$$

where the last step uses polar coordinates on the unit disc, (r,ϕ), c.f., Figure 24. In the first sector, we have $r = u$ and $\phi = \frac{\pi}{4}\frac{v}{u}$, and similar expressions exist for the other sectors. Note that in a typical sampling application, Equation 146 is evaluated for each generated sample in order to compute ray directions for ray tracing. This can be quite costly due to the trigonometric operations, and the branching necessary to select the correct expressions for each sector.

Extension to the Sphere In Paper V, we extend the above mapping from the hemisphere to the full sphere by combining the concentric map with the *octahedral map* [111]. The octahedral map is a clever way to "fold" a unit square over the sphere, by dividing it into eight triangles. Each such triangle maps to a quadrant on one of the two hemispheres. Essentially, a square that is rotated by 45° and inscribed in the unit square, maps to the upper hemisphere, while the four outer triangles are folded down to cover the lower hemisphere. The four corners meet at the south pole. Our new mapping is found by applying the concentric map within each of the eight triangles. For details, refer to our paper.

We present the new mapping in terms of a transform $(u,v) \rightarrow (r,\phi)$, where $(u,v) \in [-1,1]^2$ as before, and the resulting polar coordinates are applied in Equation 146. The expressions for r and ϕ, which are found using simple geometrical observations, are slightly more involved than before as the concentric lines are no longer axis-aligned. For example, in the inner triangle of the first quadrant, we have:

$$r = u+v \tag{147}$$

$$\phi = \frac{\pi}{4}\left(\frac{v-u}{r}+1\right). \tag{148}$$

There are eight such different expressions for (r,ϕ), one for each triangle, which results in three levels of branching in a straightforward implementation.

Optimizations A main contribution of our work is a highly optimized implementation of the mapping, which explores domain-specific knowledge to reduce the complexity. These are optimizations that a compiler cannot easily perform.

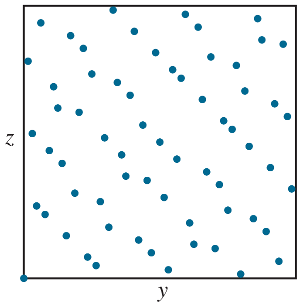
First, we realize that all eight cases can be mapped to the same triangle in the square using absolute values, and the result transformed back to the correct quadrant using sign extension. This entirely avoids branching. Second, we propose specialized numerical approximations of the trigonometric operations. For this purpose, we use *minimax* approximation [110] to find polynomial approximations that minimize the maximum error over the numerical range we are interested in. To further improve the precision we exploit the connection with the Taylor series expansions of cosine and sine, and note that the coefficients for terms of odd (cosine) or even (sine) power are close to zero in the optimized polynomials. By rewriting the optimization problems, we find higher-order minimax approximations with lower errors for a given number of nonzero polynomial terms. The same techniques are applied to find an efficient inverse transform, i.e., a mapping $M^{-1} : (x,y,z) \to (s,t)$, which is often a useful operation.

In the paper, we show robust implementations of both algorithms, i.e., avoiding division-by-zero and with well-controlled maximum errors. For efficiency, we exploit SIMD instructions to map multiple points in parallel. The published code uses the Intel® SSE instruction set, which operates on 4-wide floating-point vectors. It would be straightforward to reimplement the code using a wider instruction set, or port it to the graphics processing unit (GPU). The final result is a $8.6\times$ speedup compared to a standard scalar implementation, which brings the execution time down to 18.7 clock cycles per point. We also show results for the inverse transform, and analyze the average and maximum approximation errors.

The proposed mapping and implementation was successfully used for the research presented in Paper II and III. In both cases, the functions involved (L, R, and V) were represented in world space over the full sphere, rather than over the hemisphere in the local frame. By working in world space, the lighting term, L, can be represented as a single uncompressed image hierarchy, i.e., a mipmap hierarchy. Our work on developing an efficient mapping of the sphere, was largely motivated by this choice of representation.

4.7 Generating Samples with Good 2D Projections

In all the presented sampling methods, the qualities of the input samples play an important role in reducing the variance. This is to be expected, as the use of well-distributed sample points is generally a good idea in integration problems, as discussed in Section 3.5. In our research, we have additionally found that in many cases, it is also important that the *projections* of the sample points along certain dimensions are well-distributed. By projection, we refer to the orthographic projection of the samples along one or more of the coordinate axes. In Paper VII, we present a method for generating *ordered* three-dimensional points with good 2D projection in the non-ordered dimensions.



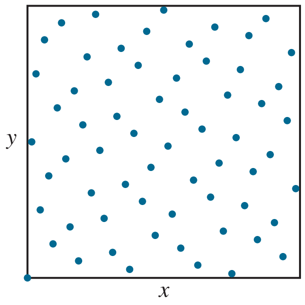


Figure 25: *Left: The two-dimensional projection of the non-ordered dimensions of the original digital* $(0,m,3)$*-net [51], on which we base our method. Right: Our permutation of the generator matrices allows the point set to be iteratively created, while projecting to the more well-distributed Larcher-Pillichshammer points [80] in the non-ordered dimensions (xy).*

4.7.1 Applications

One application for our work is rejection sampling algorithms, such as our hierarchical method presented in Section 4.2.2. In this case, samples are generated in dimension $s+1$ to solve an s-dimensional problem, and the extra dimension is used to determine the random reject or accept of each sample (c.f., Equation 66, where the coordinate in the extra dimension serves the same purpose as U_i). The location of the accepted points in the remaining dimensions determine the final distribution. Hence, it is important that the projection of the samples is well-distributed. The application also requires that the points are ordered in the first dimension, i.e., each new sample added has a larger value along this axis, as the sampling algorithm proceeds incrementally to generate samples where needed.

The application discussed in Paper VII is another important use case, which originally motivated the development of the new sampling method. In this case, the goal is to render *motion blur* effects by computing the exposure over time (Equation 25) using Monte Carlo integration. The focus of the research presented in the paper is on how to evolve the hardware *rasterization* pipeline to support stochastic sampling of moving geometry efficiently. Using rasterization, the visibility from a single point is evaluated for a large set of directions by testing all rays against a single primitive, before proceeding to the next primitive. With *stochastic* rasterization [2, 38, 77], additional sampling dimensions are added to allow the sampling time or the ray origin to be varied. In Paper VII, we focus on adding *one* extra sampling dimension to represent time, t, and let each triangle move within the frame. As such, we only consider the primary visibility from a pinhole camera, i.e., the illumination from all directly visible surfaces is integrated over time, t, and spatially over each pixel, x,y.

To reduce the effect of spatial aliasing, like jagged edges, it is important that the projection of the sample points on x, y is well-distributed. This is particularly important whenever there is only little or no motion blur. At the same time, it is important that the samples are ordered in the time dimension to reduce the number of samples that have to be generated and tested. To make traditional rasterization of static triangles efficient, hierarchical techniques are commonly used to quickly cull rays that are guaranteed not to intersect a particular primitive [90, 91]. To also reduce the number of expensive *ray-vs-moving triangle*, we have developed hierarchical tests for quickly culling rays in motion blur rendering. The core idea is to compute conservative bounds in time for the overlap between a set of rays and the moving triangle. Based on the time bounds, we generate only the sample points that lie within the range. To make this efficient, the samples must be ordered in time. In principle, the approach is similar to the hierarchical rejection sampling method discussed earlier, which also generates only the relevant samples on-the-fly. I will briefly describe the sample generation here, and refer to Paper VII for further details on the rasterization culling tests.

4.7.2 Efficient Digital Construction

To allow a good projection and fast construction, we chose to work with digital (t, m, s)-nets. Their stratification properties ensure that the two-dimensional projection is reasonably well-distributed (see Section 3.5.2). In comparison, stochastic blue noise samples, such as Poisson-disk distributions, do not usually have this property – individual points may very well project to the same coordinates. However, even with (t, m, s)-nets, there are large variations between different methods. The first goal is to find a construction that is well-distributed in two dimensions. The Larcher-Pillichshammer (LP) points [80] fulfill this goal. Grünschloß and Keller [51] then showed how this digital net can be extended to three dimensions to create a $(0, m, 3)$-net in base 2, where the first two dimensions are the original LP points. In their construction, the points are ordered along the *first* dimension, i.e., one of the two dimension that make up the LP point set.

This is not ideal for our applications, as we want the ordered dimension to be different from the two dimensions that hold the well-distributed LP points. Therefore, we have developed a *permutation* of the three original $m \times m$ generator matrices, $C_i,\ i = \{1, 2, 3\}$, over the finite field $\mathbb{F}_2$. In the original method [51], C_1 and C_2 compute the LP points, where C_1 makes ordered points. We transform the three matrices into a new set of matrices, $C'_i = C_i D$, where $D = C_3^{-1} C_1$. The matrix D is chosen so that the new points will be ordered along the third component, but still represent the original LP points in the first two. The core contribution is a derivation of compact expressions for the new set of generator matrices:

$$\begin{aligned} C'_1 &= C_1 C_3^{-1} C_1, \\ C'_2 &= C_2 C_3^{-1} C_1, \\ C'_3 &= C_3 C_3^{-1} C_1 = C_1. \end{aligned} \tag{149}$$

It turns out that this is possible, and we refer to Appendix A of Paper VII for details on the derivation. The new permuted matrices are compactly given by:

$$\begin{aligned}
C_1' &= \left(\binom{m+1-l}{m+1-k} \bmod 2\right)_{k,l=1}^{m}, \\
C_2' &= \left(\binom{m-l}{k-1} \bmod 2\right)_{k,l=1}^{m} \\
C_3' &= \begin{pmatrix} 0 & & 1 \\ & \iddots & \\ 1 & & 0 \end{pmatrix}.
\end{aligned} \tag{150}$$

The result of using our new set of permuted generator matrices is visualized in Figure 25 for $m = 6$, i.e., using $2^6 = 64$ points. It is important to note that the new method generates the same points as before, but they are created in a different order, which better suits our applications. In our paper, we include compact C code for constructing the digital net.

This section concludes the summary of my research contributions. For further details on each algorithm, including discussions on related work, implementation details, and comparison of the results, please refer to the original papers. In the following section, we will look at the research methodology that I have applied during my work.

5 Research Methodology

Most research in computer graphics falls in the area of *applied research*, even though there are exceptions where fundamental new theory has been developed as a result of research focused on solving specific practical problems. This is also largely true for my research – the primary target has been algorithmic development of more efficient methods for solving the light transport problem. The underlying problem is fairly well understood, even though the frontier is continuously moved forward due to the development of faster computer hardware, and more importantly, better algorithms.

As we saw in Section 4, our contribution to the research community is a number of novel tools and techniques for reducing the variance in light transport simulations. In this section, I will briefly discuss my research methodology that lead to these results, and also, the implementation and validation of our techniques.

5.1 Summary of Research Approach

5.1.1 Research Goals

The motivating application for my and my co-workers work is photorealistic rendering. However, it has been our goal to develop methods that are as *widely applicable* as possible. Ultimately, the value of a new technique is a combination of the magnitude of the improvement it gives, how often the method can be used, and what theoretical insights it gives.

The methods for optimized interval arithmetic presented in Paper VI are good examples of work that is broadly useful. It applies to any application that makes use of interval arithmetic, also outside the field of computer graphics. Similarly, our hierarchical sampling techniques are not tied to any particular use case, although there are clear applications in Monte Carlo rendering. The same can be said about our many methods for computing products of functions efficiently, and the mapping of these to the unit square. Other parts of our work are more directly targeted at solving light transport problems, e.g., the development of approximations of the reflectance and visibility functions in Paper II and III. On the other side of the spectrum is our generalized theory for multi-dimensional wavelet products in Paper I, which borders on basic research.

As a second goal, we want to find solutions that are *practical*. Algorithms are often quantitatively compared in terms of their performance on different problems, but their qualitative properties are often just as important in practice. There are several different aspects that make an algorithm practically useful. First, robustness is critical, i.e., that a method handles a wide range of inputs without giving unexpected results. Second, methods that require minimal manual control are desirable. If many different parameters need to be set and tweaked for good results, the practicality of an algorithm is drastically reduced. Last, the implementation complexity is a very important factor. If two methods have comparable performance, the simpler one is almost always preferable. The cost of implementation and debugging should not be underestimated, so an important goal has been to find methods with straightforward implementations.

Examples from all these areas can be found in our work. For instance, for the methods in Paper V and VI, great effort was spent to ensure robust handling of degenerate cases, such as division-by-zero and floating-point Not-a-Number (NaNs) errors. Our product sampling methods are mostly without tweakable parameters, with the exception of the visibility approximation described in Paper III. To aid the implementation of our methods, we have in several cases published pseudocode or actual source code. See, for example, the pseudocode in Paper II, the source code for the (hemi)spherical mapping functions accompanying Paper V, and the sample-generation code in Paper VII.

Last, an important personal goal has been to develop a deep *theoretical understanding* of the studied problems and the mathematical theory behind their solutions. This forms the basis for further research.

5.1.2 Workflow

The workflow has evolved over time throughout the course of my research, and it has differed slightly from project to project. However, the general approach has been fairly systematic, and is summarized in the following points:

1. Problem formulation.
2. Study related previous work and the relevant theory to describe the problem.
3. Develop a mathematical framework and model for the solution.
4. Implementation and validation of the solution.
5. Evaluation against the most relevant state-of-the-art techniques.
6. Publication of method and results.

The formulation of the problem is often found as a direct consequence of the limitations of previous work, or motivated by a specific problem that needs to be solved in a larger system. Then, by studying relevant related work and theory, the advantages and deficiencies of existing solutions are made clear. These two steps have often been iterated to define an appropriate scope of the problem at hand, to determine which methods to compare it against, and to make a rough outline of the solution.

In step (3), the majority of the algorithmic development takes place. This is a creative process that is hard to predict, but I have found it valuable to first gain a very detailed understanding of existing techniques. Finally, the implementation, testing, and evaluation of the work in steps (4)–(5) are the parts that have often taken the largest amount of time. The final result is a finished publication describing the new method, and evaluating its performance against previous alternatives.

In most cases, one or a few core ideas have formed the basis for each project. We have generally spent a fair amount of work formalizing and developing these ideas prior to implementation. The motivation for this is that the implementation and evaluation is made easier if a clear algorithmic description is available, but at the same time, the implementation of new work can give valuable insights that improve the algorithm. We have thus, in most cases, applied the above steps repeatedly to iteratively refine the method, or even adjust the initial problem formulation. I have also aimed at documenting the work along the way, often writing a substantial part of a paper draft early in the process. In doing so, the framework and theoretical model, by necessity, is forced to be well-defined early on.

5.1.3 Collaboration

Roughly half of the publications included in this dissertation are the result of collaborations in groups of four or more researchers. Notably, Paper I and IV, are based on research conducted while visiting other universities (UCSD and Université de Montréal, respectively), and Paper VI and VII are the result of collaborations at Intel Corporation. Even though the first author is usually responsible for doing a majority of the work, collaborating in larger groups allows faster progress as each individual's expertise comes to use. Early feedback and insights from others have been found to be very valuable for improving the method and the results in each project. This was also true for the remaining research projects presented in Paper II, III and V, which were completed more independently in collaboration with my supervisor.

5.2 Implementation and Evaluation

The implementation and evaluation of a new technique is a vital part in all applied research. In our case, the *implementation* refers to the writing of a computer program to perform the operations described by the new methods. This can be done in the form of smaller standalone frameworks, or as part of a larger system.

Successful implementation is important for several different reasons. First, it shows that a method is technically sound without any obvious flaws, which sometimes are difficult to find beforehand. To be sure of the conclusions, the implementation itself must be *validated*, i.e., we must assure ourselves that it performs the intended operations. Second, it gives a good hint of a method's practicality in terms of implementation complexity and how well it can be integrated into existing code bases. Last, the implementation allows the real-life performance of a new technique to be measured. A thorough *evaluation* often forms an integral part of each publication, as the prime goal is to demonstrate better performance or wider applicability than previous solutions. I will now briefly discuss the implementation, validation, and evaluation of our work.

5.2.1 Implementation Strategies

For the implementation of our work, MathWorks MATLAB® has been a valuable tool for early prototyping and validation of the technical soundness. After sketching out and testing the core ideas, final implementations have been written in the programming language C++. In many cases, this has also involved implementing algorithms described in previous work for evaluation purposes. We have aimed at making all implementations well-tested and reasonably optimized to ensure a fair comparison.

The algorithm presented in the first publication (Paper I) was implemented in a simple ray tracing-based renderer, which was written from scratch for my earlier Master's thesis research [20]. This framework was later redesigned and re-implemented to form a more flexible ray tracer, internally called *renderPet*, in which the algorithms described in Paper II, III, and V were implemented. This renderer was similar in design to the open source *pbrt* system [106], and evolved over time (during the years 2005–2008) to support animations, texturing, many different BRDF models, different importance sampling strategies, photon mapping, different types of light sources, multithreading, and so on.

The method of Paper IV was separately implemented in a ray tracer primarily developed by the co-authors Luc Leblanc and Fabrice Rousselle. The final algorithm was also compared against my implementation of wavelet-based importance sampling (Paper I). The work described in the last two papers (Paper VI and VII) was implemented in separate standalone frameworks at Intel Corporation. In the first case, an object-oriented system of classes were written by me to auto-generate optimized Intel® SSE intrinsics code, which was compiled using existing commercial compilers. Finally, the part of Paper VII relevant to this dissertation, i.e., the generation of high-quality sampling points, was implemented in a software-based motion blur rasterization framework running on the GPU, which was written by me partly for other purposes.

5.2.2 Validation

The Monte Carlo variance reduction techniques proposed in Paper I–IV are all unbiased in their basic form (even though some biased extensions are presented). The techniques have been verified by comparing the results against simplistic implementations of unoptimized path tracing, i.e., naïve Monte Carlo sampling of the rendering equation using uncorrelated pseudorandom numbers as input. By averaging over repeated renderings using our methods, we have empirically verified that the generated images have the expected mean. Similarly, the consistency has been verified by letting the rendered images converge using many samples.

The compiler optimizations for interval arithmetic introduced in Paper VI were similarly empirically validated. Since the optimizations are pure code optimizations, which should not change the computed intervals, the implementation was validated by comparing the output against unoptimized interval arithmetic code.

For this purpose, a large set of randomized inputs and several different test cases were used.

The mapping functions described in Paper V were somewhat easier to validate. First, the transform was mathematically proven to be area-preserving, and then the optimized implementations were validated against unoptimized scalar reference code, which has been extensively tested. Similarly, the method for generating samples presented in Paper VII was mathematically proven and the output verified against the expected result. In this case, the generated sampling points have the same coordinates as an existing method, but occur in a different order to give better two-dimensional projections. Validation is thus straightforward.

In addition to the above, in all of my work, I have tried to exercise good programming practices to minimize the risk of errors. We do this, for example, by using object-oriented design and testing each subset of the implementations, e.g., individual subroutines, before continuing with other parts.

5.2.3 Evaluation of the Results

For the evaluation of our research results, we have used a variety of different methods. In each paper, we have aimed for comparing our techniques against the best previously known methods for solving the same problem. In practice, this often becomes a matter of selecting a representative subset of previous work. The choice is somewhat limited by which methods are available as published source code, code that is available from the original authors, or that can be re-implemented with reasonable effort. In our research, we have used all three approaches, i.e., in several cases we have compared against the original implementations of related work, and in other cases, we have implemented previous work for comparison purposes. For details on each project, I refer to the original publications.

In order to compare the results against previous methods, we have used both *qualitative* and *quantitative* measures. First, most of our papers present qualitative results in the form of rendered images to highlight the *visual* impact of different aspects of each algorithm. Visual results are important, as the ultimate goal of computer graphics is to generate images faster or with better quality than before. With images it is also easy to demonstrate the effect of, for example, different approximations or the benefit of using more well-distributed sampling points. In two projects – the mapping functions described in Paper V, and the compiler optimizations in Paper VI – no meaningful visual results were generated and only quantitative results are reported.

For the rendering methods presented in Paper I–IV, the visual quality is directly related to the amount of computation time spent, as the Monte Carlo simulations converge towards a noise-free result. However, differences in the implementations of the compared methods, especially if they are written by different people, may make a fair comparison difficult. Therefore, we have generally presented visual results generated using a fixed number of random samples per integration point, together with the computation times in seconds. In some cases (Paper II and III), we have also presented visual *equal-time* comparisons, where the algorithms are allowed to run for a fixed amount of time and the results compared. In addition to still images, we have produced visual results in the form of moving images in several projects. This is important as the visual results must be temporally stable for rendering of animations, which is a critical consideration in our research field.

In addition to the visual results, we include a quantitative evaluation in each paper. For the Monte Carlo techniques, we measure the *mean squared error* (MSE) of the estimators, which is standard practice. The MSE of an estimator, $\widetilde{\theta}$, of an unknown quantity, θ, is defined as:

$$MSE(\widetilde{\theta}) = E[(\widetilde{\theta}-\theta)^2] = V(\widetilde{\theta}) + \left(\underbrace{E(\widetilde{\theta})-\theta}_{Bias}\right)^2 . \tag{151}$$

If the estimator is unbiased, then the MSE is equal to the variance (c.f., Equation 40). In our work, we have presented both unbiased methods and a few biased extensions for further reduction of the variance.[13] The mean squared error is an appropriate metric in this case, as it captures both the variance and the bias (if it exists). In practice, for evaluating the error, we let $\widetilde{\theta}$ be a rendered image, and θ an unbiased reference rendering, generated with a very large number of samples. The MSE is then given as the squared error averaged over all pixels in the images.

Other quantitative measures are the total *execution time* and the *memory consumption* of each technique, given different parameters. We generally try to report both measures, although the memory consumption is only listed in case it is of practical importance. The execution time is particularly important in Paper V, as its focus lies on very fast mapping of points between the spherical and planar domains. In this project, we additionally report the maximum approximation error, in addition to the mean error, as we are interested in an upper bound on the error. For the interval arithmetic optimizations proposed in Paper VI, we similarly present runtime performance of the optimized code, but also instruction counts. The latter gives a clear picture of how well different compiler optimizations work. Paper VII reports both instruction counts and simulated memory *bandwidth*, computed using a simple architectural simulator. In this case, the actual runtime performance is difficult to estimate, as the research targets hardware innovations.

[13]In cases where data for both unbiased and biased methods have been presented in the same chart, we have vaguely used the term *variance*, even though MSE would have been more descriptive.

6 Conclusion and Future Work

The primary purpose of this introductory chapter has been to give an overview of our research and to tie its different parts together in a coherent framework. As we have seen in Section 4, even though each of the publications focus on a specific method or subproblem, they all serve as building blocks to help make Monte Carlo simulation of light transport more efficient. It has also been my intention to give enough background material in Section 2 and 3, that our work should be accessible by those working in other areas of computer graphics, or perhaps even in other research fields. The Monte Carlo method is widely used, and I hope that, at least, parts of our research on hierarchical variance reduction techniques will be much more widely useful than what we have shown in our example applications.

Focusing on our applications, we note that unbiased Monte Carlo rendering is an active research area, despite its relatively long history in computer graphics. In fact, the approach is also gaining popularity in the industry, e.g., in high-quality offline rendering for product visualization, or rendering of special effects and animated feature films – areas which have previously been dominated by scanline-based renderers, such as RenderMan [4]. Even for real-time graphics, there has recently been attention on generalizing the hardware rasterization pipeline to support stochastic sampling [2, 77] (see also Paper VII). In this case, the goal is not full light transport simulation, but to solve the imaging integral over time and aperture (assuming a thin lens model) using Monte Carlo integration with a small number of random samples per pixel.[14]

There are many reasons for the increasing popularity of unbiased Monte Carlo methods. First, the simplicity of the approach is attractive, since it allows physically accurate simulation of light transport in a wide range of settings, without the need for too many specialized approximations or tricks to simulate certain effects. These can otherwise lead to an enormous software complexity. Using a more brute force approach based on stochastic sampling can reduce the cost of software development. It may also reduce the amount of time artists need to spend to get the desired results. Second, Monte Carlo techniques make it is easier to control the errors in the rendering. Insufficient sampling undeniably results in random noise, while the errors introduced by the approximations of other rendering techniques may have widely different characteristics.

Additionally, Monte Carlo techniques are by definition relatively straightforward to parallelize to a multi-threaded execution, since they work with many independent random samples. This is an important factor, as the prevailing trend in hardware development is to increase the number of compute cores, rather than the clock frequency or memory bandwidth available to each core. This is exemplified by the widespread use of programmable graphics processors, which typically have a large number of cores, but also by today's multicore CPUs that usually have four

[14]In practice, the samples are often deterministic low-discrepancy points or otherwise somewhat restricted in their randomness due to hardware limitations.

to eight physical cores.

In the research presented in this dissertation, we have focused on variance reduction techniques that are applied locally at each integration point in a light transport simulation. This reduces the problem to a two-dimensional integral over incident directions, which has allowed us to develop advanced techniques for explicitly computing and sampling accurate probability distributions. We have developed the first methods for explicitly computing *product* distributions, and used these to apply importance sampling and control variate techniques to reduce the variance. Upon reflection, the common theme throughout our work has been to apply *hierarchical* techniques for analyzing, evaluating, and sampling the involved functions. The hierarchical nature of our algorithms is key to performance, as it allows us to focus the computations to regions of the domain that are important.

There are many avenues for future work. While we have focused on handling arbitrary complex materials under direct illumination, it would be interesting to apply similar hierarchical sampling techniques to indirect illumination. This would require constructing an approximation of the incident indirect light field to guide the sampling. For this purpose, inspiration may come from recent advances in interactive global illumination (see the survey by Ritschel et al. [117]). An approximate global illumination solution may, for example, serve as a control variate term for unbiased Monte Carlo rendering.

Another interesting direction would be to combine our hierarchical sampling methods with techniques for automatically adapting the sampling rate based on a predicted variance. For the presented results, we have generally used a fixed number of samples per integration point (e.g., pixel), though this is not required. Indeed, there are often large local variations in the number of samples needed to reach a certain variance, and recent methods for adaptive sampling [52, 119] show great promise. In our methods, we already build accurate approximations of the unknown integrands, i.e., by estimating and multiplying multiple terms. It would be interesting to use this machinery to adaptively determine appropriate sampling rates, in order to reach a consistent noise level across the image. On a similar note, there is great potential for applying filtering and reconstruction techniques to further reduce the variance (at the expense of adding bias). Recent work by Lehtinen et al. [82, 83] has clearly shown that even sparse random samples contain enough information for a high-quality reconstruction of the original signal, although it comes at a high cost. It may be possible to exploit hierarchical techniques similar to ours, to make high-quality filtering and reconstruction possible at a more reasonable cost.

On a practical level, it would be important to develop data-parallel implementations of our algorithms that exploit the wide vector units of current processors. The work presented in Paper V is an example of this, but we have not specifically focused on providing vectorized implementations in the remaining papers. This would be important to maximize the efficiency of our algorithms on modern CPUs. It should be noted that the development of good parallel programming tools, such as the `ispc` language [107], have made the task somewhat easier.

In conclusion, there are many interesting and important directions for further study. Hierarchical techniques have always been important in computer science, and our research has shown that much can be gained by developing hierarchical Monte Carlo techniques. The research I have presented here is the result of many years of hard work, but it still only represents a small step towards fully photorealistic rendering, simulating all the advanced interactions of light and matter. We are at an interesting point in time, however, where fast algorithmic development combined with massively parallel hardware, changes the landscape of innovation.

Bibliography

[1] AGU, E. O. *Diffraction Shading Models in Computer Graphics*. PhD thesis, University of Massachusetts at Amherst, 2001.

[2] AKENINE-MÖLLER, T., MUNKBERG, J., AND HASSELGREN, J. Stochastic Rasterization using Time-Continuous Triangles. In *Proceedings of Graphics Hardware* (2007), pp. 7–16.

[3] AKENINE-MÖLLER, T., TOTH, R., MUNKBERG, J., AND HASSELGREN, J. Efficient Depth of Field Rasterization Using a Tile Test Based on Half-Space Culling. *Computer Graphics Forum, 31*, 1 (2012), 3–18.

[4] APODACA, A. A., AND GRITZ, L. *Advanced RenderMan*. Morgan Kaufmann, 1999.

[5] APPEL, A. Some Techniques for Shading Machine Renderings of Solids. In *Proceedings of the Spring Joint Computer Conference* (1968), ACM, pp. 37–45.

[6] ASHIKHMIN, M., AND SHIRLEY, P. An Anisotropic Phong BRDF Model. *journal of graphics tools, 5* (2000), 25–32.

[7] ASHIKMIN, M., PREMOŽE, S., AND SHIRLEY, P. A Microfacet-based BRDF Generator. In *Proceedings of SIGGRAPH 2000* (2000), Annual Conference Series, ACM, pp. 65–74.

[8] BALZER, M., SCHLÖMER, T., AND DEUSSEN, O. Capacity-Constrained Point Distributions: A Variant of Lloyd's Method. *ACM Transactions on Graphics, 28*, 3 (2009), 86:1–86:8.

[9] BARSKY, B. A., HORN, D. R., KLEIN, S. A., PANG, J. A., AND YU, M. Camera Models and Optical Systems used in Computer Graphics: Part I, Object-based Techniques. In *Proceedings of ICCSA* (2003), pp. 246–255.

[10] BARSKY, B. A., HORN, D. R., KLEIN, S. A., PANG, J. A., AND YU, M. Camera Models and Optical Systems used in Computer Graphics: Part II, Image-based Techniques. In *Proceedings of ICCSA* (2003), pp. 256–265.

[11] BLINN, J. F. Models of Light Reflection for Computer Synthesized Pictures. In *Computer Graphics* (1977), vol. 11, ACM, pp. 192–198.

[12] BLINN, J. F. Simulation of Wrinkled Surfaces. In *Computer Graphics* (1978), vol. 12, ACM, pp. 286–292.

[13] BLINN, J. F., AND NEWELL, M. E. Texture and Reflection in Computer Generated Images. *Communications of the ACM, 19*, 10 (1976), 542–547.

[14] BORN, M., AND WOLF, E. *Principles of Optics: Electromagnetic Theory of Propagation, Interference and Diffraction of Light*, 7th ed. Cambridge University Press, 1999.

[15] BOWERS, J., WANG, R., WEI, L.-Y., AND MALETZ, D. Parallel Poisson Disk Sampling with Spectrum Analysis on Surfaces. *ACM Transactions on Graphics, 29*, 6 (2010), 166:1–166:10.

[16] BRESENHAM, J. E. Algorithm for Computer Control of a Digital Plotter. *IBM Systems Journal, 4*, 1 (1965), 25–30.

[17] BURKE, D., GHOSH, A., AND HEIDRICH, W. Bidirectional Importance Sampling for Direct Illumination. In *Eurographics Symposium on Rendering* (2005), pp. 147–156.

[18] CATMULL, E. E. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. PhD thesis, 1974.

[19] CEREZO, E., PÉREZ, F., PUEYO, X., SERON, F. J., AND SILLION, F. X. A Survey on Participating Media Rendering Techniques. *The Visual Computer, 21* (2005), 303–328.

[20] CLARBERG, P. Efficient Sampling of Products of Functions using Wavelets. Master's thesis, Lund University, 2005.

[21] CLARKE, F. J. J., AND PARRY, D. J. Helmholtz Reciprocity: Its Validity and Application to Reflectometry. *Lighting Research & Technology, 17* (1985), 1–11.

[22] CLAUSTRES, L., PAULIN, M., AND BOUCHER, Y. BRDF Measurement Modelling using Wavelets for Efficient Path Tracing. *Computer Graphics Forum, 22*, 4 (2003), 701–716.

[23] COOK, R. L. Stochastic Sampling in Computer Graphics. In *Computer Graphics* (1986), vol. 20, ACM, pp. 51–72.

[24] COOK, R. L., PORTER, T., AND CARPENTER, L. Distributed Ray Tracing. In *Computer Graphics* (1984), vol. 18, ACM, pp. 137–145.

[25] COOK, R. L., AND TORRANCE, K. E. A Reflectance Model for Computer Graphics. *ACM Transactions on Graphics, 1*, 1 (1982), 7–24.

[26] DANA, K. J., VAN GINNEKEN, B., NAYAR, S. K., AND KOENDERINK, J. J. Reflectance and Texture of Real-World Surfaces. *ACM Transactions on Graphics, 18*, 1 (1999), 1–34.

[27] DEBEVEC, P. Rendering Synthetic Objects into Real Scenes: Bridging Traditional and Image-based Graphics with Global Illumination and High Dynamic Range Photography. In *Proceedings of SIGGRAPH 98* (1998), Annual Conference Series, ACM, pp. 189–198.

[28] DEVLIN, K., CHALMERS, A., WILKIE, A., AND PURGATHOFER, W. STAR: Tone Reproduction and Physically Based Spectral Rendering. In *State of the Art Reports, Eurographics* (2002), pp. 101–123.

[29] DICK, J., AND PILLICHSHAMMER, F. *Digital Nets and Sequences: Discrepancy Theory and Quasi-Monte Carlo Integration*. Cambridge University Press, 2010.

[30] DIPPÉ, M. A. Z., AND WOLD, E. H. Antialiasing through Stochastic Sampling. In *Computer Graphics* (1985), vol. 19, ACM, pp. 69–78.

[31] DONNER, C., AND JENSEN, H. W. Light Diffusion in Multi-Layered Translucent Materials. *ACM Transactions on Graphics, 24*, 3 (2005), 1032–1039.

[32] DONNER, C., LAWRENCE, J., RAMAMOORTHI, R., HACHISUKA, T., JENSEN, H. W., AND NAYAR, S. An Empirical BSSRDF Model. *ACM Transactions on Graphics, 28*, 3 (2009), 30:1–30:10.

[33] DORSEY, J., AND RUSHMEIER, H. Advanced Material Appearance Modeling. SIGGRAPH Courses, 2009.

[34] DÜR, A. An Improved Normalization for the Ward Reflectance Model. *journal of graphics tools, 11*, 1 (2006).

[35] EBEIDA, M. S., DAVIDSON, A. A., PATNEY, A., KNUPP, P. M., MITCHELL, S. A., AND OWENS, J. D. Efficient Maximal Poisson-disk Sampling. *ACM Transactions on Graphics, 30*, 4 (2011), 49:1–49:12.

[36] EBEIDA, M. S., MITCHELL, S. A., PATNEY, A., DAVIDSON, A., AND OWENS, J. D. A Simple Algorithm for Maximal Poisson-Disk Sampling in High Dimensions. *Computer Graphics Forum, 31*, 2 (2012).

[37] EDWARDS, D., BOULOS, S., JOHNSON, J., SHIRLEY, P., ASHIKHMIN, M., STARK, M., AND WYMAN, C. The Halfway Vector Disk for BRDF Modeling. *ACM Transactions on Graphics, 25*, 1 (2006), 1–18.

[38] FATAHALIAN, K., LUONG, E., BOULOS, S., AKELEY, K., MARK, W. R., AND HANRAHAN, P. Data-Parallel Rasterization of Micropolygons with Defocus and Motion Blur. In *High Performance Graphics* (2009), pp. 59–68.

[39] FATTAL, R. Blue-Noise Point Sampling using Kernel Density Model. *ACM Transactions on Graphics, 30*, 4 (2011), 1–10.

[40] FAURE, H. Discrépance de Suites Associées à un Système de Numération (en Dimension s). *Acta Arith., 41* (1982), 337–351. (In French).

[41] FELLER, W. *An Introduction to Probability Theory and Its Applications*, 3rd ed. Wiley, 1968.

[42] GAMITO, M. N., AND MADDOCK, S. C. Accurate Multidimensional Poisson-Disk Sampling. *ACM Transactions on Graphics, 29*, 1 (2009), 8:1–8:19.

[43] GLASSNER, A. S. A Model for Fluorescence and Phosphorescence. In *Proceedings Fifth Eurographics Workshop on Rendering* (1994), pp. 57–68.

[44] GORAL, C. M., TORRANCE, K. E., GREENBERG, D. P., AND BATTAILE, B. Modeling the Interaction of Light between Diffuse Surfaces. In *Computer Graphics* (1984), vol. 18, ACM, pp. 213–222.

[45] GORSKI, K. M., HIVON, E., BANDAY, A. J., WANDELT, B. D., HANSEN, F. K., REINECKE, M., AND BARTELMANN, M. HEALPix – A Framework for High Resolution Discretization, and Fast Analysis of Data Distributed on the Sphere. *The Astrophysical Journal, 622* (2005), 759–771.

[46] GOURAUD, H. Continuous Shading of Curved Surfaces. *IEEE Transactions on Computers, C-20*, 6, 623–629.

[47] GREENE, N. Environment Mapping and Other Applications of World Projections. *IEEE Computer Graphics and Applications, 6*, 11 (1986), 21–29.

[48] GREENE, N., AND KASS, M. Error-Bounded Antialiased Rendering of Complex Environments. In *Proceedings of SIGGRAPH 94* (1994), Annual Conference Series, ACM, pp. 59–66.

[49] GRIMMETT, G., AND STIRZAKER, D. *Probability and Random Processes*, 3rd ed. Oxford University Press, 2001.

[50] GRÜNSCHLOSS L., HANIKA, J., SCHWEDE, R., AND KELLER, A. (t,m,s)-Nets and Maximized Minimum Distance. In *Monte Carlo and Quasi-Monte Carlo Methods 2006*. Springer Berlin Heidelberg, 2008, pp. 397–412.

[51] GRÜNSCHLOSS L., AND KELLER, A. (t,m,s)-Nets and Maximized Minimum Distance, Part II. In *Monte Carlo and Quasi-Monte Carlo Methods 2008*. Springer Berlin Heidelberg, 2009, pp. 395–409.

[52] HACHISUKA, T., AND JENSEN, H. W. Robust Adaptive Photon Tracing using Photon Path Visibility. *ACM Transactions on Graphics, 30*, 5 (2011), 114:1–114:11.

[53] HALTON, J. H. Algorithm 247: Radical-Inverse Quasi-Random Point Sequence. *Communications of the ACM, 7*, 12 (1964), 701–702.

[54] HANRAHAN, P., AND KRUEGER, W. Reflection from Layered Surfaces due to Subsurface Scattering. In *Proceedings of SIGGRAPH 93* (1993), Annual Conference Series, ACM, pp. 165–174.

[55] HASSELGREN, J., AND AKENINE-MÖLLER, T. PCU: The Programmable Culling Unit. *ACM Transactions on Graphics, 26*, 3 (2007), 92:1–10.

[56] HASSELGREN, J., MUNKBERG, J., AND AKENINE-MÖLLER, T. Automatic Pre-Tessellation Culling. *ACM Transactions on Graphics, 28*, 2 (2009), 19:1–10.

[57] HASTINGS, W. K. Monte Carlo Sampling Methods Using Markov Chains and Their Applications. *Biometrika, 57*, 1 (1970), 97–109.

[58] HE, X. D., TORRANCE, K. E., SILLION, F. X., AND GREENBERG, D. P. A Comprehensive Physical Model for Light Reflection. In *Computer Graphics* (1991), vol. 25, ACM, pp. 175–186.

[59] HEIDRICH, W., SLUSALLEK, P., AND SEIDEL, H.-P. Sampling Procedural Shaders using Affine Arithmetic. In *Proceedings of SIGGRAPH 98* (1998), Annual Conference Series, ACM, pp. 158–176.

[60] HOWELL, J. R., SIEGEL, R., AND MENGUC, M. P. *Thermal Radiation Heat Transfer*, 5th ed. CRC Press, 2010.

[61] HULLIN, M. B., HANIKA, J., AND HEIDRICH, W. Polynomial Optics: A Construction Kit for Efficient Ray-Tracing of Lens Systems. *Computer Graphics Forum (Proceedings of EGSR 2012), 31*, 4 (2012), 1375–1383.

[62] JACOBS, J. F. *The SAGE Air Defense Systems: A personal history*. MITRE Corporation, 1986.

[63] JAROSZ, W. *Efficient Monte Carlo Methods for Light Transport in Scattering Media*. PhD thesis, University of California, San Diego, 2008.

[64] JENSEN, H. W. *Realistic Image Synthesis Using Photon Mapping*. A K Peters, 2001.

[65] KAJIYA, J. T. The Rendering Equation. In *Computer Graphics* (1986), vol. 20, ACM, pp. 143–150.

[66] KOENDERINK, J. J., DOORN, A. J. V., AND STAVRIDI, M. Bidirectional Reflection Distribution Function Expressed in Terms of Surface Scattering Modes. In *Proceedings of ECCV* (1996), pp. 28–39.

[67] KOLB, C., MITCHELL, D., AND HANRAHAN, P. A Realistic Camera Model for Computer Graphics. In *Proceedings of SIGGRAPH 95* (1995), Annual Conference Series, ACM, pp. 317–324.

[68] KOLLIG, T., AND KELLER, A. Efficient Multidimensional Sampling. *Computer Graphics Forum, 21*, 3 (2002).

[69] KUIPERS, L., AND NIEDERREITER, H. *Uniform Distribution of Sequences*. Wiley, 1974.

[70] 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.

[71] LAFORTUNE, E. *Mathematical Models and Monte Carlo Algorithms for Physically based Rendering*. PhD thesis, Katholieke Universiteit Leuven, 1996.

[72] LAFORTUNE, E. P., AND WILLEMS, Y. D. The Ambient Term as a Variance Reducing Technique for Monte Carlo Ray Tracing. In *Eurographics Workshop on Rendering* (1994), pp. 168–176.

[73] LAFORTUNE, E. P., AND WILLEMS, Y. D. Using the Modified Phong Reflectance Model for Physically Based Rendering. Tech. Rep. CW 197, K.U. Leuven, November 1994.

[74] LAFORTUNE, E. P., AND WILLEMS, Y. D. A 5D Tree to Reduce the Variance of Monte Carlo Ray Tracing. In *Eurographics Workshop on Rendering* (1995), pp. 11–20.

[75] LAFORTUNE, E. P. F., FOO, S.-C., TORRANCE, K. E., AND GREENBERG, D. P. Non-Linear Approximation of Reflectance Functions. In *Proceedings of SIGGRAPH 97* (1997), Annual Conference Series, ACM, pp. 117–126.

[76] LAGAE, A., AND DUTRÉ, P. A Comparison of Methods for Generating Poisson Disk Distributions. *Computer Graphics Forum* (2008), 114–129.

[77] LAINE, S., AILA, T., KARRAS, T., AND LEHTINEN, J. Clipless Dual-Space Bounds for Faster Stochastic Rasterization. *ACM Transactions on Graphics, 30*, 4 (2011), 106:1–106:6.

[78] LALONDE, P. *Representations and Uses of Light Distribution Functions*. PhD thesis, University of British Columbia, 1997.

[79] LALONDE, P., AND FOURNIER, A. A Wavelet Representation of Reflectance Functions. *IEEE Transactions on Visualization and Computer Graphics, 3*, 4 (1997), 329–336.

[80] LARCHER, G., AND PILLICHSHAMMER, F. Walsh Series Analysis of the L_2-Discrepancy of Symmetrisized Point Sets. *Monatshefte für Mathematik*, 132 (2001), 1–18.

[81] LAWRENCE, J., RUSINKIEWICZ, S., AND RAMAMOORTHI, R. Efficient BRDF Importance Sampling using a Factored Representation. *ACM Transactions on Graphics, 23*, 3 (2004), 496–505.

[82] LEHTINEN, J., AILA, T., CHEN, J., LAINE, S., AND DURAND, F. Temporal Light Field Reconstruction for Rendering Distribution Effects. *ACM Transactions on Graphics, 30*, 4 (2011), 55:1–55:12.

[83] LEHTINEN, J., AILA, T., LAINE, S., AND DURAND, F. Reconstructing the Indirect Light Field for Global Illumination. *ACM Transactions on Graphics, 31*, 4 (2012), 51:1–51:10.

[84] LLOYD, S. Least Squares Quantization in PCM. *IEEE Transactions on Information Theory, 28*, 2 (2006), 129–137.

[85] MALLAT, S. G. A Theory for Multiresolution Signal Decomposition: The Wavelet Representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 11*, 7 (1989), 674–693.

[86] MALLAT, S. G. *A Wavelet Tour of Signal Processing*, 3rd ed. Academic Press, 2008.

[87] MATÉRN, B. Spatial Variation. *Meddelanden från Statens Skogsforskningsinstitut, 49*, 5 (1960), 1–144.

[88] MATUSIK, W., PFISTER, H., BRAND, M., AND MCMILLAN, L. A Data-Driven Reflectance Model. *ACM Transactions on Graphics, 22*, 3 (2003), 759–769.

[89] MATUSIK, W., PFISTER, H., BRAND, M., AND MCMILLAN, L. Efficient Isotropic BRDF Measurement. In *Proceedings of Eurographics Workshop on Rendering* (2003), pp. 241–247.

[90] MCCOOL, M. D., WALES, C., AND MOULE, K. Incremental and Hierarchical Hilbert Order Edge Equation Polygon Rasterization. In *Proceedings of Graphics Hardware* (2001), pp. 65–72.

[91] MCCORMACK, J., AND MCNAMARA, R. Tiled Polygon Traversal using Half-Plane Edge Functions. In *Proceedings of Graphics Hardware* (2000), pp. 15–21.

[92] METROPOLIS, N., ROSENBLUTH, A. W., ROSENBLUTH, M. N., TELLER, A. H., AND TELLER, E. Equations of State Calculations by Fast Computing Machines. *Journal of Chemical Physics, 21*, 6, 1087–1092.

[93] MILLER, G. S., AND HOFFMAN, C. R. Illumination and Reflection Maps: Simulated Objects in Simulated and Real Environments. Course Notes for Advanced Computer Graphics Animation, SIGGRAPH 84, 1984.

[94] MOORE, R. E. *Interval Analysis*. Prentice-Hall, 1966.

[95] MORTON, G. M. A Computer Oriented Geodetic Data Base; and a New Technique in File Sequencing. Tech. rep., IBM Ltd., Ottawa, Canada, 1966.

[96] NEUMANN, L., NEUMANN, A., AND SZIRMAY-KALOS, L. Compact Metallic Reflectance Models. *Computer Graphics Forum, 18* (1999), 161–172.

[97] NG, R., RAMAMOORTHI, R., AND HANRAHAN, P. Triple Product Wavelet Integrals for All-Frequency Relighting. *ACM Transactions on Graphics, 23*, 3 (2004), 477–487.

[98] NICODEMUS, F. E. Directional Reflectance and Emissivity of an Opaque Surface. *Applied Optics, 4* (1965), 767–773.

[99] NICODEMUS, F. E., RICHMOND, J. C., HSIA, J. J., GINSBERG, I. W., AND LIMPERIS, T. National Bureau of Standards, 1977.

[100] NIEDERREITER, H. Point Sets and Sequences with Small Discrepancy. *Monatshefte für Mathematik, 104* (1987), 273–337.

[101] NIEDERREITER, H. *Random Number Generation and Quasi-Monte Carlo Methods*. SIAM, 1992.

[102] OREN, M., AND NAYAR, S. K. Generalization of Lambert's Reflectance Model. In *Proceedings of SIGGRAPH 94* (1994), Annual Conference Series, ACM, pp. 239–246.

[103] OSTROMOUKHOV, V. Sampling with Polyominoes. *ACM Transactions on Graphics, 26*, 3 (2007), 78:1–78:6.

[104] OWEN, A. B. Randomly Permuted (t,m,s)-Nets and (t,s)-Sequences. *Lecture Notes in Statistics, 107* (1995), 299–317.

[105] PEERS, P., VOM BERGE, K., MATUSIK, W., RAMAMOORTHI, R., LAWRENCE, J., RUSINKIEWICZ, S., AND DUTRÉ, P. A Compact Factored Representation of Heterogeneous Subsurface Scattering. *ACM Transactions on Graphics, 25*, 3 (2006), 746–753.

[106] PHARR, M., AND HUMPHREYS, G. *Physically Based Rendering*, 2nd ed. Morgan Kaufmann, 2010.

[107] PHARR, M., AND MARK, W. R. ispc: A SPMD Compiler for High-Performance CPU Programming. In *Proceedings of Innovative Parallel Computing (InPar)* (2012).

[108] PHONG, B. T. Illumination for Computer Generated Pictures. *Communications of the ACM, 18*, 6 (1975), 311–317.

[109] POULIN, P., AND FOURNIER, A. A Model for Anisotropic Reflection. In *Computer Graphics* (1990), vol. 24, ACM, pp. 273–282.

[110] POWELL, M. J. D. *Approximation Theory and Methods*. Cambridge University Press, 1981.

[111] PRAUN, E., AND HOPPE, H. Spherical Parametrization and Remeshing. *ACM Transactions on Graphics, 22*, 3 (2003), 340–349.

[112] PROAKIS, J. G., AND MANOLAKIS, D. K. *Digital Signal Processing*, 4th ed. Prentice Hall, 2006.

[113] RAMAMOORTHI, R., ANDERSON, J., MEYER, M., AND NOWROUZEZAHRAI, D. A Theory of Monte Carlo Visibility Sampling. *ACM Transactions on Graphics, 31*, 5 (2012), 121:1–121:16.

[114] REDMOND, K. C., AND SMITH, T. M. *Project Whirlwind: History of a Pioneer Computer*. Digital Press, 1980.

[115] REINHARD, E., HEIDRICH, W., DEBEVEC, P., PATTANAIK, S., WARD, G., AND MYSZKOWSKI, K. *High Dynamic Range Imaging: Acquisition, Display, and Image-Based Lighting*, 2nd ed. Morgan Kaufmann, 2010.

[116] REINHARD, E., KHAN, E. A., AKYÜZ, A. O., AND JOHNSON, G. M. *Color Imaging: Fundamentals and Applications*. A K Peters, 2008.

[117] 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.

[118] ROBERT, C. P., AND CASELLA, G. *Monte Carlo Statistical Methods*. Springer, 2010.

[119] ROUSSELLE, F., KNAUS, C., AND ZWICKER, M. Adaptive Sampling and Reconstruction using Greedy Error Minimization. *ACM Transactions on Graphics, 30*, 6 (2011), 159:1–159:12.

[120] RUBIN, D. B. A Noniterative Sampling/Importance Resampling Alternative to the Data Augmentation Algorithm for Creating a Few Imputations When Fractions of Missing Information Are Modest: the SIR Algorithm. *Journal of the American Statistical Association, 82* (1987), 543–546.

[121] RUBIN, D. B. Using the SIR Algorithm to Simulate Posterior Distributions. *Bayesian Statistics, 3* (1988), 395–402.

[122] RUBINSTEIN, R. Y. *Simulation and the Monte Carlo Method*. Wiley-Interscience, 1981.

[123] RUMP, M., MÜLLER, G., SARLETTE, R., KOCH, D., AND KLEIN, R. Photo-Realistic Rendering of Metallic Car Paint from Image-Based Measurements. *Computer Graphics Forum (Proceedings of Eurographics 2008), 27*, 2 (2008), 527–536.

[124] SCHLICK, C. An Inexpensive BRDF Model for Physically-based Rendering. *Computer Graphics Forum (Proceedings of Eurographics 1994), 13*, 3 (1994), 233–246.

[125] SCHLÖMER, T., AND DEUSSEN, O. Towards a Standardized Spectral Analysis of Point Sets with Applications in Graphics. Tech. rep., May 2010.

[126] SCHLÖMER, T., HECK, D., AND DEUSSEN, O. Farthest-Point Optimized Point Sets with Maximized Minimum Distance. In *High Performance Graphics* (2011), pp. 135–142.

[127] SCHRÖDER, P., AND SWELDENS, W. Spherical Wavelets: Efficiently Representing Functions on The Sphere. In *Proceedings of SIGGRAPH 95* (1995), Annual Conference Series, ACM, pp. 161–172.

[128] SHIRLEY, P., AND CHIU, K. A Low Distortion Map between Disk and Square. *Journal of Graphics Tools, 2*, 3 (1997), 45–52.

[129] SHIRLEY, P., WANG, C., AND ZIMMERMAN, K. Monte Carlo Techniques for Direct Lighting Calculations. *ACM Transactions on Graphics, 15*, 1 (1996), 1–36.

[130] SHIRLEY, P. S. *Physically Based Lighting Calculations for Computer Graphics*. PhD thesis, University of Illinois at Urbana-Champaign, 1991.

[131] SILLION, F. X., ARVO, J. R., WESTIN, S. H., AND GREENBERG, D. P. A Global Illumination Solution for General Reflectance Distributions. In *Computer Graphics* (1991), vol. 25, ACM, pp. 187–196.

[132] SLOAN, P.-P., KAUTZ, J., AND SNYDER, J. Precomputed Radiance Transfer for Real-Time Rendering in Dynamic, Low-Frequency Lighting Environments. *ACM Transactions on Graphics, 21*, 3 (2002), 527–536.

[133] SOBOL', I. M. The Distribution of Points in a Cube and the Approximate Evaluation of Integrals. *U.S.S.R. Computational Mathematics and Mathematical Physics, 7*, 4 (1967), 86–12.

[134] STAM, J. Diffraction Shaders. In *Proceedings of SIGGRAPH 99* (1999), Annual Conference Series, ACM, pp. 101–110.

[135] STAM, J. Simulating Diffraction. In *GPU Gems*. Addison Wesley, 2004, ch. 8.

[136] STOLLNITZ, E. J., DEROSE, A. D., AND SALESIN, D. H. *Wavelets for Computer Graphics*. Morgan Kaufmann, 1996.

[137] SUN, B., SUNKAVALLI, K., RAMAMOORTHI, R., BELHUMEUR, P., AND NAYAR, S. Time-Varying BRDFs. *IEEE Transactions on Visualization and Computer Graphics, 13* (2007), 595–609.

[138] SUN, Y. *A Spectrum-based Framework for Realistic Image Synthesis*. PhD thesis, Simon Fraser University, 2000.

[139] SUTHERLAND, I. E. Sketchpad: A Man-Machine Graphical Communication System. In *Proceedings of the Spring Joint Computer Conference* (1963), pp. 329–346.

[140] SUTHERLAND, I. E., SPROULL, R. F., AND SCHUMACKER, R. A. A Characterization of Ten Hidden-Surface Algorithms. *ACM Computing Surveys, 6*, 1 (1974), 1–55.

[141] SZÉCSI, L., SBERT, M., AND SZIRMAY-KALOS, L. Combined Correlated and Importance Sampling in Direct Light Source Computation and Environment Mapping. *Computer Graphics Forum (Proceedings of Eurographics 2004), 23*, 3 (2004), 585–594.

[142] SZIRMAY-KALOS, L., CSONKA, F., AND ANTAL, G. Global Illumination as a Combination of Continuous Random Walk and Finite-Element Based Iteration. *Computer Graphics Forum (Proceedings of Eurographics 2001), 20*, 3 (2001), 288–298.

[143] TALBOT, J., CLINE, D., AND EGBERT, P. Importance Resampling for Global Illumination. In *Eurographics Symposium on Rendering* (2005), pp. 139–146.

[144] TANNENBAUM, D. C., TANNENBAUM, P., AND WOZNY, M. J. Polarization and Birefringency Considerations in Rendering. In *Proceedings of SIGGRAPH 94* (1994), Annual Conference Series, ACM, pp. 221–222.

[145] TORRANCE, K., AND SPARROW, E. Theory for Off-Specular Reflection from Roughened Surfaces. *Journal of the Optical Society of America, 57* (1967), 1105–1114.

[146] VAN DER CORPUT, J. G. Verteilungsfunktionen I. In *Nederl. Akad. Wetensch.* (1935), vol. 38, pp. 813–821. (In Dutch).

[147] VEACH, E. *Robust Monte Carlo Methods for Light Transport Simulation*. PhD thesis, Stanford University, 1997.

[148] VEACH, E., AND GUIBAS, L. J. Metropolis Light Transport. In *Proceedings of SIGGRAPH 97* (1997), Annual Conference Series, ACM, pp. 65–76.

[149] VELÁZQUEZ-ARMENDÁRIZ, E., ZHAO, S., HAŠAN, M., WALTER, B., AND BALA, K. Automatic Bounding of Programmable Shaders for Efficient Global Illumination. *ACM Transactions on Graphics, 28*, 5 (2009), 142:1–9.

[150] WALTER, B. Notes on the Ward BRDF. Tech. Rep. PCG-05-06, Cornell, April 2005.

[151] WANG, R., ZHOU, K., SNYDER, J., LIU, X., BAO, H., PENG, Q., AND GUO, B. Variational Sphere Set Approximation for Solid Objects. *The Visual Computer, 22*, 9 (2006), 612–621.

[152] WARD, G. J. Measuring and Modeling Anisotropic Reflection. In *Computer Graphics* (1992), vol. 26, ACM, pp. 265–272.

[153] WARD, G. J., RUBINSTEIN, F. M., AND CLEAR, R. D. A Ray Tracing Solution for Diffuse Interreflection. In *Computer Graphics* (1988), vol. 22, ACM, pp. 85–92.

[154] WARNOCK, J. E. *A Hidden Surface Algorithm for Computer Generated Halftone Pictures*. PhD thesis, The University of Utah, 1969.

[155] WEI, L.-Y. Parallel Poisson Disk Sampling. *ACM Transactions on Graphics, 27*, 3 (2008), 20:1–20:9.

[156] WEI, L.-Y., AND WANG, R. Differential Domain Analysis for Non-Uniform Sampling. *ACM Transactions on Graphics, 30*, 4 (2011), 50:1–50:10.

[157] WENDLAND, H. *Scattered Data Approximation.* Cambridge University Press, 2004.

[158] WESTIN, S. H., ARVO, J. R., AND TORRANCE, K. E. Predicting Reflectance Functions from Complex Surfaces. In *Computer Graphics* (1992), vol. 26, ACM, pp. 255–264.

[159] WHITTED, T. An Improved Illumination Model for Shaded Display. *Communications of the ACM, 23*, 6, 343–349.

[160] WILKIE, A., TOBLER, R. F., AND PURGATHOFER, W. Combined Rendering of Polarization and Fluorescence Effects. Tech. Rep. TR-186-2-01-11, Institute of Computer Graphics and Algorithms, Vienna University of Technology, 2001.

[161] WILLIAMS, L. Pyramidal Parametrics. In *Computer Graphics* (1983), vol. 17, ACM, pp. 1–11.

[162] WOLFF, L. B., AND KURLANDER, D. J. Ray Tracing with Polarization Parameters. *IEEE Computer Graphics and Applications, 10*, 6 (1990), 44–55.

Paper I

Wavelet Importance Sampling: Efficiently Evaluating Products of Complex Functions

Petrik Clarberg[*] Wojciech Jarosz[†] Tomas Akenine-Möller[*]
Henrik Wann Jensen[†]

[*]Lund University [†]UC San Diego

ABSTRACT

We present a new technique for importance sampling products of complex functions using wavelets. First, we generalize previous work on wavelet products to higher dimensional spaces and show how this product can be sampled on-the-fly without the need of evaluating the full product. This makes it possible to sample products of high-dimensional functions even if the product of the two functions in itself is too memory consuming. Then, we present a novel hierarchical sample warping algorithm that generates high-quality point distributions, which match the wavelet representation exactly. One application of the new sampling technique is rendering of objects with measured BRDFs illuminated by complex distant lighting — our results demonstrate how the new sampling technique is more than an order of magnitude more efficient than the best previous techniques.

ACM Transactions on Graphics (Proceedings of SIGGRAPH 2005),
24(3):1166–1175, 2005.

1 Introduction

In many areas of science, the integral of two or more complex functions needs to be evaluated efficiently. In computer graphics, we have the rendering equation [12], which contains a product of the incident lighting and the material properties of a given object. To evaluate this integral it is common to use Monte Carlo sampling to sample unknown elements of the integral such as visibility. Monte Carlo sampling relies on random sampling of the integral, and is a widely used method for evaluating complex functions. Unfortunately, Monte Carlo methods are computationally costly, and to increase the efficiency, it is necessary to include as much information about the integral as possible in the sampling process. For this purpose importance sampling is a powerful technique. The goal of importance sampling is to distribute samples according to the known elements of the space being sampled in order to reduce the variance due to these elements. In the case of the rendering equation, efficient techniques exist for distributing the samples according to the reflection model being used, or the incident lighting, but not according to both. In the case of known complex lighting and sophisticated reflection models, it would be much more powerful to distribute the samples according to the product of both as shown in Figure 1, but currently there is no efficient method for doing this.

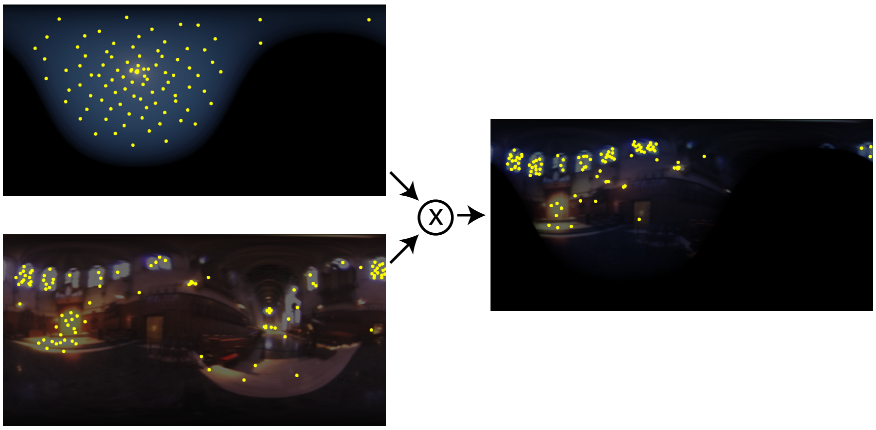


Figure 1: *Left: BRDF importance sampling and environment map importance sampling. Right: Importance sampling of the combination of BRDF and environment. With our new technique we can efficiently sample the product of the BRDF and the environment map without evaluating the full product. Sampling the product results in a superior sampling distribution (shown in the right image) compared with sampling the individual functions (shown on the left). 100 samples were used for each image. The BRDF is a measured acrylic blue material, shown for a single normal and viewing direction, and the environment map is Grace cathedral. Sample points for the product were generated in 0.1 milliseconds.*

In this paper, we introduce a novel method for importance sampling the product of two or more complex functions. Our algorithm uses wavelets to represent the functions under consideration. Given two functions represented in the Haar wavelet basis, it has recently been shown that the wavelet decomposition of the product can be directly computed using tripling coefficients [22]. As a first contribution, we generalize this theory to products of higher dimensional spaces. Second, we introduce a novel hierarchical sample warping scheme that can be folded into the wavelet product evaluation. The new sampling scheme provides high-quality sample distributions as shown in Figure 1, but more importantly it enables the sampling of a complex product without the need for evaluating the full product. This makes the sampling very fast (it can be used on-the-fly during rendering), and it makes it possible to sample according to the product of high-dimensional functions for which the actual product would take up too much space to be practically useful. Our results demonstrate that the new sampling technique provides superior sampling and quality when rendering models with measured material properties illuminated by complex distant high-dynamic range lighting.

2 Previous Work

In this section, we review previous work on importance sampling in computer graphics. Most of the previous work uses importance sampling in the context of the rendering equation [12]:

$$L(\mathbf{x},\omega_o)=L_e(\mathbf{x},\omega_o)+\int_\Omega f_r(\mathbf{x},\omega_i,\omega_o)L_i(\mathbf{x},\omega_i)\cos\theta_i d\omega_i,$$

which is fundamental for rendering realistic images. Here, we need to evaluate the product of the BRDF, f_r, the incident radiance, L_i, and a cosine-term.

BRDF importance sampling is an important and commonly used technique for increasing the efficiency of ray tracing based algorithms. Several important BRDF models can be directly importance sampled, including the Phong model [28], the Ward model [31], and the Lafortune model [15]. See Pharr and Humphreys [25] for more examples. Complex BRDF models such as the Torrance-Sparrow model cannot be analytically inverted and require numerical approximations. Both the Ward model and the Lafortune model have been used to approximate measured BRDF data.

Other work has addressed the problem of importance sampling measured BRDFs. Lalonde [16] used a wavelet representation of the BRDF and presented a novel importance sampling scheme based on random sampling of the wavelet tree. Similar methods were used in [5, 4]. Matusik [20] also used wavelets to represent BRDFs and he presented a numerical method for sampling the BRDF data. Recently, Lawrence et al. [18] introduced a technique for sampling BRDFs based on a factored representation. They represent the BRDF in Rusinkiewicz's parameterization [27], which is compact and compresses well, and their technique can be used for directly importance sampling a 4D BRDF efficiently. BRDF importance sampling is an effective technique, but these methods do not take into account the lighting in the scene, and they become inefficient with complex lighting.

Environment map sampling is another powerful method for rendering objects under complex lighting captured in a high-dynamic range environment map [9]. LightGen [6], Agarwal et al. [1], Kollig and Keller [13], and Ostromoukhov [24] all resampled the environment map by placing pre-integrated directional lights at the brightest locations. This is an efficient method for rendering non-specular materials illuminated by environment map lighting, but as the materials get increasingly specular these methods need a very large number of lights to adequately represent the environment map. Cabral [3] and Ramamoorthi and Hanrahan [26] used spherical harmonics to directly filter the environment map according to the BRDF — these methods do not support efficient sampling, and the spherical harmonics representation is efficient only when the BRDF is smooth and non-specular.

Monte Carlo rendering has a long history in computer graphics starting with the seminal work by Cook et al. [7] and Kajiya [12]. There are numerous Monte Carlo techniques for solving the rendering equation. See Dutré et al. [10] for an overview. Most Monte Carlo techniques use the BRDF sampling methods or the environment map sampling methods just described to solve the rendering equation. Some methods such as path tracing [12, 14] and photon mapping [11] have been extended to importance sampling of the product of the BRDF and the lighting. Both approaches are based on the use of a coarse representation combined with adaptive sampling in order to evaluate the rendering equation. Their efficiency is limited by the coarse representation and they are too costly in the case of complex lighting and specular BRDF models. Veach and Guibas [30] presented a novel technique for combining estimators in Monte Carlo methods using multiple importance sampling. Multiple importance sampling is a powerful method for addressing the situation where either the lighting or the BRDF is complex, as it will pick the best of the available sampling techniques. When both lighting and the BRDF are complicated, multiple importance sampling provides less of an advantage, as it cannot account for the product of the two. It is likely to waste samples in regions with little or no influence on the final result.

Recently, Burke et al. [2] presented a novel technique for rendering objects with complex materials illuminated by an environment map. Their technique uses importance sampling of either the environment map or the BRDF and applies rejection sampling to discard samples if the product of the BRDF and the lighting is not large enough to motivate sampling. This helps reduce the number of samples, but the method is very costly due to the rejection sampling scheme. If both the BRDF and the lighting are complex, Burke et al. reports that more than 90% of the samples are rejected, requiring further evaluation of the functions to locate good samples.

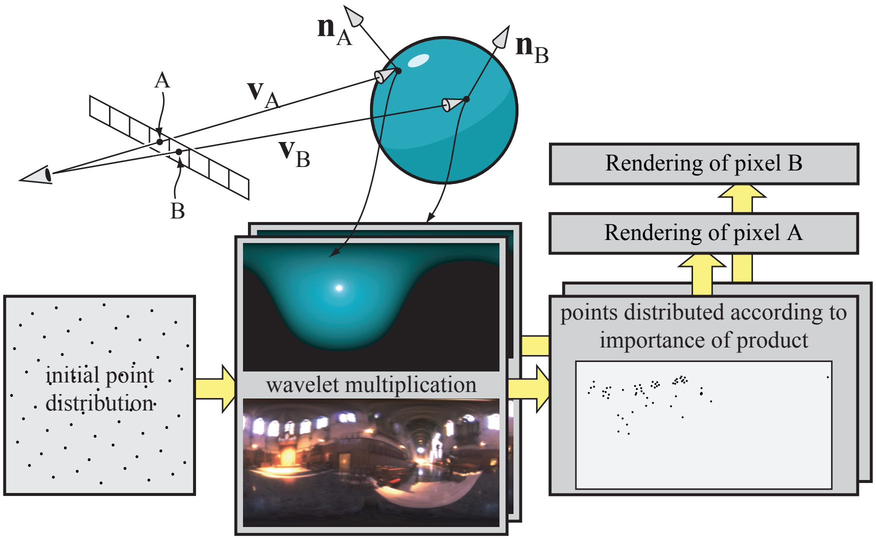


Figure 2: *This figure shows the main steps of our technique. An initial point distribution is warped according to the wavelet product of BRDF and environment map. For each pixel, a new well-distributed sampling is computed, taking both BRDF and environment map into account. The generated sample distribution will be different for two different pixels A and B, since the BRDF changes across the surface.*

In contrast to the previous work our method is capable of efficiently importance sampling the product of the lighting and the BRDF. In the following sections we will describe how this is done by using a compact wavelet representation combined with a generalized wavelet product, and a novel hierarchical sample warping scheme.

3 Overview

Figure 2 gives an overview of the elements of our new sampling technique for the particular example of sampling the product of an environment map and a BRDF. First, we take a high-quality point distribution. Second, we perform a hierarchical evaluation of the wavelet product. As this product is evaluated, the point distribution is warped hierarchically according to the evaluated product. The wavelet product is only completed for regions with one or more samples. The final output is a sample distribution that exactly matches the product of the wavelets without the need of evaluating the full product.

4 Wavelets and Wavelet Products

In this section, we first review the Haar basis notation, since it is used throughout the paper. Other bases can be used as well, but the product tends to get much more complex. The wavelet product in two dimensions is described in Section 4.2. Finally, in Section 4.3, we present a new contribution: a generalized wavelet product, which works in higher dimensions.

4.1 The Haar basis

For the normalized Haar basis, the one-dimensional mother scaling function, $\phi(x)$, and the mother wavelet function, $\psi(x)$, are defined as [19, 29]:

$$\phi(x) := \begin{cases} 1, & \text{for } 0 \le x < 1 \\ 0, & \text{otherwise} \end{cases}, \quad \psi(x) := \begin{cases} 1, & \text{for } 0 \le x < 1/2 \\ -1, & \text{for } 1/2 \le x < 1 \\ 0, & \text{otherwise.} \end{cases}$$

The normalized scaling and wavelet basis functions are:

$$\begin{aligned} \phi_t^l(x) &:= 2^{l/2}\phi(2^l x - t), \\ \psi_t^l(x) &:= 2^{l/2}\psi(2^l x - t), \end{aligned}$$

where l is the level and t the translation of the functions. The Haar basis is orthonormal, meaning that the inner product of two basis functions is zero except when they are the same, in which case the inner product is one.

Expanding a one-dimensional image, H, in the Haar basis is described as:

$$H(x) = H_{0,0}^0 \phi_0^0 + \sum_l \sum_t H_{t,1}^l \psi_t^l = \sum_{\mathbf{i}} H_{\mathbf{i}} \Psi_{\mathbf{i}}, \tag{1}$$

where the second subscript for the basis coefficients, $H_{t,f}^l$, is zero for the scaling function, and one for the wavelet basis functions. Thus, $H_{0,0}^0$ is the scaling coefficient, and $H_{t,1}^l$ are the detail coefficients. The last step in Equation 1 shows the shorthand notation that we will use, where l, t, and the second subscript, indicating the type of basis function, have been "baked" into a single vector parameter $\mathbf{i}$, as done by Ng et al. [22]. Note that this notation generalizes to higher dimensions, such as for two-dimensional images.

As can be seen in Equation 1, only a single scaling coefficient plus scaling function are used. However, as shown in Figure 3, the scaling coefficients at other levels are computed as a part of the decompression process. The scaling coefficient for a certain level l and translation t holds the average of all pixels under the support of the scaling function. This is a key observation, also used by Lalonde [16], that we will use in Section 5. In the example figure, the two scaling coefficients for, e.g., level 1 are 8 and 2.

It should also be noted that a good approximation is obtained if only the n largest coefficients in Equation 1 are kept. This provides lossy compression.

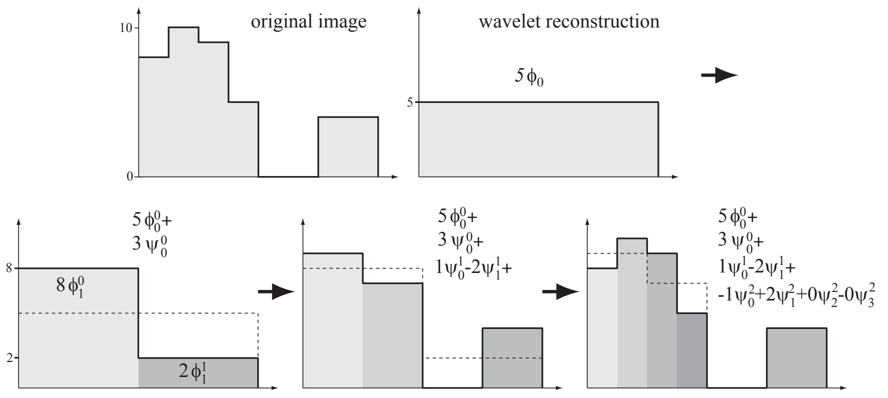


Figure 3: *The 1D image (upper, left) is:* $[8,10,9,5,0,0,4,4]$*, and its unnormalized (used here because it is simpler to display) Haar representation is:* $[5,3,1,-2,-1,2,0,0]$. *The image is then reconstructed one level at a time as follows:* $[5] \rightarrow [5{+}3, 5{-}3] = [8,2] \rightarrow [8{+}1, 8{-}1, 2{-}2, 2{+}2] = [9,7,0,4]$ *and so on.*

4.2 Two-Dimensional Wavelet Product

As a side result in their research on triple products, Ng et al. [22] showed that for a product $G = E \cdot F$ of two-dimensional images, a wavelet representation of $G = \sum G_{\mathbf{i}}\Psi_{\mathbf{i}}$ can be directly computed from the wavelet representations of $E = \sum E_{\mathbf{j}}\Psi_{\mathbf{j}}$ and $F = \sum F_{\mathbf{k}}\Psi_{\mathbf{k}}$, that is, without decompressing them. We will briefly review the theory behind such *wavelet products* here.

The wavelet product we want to compute is described by:

$$G = E \cdot F \Leftrightarrow \sum G_{\mathbf{i}}\Psi_{\mathbf{i}} = \left(\sum E_{\mathbf{j}}\Psi_{\mathbf{j}}\right) \cdot \left(\sum F_{\mathbf{k}}\Psi_{\mathbf{k}}\right) \tag{2}$$

Taking the inner product of $\Psi_{\mathbf{i}}$ with the equation above yields the $\mathbf{i}^{\text{th}}$ basis coefficient for G:

$$G_{\mathbf{i}} = \sum_{\mathbf{j}}\sum_{\mathbf{k}} C_{\mathbf{ijk}} E_{\mathbf{j}} F_{\mathbf{k}}, \text{ where} \tag{3}$$

$$C_{\mathbf{ijk}} = \int\int \Psi_{\mathbf{i}}(\mathbf{x})\Psi_{\mathbf{j}}(\mathbf{x})\Psi_{\mathbf{k}}(\mathbf{x})d\mathbf{x}. \tag{4}$$

The terms $C_{\mathbf{ijk}}$ are called tripling coefficients, and the Ψ are two-dimensional basis functions.

4.3 Generalized Wavelet Product

In this section, we will generalize Equation 4, in order to compute the $\mathbf{i}^{\text{th}}$ basis coefficient of the product, $G = E \cdot F$, where G and E have n dimensions each, and

F has m dimensions, and $0 < m \leq n$. Equation 4 and the related *Haar tripling coefficient theorem* [22] only work for $n = m = 2$. In the following, we assume that the non-standard decomposition technique [29] is used.

For the generalized case, Equation 3 still holds, but the computation of tripling coefficients (Equation 4) is different and depends on m and n as shown below. Assume we have the following vectors $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, $\bar{\mathbf{x}} = (x_1, x_2, \ldots, x_m)$, $m \leq n$, and that we want to compute the $\mathbf{i}^{\text{th}}$ basis coefficient of $G(\mathbf{x}) = E(\mathbf{x}) \cdot F(\bar{\mathbf{x}})$. The derivation can be found in Appendix A, and the main result is summarized in Equation 5.

$$\begin{aligned} C_{\mathbf{ijk}} &= \underbrace{\int \cdots \int}_{n} \Psi_{\mathbf{i}}(\mathbf{x}) \Psi_{\mathbf{j}}(\mathbf{x}) \Psi_{\mathbf{k}}(\bar{\mathbf{x}}) d\mathbf{x} \\ &= \prod_{q=1}^{m} c_{\alpha(\mathbf{ijk},q)} \times \prod_{p=m+1}^{n} \Delta_{\alpha(\mathbf{ij},p)} \end{aligned} \tag{5}$$

Here, $c_{\alpha(\mathbf{ijk},q)}$ is used to denote a *one-dimensional* tripling coefficient, and the $\Delta_{\alpha(\mathbf{ij},p)}$ is, what we call, a one-dimensional *non-standard* coupling coefficient. The functions $\alpha(\mathbf{ijk}, q)$ and $\alpha(\mathbf{ij}, q)$ pick out the relevant parameters from $\mathbf{i}$, $\mathbf{j}$, and $\mathbf{k}$ for the q:th dimension. See again Appendix A for the details on notation, and on how to compute the non-standard coupling coefficients for the Haar basis.

To be able to evaluate Equation 5, we need the following theorem.

One-dimensional Haar Tripling Coefficient Theorem *The integral, $c_{\alpha(\mathbf{ijk},q)}$, of three one-dimensional Haar basis functions is non-zero if and only if the support of the basis functions overlap and either of these two cases hold:*

1. *Two are identical basis functions, and the third basis function, at level l, is either a scaling function sharing their level, or the third basis function is at a strictly coarser level, $l \Rightarrow c_{\alpha(\mathbf{ijk},q)} = \pm 2^{l/2}$.*
2. *There is one scaling function at level l_1, and both the other basis functions are at strictly coarser levels, l_2 and $l_3 \Rightarrow c_{\alpha(\mathbf{ijk},q)} = \pm 2^{(l_2+l_3-l_1)/2}$.*

The proof of the theorem can be found in Appendix B. Applying this theorem twice to the two-dimensional case yields the same results as the two-dimensional Haar tripling coefficient theorem as expected.

Developing techniques for computing tripling coefficients in higher dimensions in the same manner as Ng et al. [22] did for two dimensions appears to be time-consuming as each dimension would probably need a separate derivation and theorem. Instead, we have presented the fundamental result in Equation 5 together with the theorem above that allows for product computations in any dimensions without decompressing the wavelet images. In summary, the generalized tripling coefficients are computed as a simple product of one-dimensional tripling coefficients, and it has the same sublinear properties as the triple product for lossy approximations.

5 Importance Sampling

Assume an n-dimensional wavelet-compressed function:

$$H = \sum_{\mathbf{i}} H_{\mathbf{i}} \Psi_{\mathbf{i}} \tag{6}$$

Sampling the wavelet requires computing the probabilities of different regions of the wavelet tree. We define these recursively such that the probabilities of the child regions at each wavelet node sum to 1. Using the scaling coefficients for a given region, the child probablities are defined as:

$$P^l_{\mathbf{i}} = \frac{H^l_{\mathbf{i},0}}{\sum_{\mathbf{t}} H^l_{\mathbf{t},0}} \tag{7}$$

The simplest method of importance sampling a wavelet-represented function would be to treat the wavelet as a decision tree for hierarchical random thresholding. Previous work on sampling wavelets have used this method [16, 5]. Another technique could be to ignore the hierarchical structure and randomly threshold according to values of the squares at the finest level. However, neither technique produces high quality point distributions, and thresholding according to the finest level requires the computation of the full wavelet product. In the following section, we present an alternative hierarchical warping algorithm that rapidly produces high-quality multi-dimensional sample distributions without having to evaluate a full wavelet product.

5.1 Hierarchical Warping

Our hierarchical warping technique transforms a uniformly distributed set of samples into a warped set of samples according to the wavelet tree. The warping algorithm begins at the coarsest level and proceeds recursively through each level of the wavelet hierarchy. One level of our warping algorithm in two dimensions is illustrated in Figure 4.

When warping multi-dimensional points, we consider each dimension individually. Starting with the first dimension, we split the input point set into two halves. For the following dimension, we split each of these new point sets into two new sets, and so on. The process can be thought of as building a kD-tree of the sample points.

To get the correct distribution when splitting the point set along a certain dimension, we need to compute the total probability for each half of that dimension.

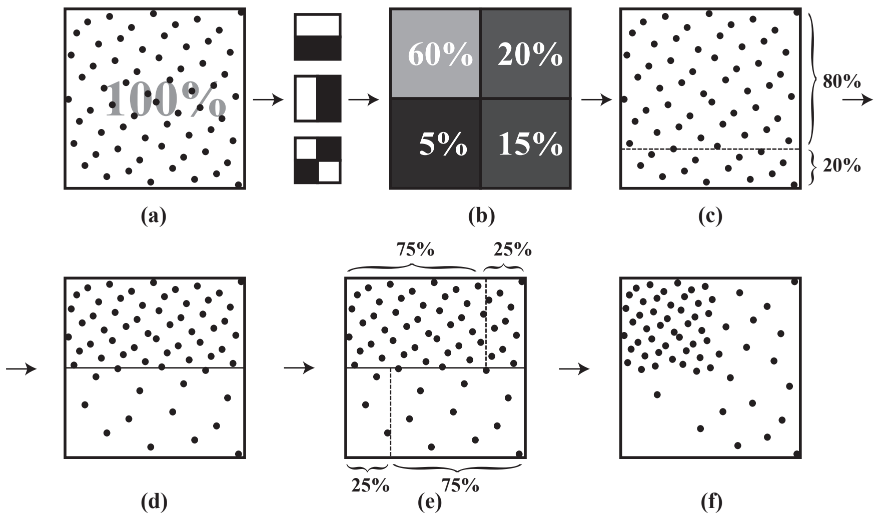


Figure 4: *Warping input points (a) according to one level of a wavelet-compressed importance map where the quadrant percentages (b) are derived from the wavelet coefficients for the current region using Equation 7. The initial point set is first partitioned into two rows with heights determined by their total probabilities (c) and then scaled to fit within the rows (d). Finally, each row is individually divided horizontally according to the probabilities of its child regions (e) and the points are again scaled to fit within the regions to arrive at the warped points for that level (f). The process repeats at step (a) for each child region using its allotted point set as input.*

These are simply the sum of the child region probabilities within each half, call these probabilities $P^l_{\mathbf{i},x-}$ and $P^l_{\mathbf{i},x+}$, where $P^l_{\mathbf{i},x+} = 1 - P^l_{\mathbf{i},x-}$. To perform a split, we first divide the input sample points about a splitting plane positioned so that the lower set contains $P^l_{\mathbf{i},x-}$ fraction of the volume and the upper set contains $P^l_{\mathbf{i},x+}$. Next, both of these point sets are scaled to fit back within the unit interval. The procedure is then repeated independently on these two point sets, but along the subsequent dimensions until all points have been warped along each dimension.

At the completion of one level of n-dimensional warping, the algorithm recurses on all of the child regions which have at least one sample allocated. At each level of the hierarchical process, the expected number of points in each child region is proportional to the probability of that region. Hence, by induction, once recursion terminates the overall distribution of the warped points follows the energy distribution in the whole importance function.

Our warping algorithm is straightforward to use with low discrepancy points as generated by quasi-Monte Carlo methods [23]. Figure 5 demonstrates the effect that changing the input point set has on the quality of the output distribution. Con-

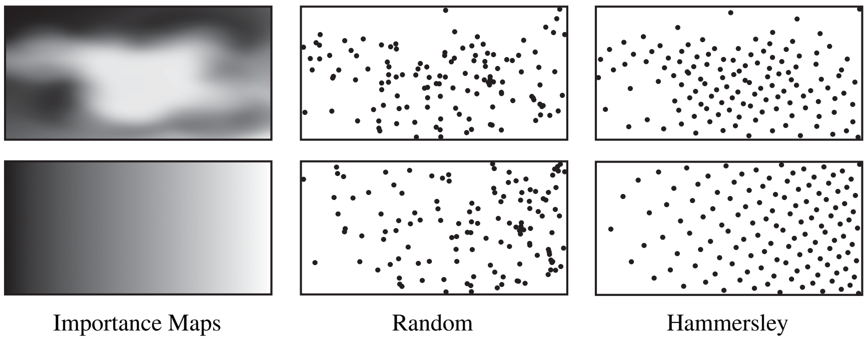


Figure 5: *Two example importance functions and 256 warped samples using uniform random points and Hammersley points. Our warping algorithm is able to preserve the quality of input point set.*

sequently, our warping technique allows for variance reduction during rendering by using quasi-Monte Carlo sample points instead of uniform random points. This is shown in Figure 6.

5.2 Rapidly Sampling Wavelet Products

It is important to note that our sample warping technique only relies on the scaling coefficients at each level of the hierarchy. As shown in Section 4.2 and 4.3, coefficients of a wavelet product can be computed efficiently on-the-fly. In the Haar basis, it is trivial to reconstruct the necessary scaling coefficients from these wavelet coefficients. Hence, it is possible to rapidly warp points according to a wavelet product by computing product coefficients as needed.

In fact, our warping algorithm works particularly well for wavelet products because if no samples are placed in a particular region we do not have to further evaluate the product for that area. This gives significant savings because it eliminates the need to compute coefficients for regions that are not being sampled. This is particularly true in our context where, for example, a highly specular BRDF is typically close to zero for a large portion of the integration domain.

6 Rendering

In this section, we will describe the application of wavelet-based importance sampling of products to the rendering of scenes with general BRDFs under complex direct illumination. The equation for evaluating direct illumination (derived from

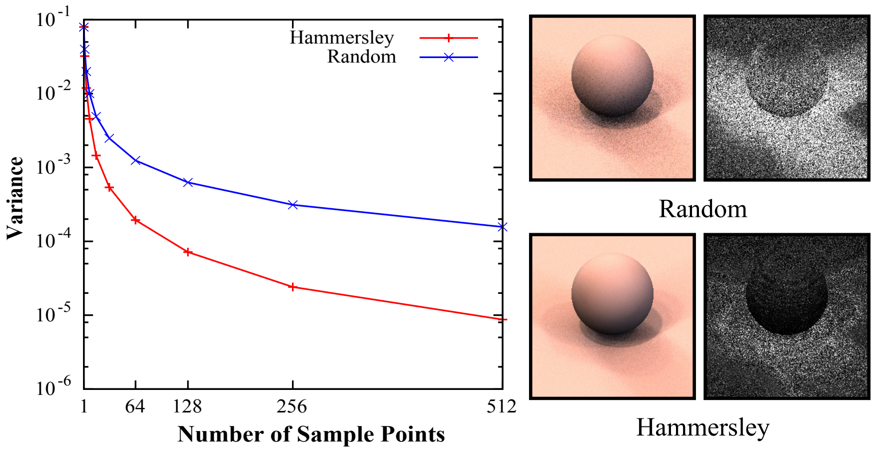


Figure 6: *Variance as a function of the number of samples used for rendering a simple scene illuminated by St. Peters cathedral. On the right are images rendered using 32 samples per pixel and their corresponding variance images. The warping scheme tends to preserve properties of the initial point distribution, hence the variance with Hammersley points is significantly lower than with uniform random points. Using 64 Hammersley points results in less variance than using 512 random points.*

the rendering equation [12]), is:

$$L(\mathbf{x},\omega_o) = \int_\Omega f_r(\mathbf{x},\omega_i,\omega_o)L_i(\mathbf{x},\omega_i)v(\mathbf{x},\omega_i)\cos\theta_i d\omega_i.$$

In our implementation, we are considering illumination $L_i(\mathbf{x},\omega_i)$ provided by a high-dynamic range environment map, and we fold the cosine term into the BRDF and work with the reflectivity, $\rho(\mathbf{x},\omega_i,\omega_o)$. Therefore, a Monte Carlo estimator for the above integral can be written as:

$$\bar{L}(\mathbf{x},\omega_o) = \frac{1}{N}\sum_{i=1}^{N}\frac{\rho(\mathbf{x},\omega_i,\omega_o)L_i(\mathbf{x},\omega_i)v(\mathbf{x},\omega_i)}{p(\omega_i)}. \tag{8}$$

By representing the BRDF and the environment map as wavelets, we can distribute samples according to their product, $H = \sum H_\mathbf{i}\Psi_\mathbf{i}$. The probability associated with a sample ω_i is expressed in terms of wavelet scaling coefficients as:

$$p(\omega_i) = c\frac{H^l_{\mathbf{t},0}}{H^0_{0,0}}, \tag{9}$$

where $H^l_{\mathbf{t},0}$ is the scaling coefficient for the wavelet square in which the sample lies, and $c = 2^{nl/2}$ is a constant derived from the normalized Haar wavelet decomposition, assuming n-dimensional products.

Unbiased vs Biased Rendering We could use equation 8 to directly render an unbiased image by sampling the BRDF, environment map, visibility and divide by the sample probabilities. However, by accepting a small bias introduced by the wavelet approximation:

$$H^l_{\mathbf{t},0} \approx \rho(\mathbf{x}, \omega_i, \omega_o) L_i(\mathbf{x}, \omega_i)/c, \tag{10}$$

the rendering can be made significantly less noisy since two of the terms, which are a major source of variance, cancel out in the division. Combining Equation 10 with Equations 8 and 9, we arrive at:

$$\bar{L}(\mathbf{x}, \omega_o) \approx \frac{H^0_{0,0}}{N} \sum_{i=1}^{N} v(\mathbf{x}, \omega_i). \tag{11}$$

The scaling coefficient $H^0_{0,0}$ is already available, as it was needed during the warping step, and we are using the fact that it represents the pre-integrated value of the BRDF multiplied by the environment map. Thus, rendering is reduced to simple sampling of visibility. Areas with no occlusion will be noise-free since the pre-integrated value is known from the wavelet multiplication. In shadows, noise is inevitable, but since the samples are distributed according to the combination of the BRDF and the environment map, the sampling quickly converges towards a noise-free result.

6.1 2D Wavelet Importance Sampling

We represent a general 4D BRDF reparameterized about the reflected direction as in [26]. This is stored as a 2D tabulation of 2D wavelets. Since the environment map has to be represented in the same coordinate space as the BRDF in order to perform a wavelet product, we replicate the 2D environment re-centered about an array of 2D directions. We found this representation more appropriate than having a 6D BRDF and a 2D environment map as in [22], since it stores less redundant data. Furthermore, most scenes contain many BRDFs but typically only one or a few environment maps.

With our representation, two of the dimensions overlap between the environment map and the BRDF, allowing us to perform wavelet importance sampling on the product in 2D. We represent functions on the sphere in 2D as simple spherical maps sampled over (θ, ϕ), but other parameterizations, such as cube maps, would also work. Note that if a non-uniform parameterization is used, the solid angle each pixel represents must be taken into account. To prevent aliasing, we use super-sampling when creating the wavelet representations.

To get a smooth result, we bilinearly interpolate between the four nearest wavelets in both the environment map and the BRDF. Compression is achieved by discarding wavelet coefficients below a certain threshold. We handle color by storing wavelet coefficients as RGB triplets instead of single values. When sampling according to RGB wavelets, we distribute samples according to the luminance and multiply each sampled value by the normalized color.

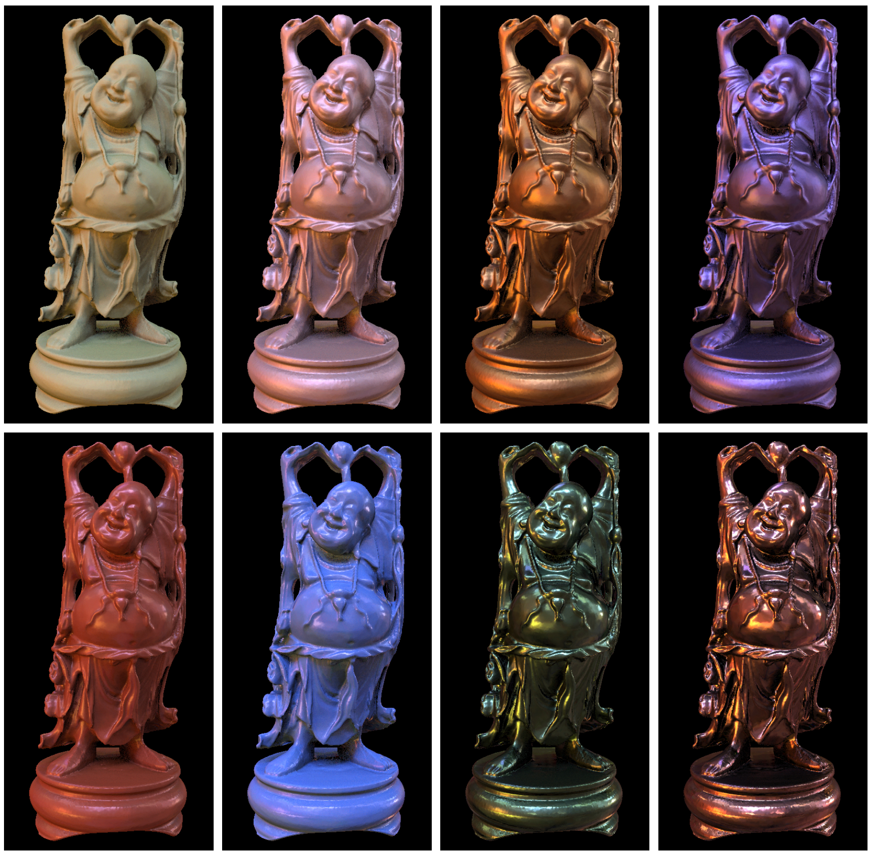


Figure 7: *The Buddha model in Grace cathedral rendered with different measured BRDFs. From left to right: latex fabric, silver paint, gold, blue metallic, brown wax, blue acrylic, green metallic, and copper. Wavelet importance sampling was performed on-the-fly, using 30 Hammersley samples per pixel. The BRDFs were stored with a wavelet sparsity of around 1.5%. The images were rendered in 53–66 seconds at resolution 400×800.*

6.2 4D Wavelet Importance Sampling

In addition to representing BRDFs as tabulated 2D wavelets, we have done some initial experiments with a true 4D×2D wavelet product using the theory in Section 4.3. Here, we store the environment map as a regular 2D map, and work with simple rotationally symmetric BRDFs represented as 4D functions in world-space. The first two dimensions specify the central BRDF direction, and the last two dimensions represent outgoing light direction.

This technique allows us to distribute 4D sample points according to the wavelet product as a preprocess, and eliminates the need to perform on-the-fly wavelet products during rendering. After warping, the wavelets need no longer to be kept in memory, and the warped 4D points are stored in a kD-tree with 2D keys and 2D values per key for efficient range searching during rendering. At render time, we find the n closest sample points in the first two dimensions, and use the last two dimensions of these sample points as world-space light sample directions. We also apply a weighting kernel to the chosen samples based on the deviation from the actual BRDF direction.

7 Results

This section demonstrates our results of the new sampling technique. All results have been generated on a 3.4GHz PC.

The first example, shown in Figure 7, is a rendering of a Buddha model with different measured BRDFs illuminated by a high-dynamic range environment map. From left to right, the selected BRDFs change from mostly diffuse to mostly specular. All images have been rendered with 30 visibility samples based on the product of the BRDF and the environment map. Note that the full range of BRDFs are practically noise-free, even with this low number of samples.

Previous work on rendering objects illuminated by high-dynamic range environment maps have focused on the sampling of either the environment map or the BRDF. In Figure 8 and 9, we compare our method to structured importance sampling [1]. We optimized the amount of jittering in structured importance sampling to reduce banding in the shadow, while avoiding excessive noise in the glossy reflection. We also compare our method to wavelet-based BRDF importance sampling, i.e., not using the wavelet product.

The wavelet product sampling technique produces images with low levels of noise at just 10–30 samples, while structured importance sampling needs 100 or more samples to produce similar results. In particularly difficult cases, even 1000 samples are not enough to accurately capture glossy effects (see Figure 10). BRDF importance sampling performs relatively well for the glossy Buddha, but very poorly for the diffuse floor, giving an overall quality worse than that of structured importance sampling. Performing the wavelet product sampling on-the-fly obviously adds some overhead. With our current implementation it is only possible to use approximately half the number of samples for equal rendering times. As the number of samples grows, this difference becomes smaller.

Figure 11 shows the effect of using a sparse wavelet representation of the BRDF. We have found that for most measured BRDFs, a wavelet resolution of 64×64, and about 2% of the wavelet coefficients are enough to produce renderings indistinguishable from reference images. This combined with reparameterization, allows us to store general BRDFs in around 300kB (resolution $16 \times 16 \times 64 \times 64$) or in higher resolution ($32 \times 32 \times 128 \times 128$) using 5MB. The memory usage could be further optimized, but it is not a major obstacle at this point. The computation

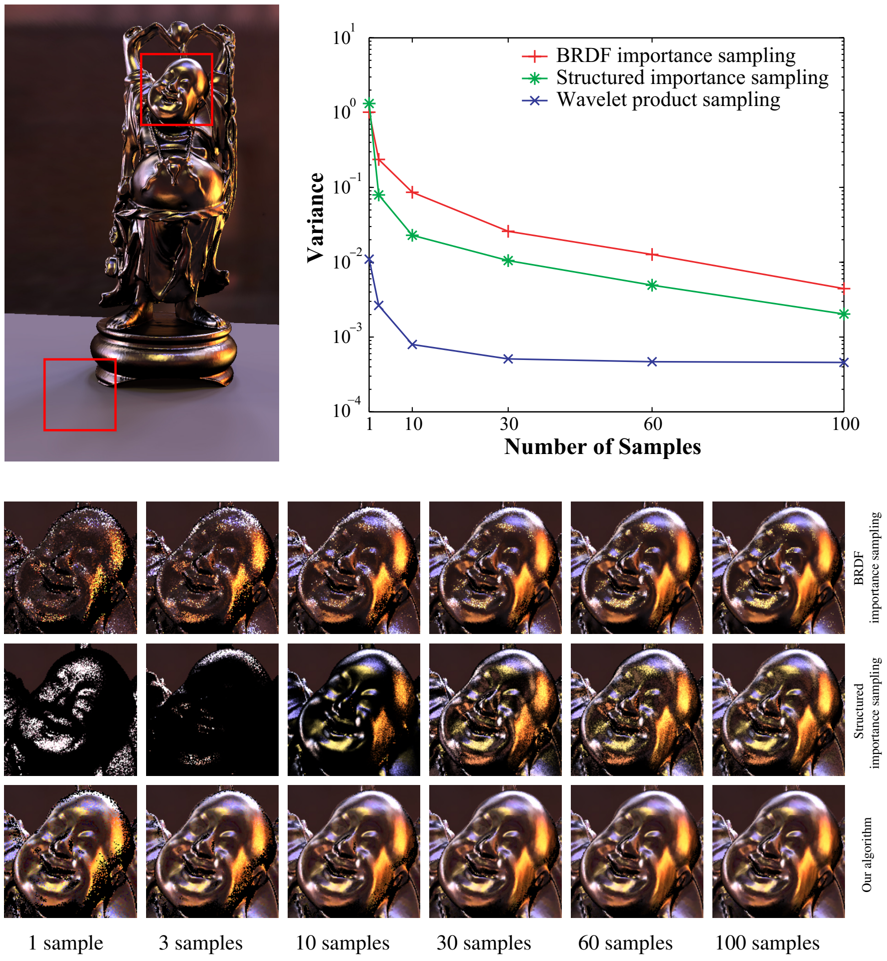


Figure 8: *Wavelet importance sampling of the product compared to structured importance sampling of the environment map (Grace cathedral) and to wavelet-based BRDF importance sampling with varying number of samples. From left to right: 1, 3, 10, 30, 60, 100 samples per pixel. The larger image on the top left represents ground truth and was rendered using brute force ray tracing. The light directions generated by structured importance sampling were jittered to avoid banding in the shadows. Note that structured importance sampling does not work well with very few number of samples, whereas wavelet product sampling quickly gives a good approximation. BRDF importance sampling works relatively well for the glossy Buddha, but very poorly for the diffuse floor (see Figure 9). The rendering times for structured importance sampling were between 4–103 seconds, and for wavelet product sampling between 15–205 seconds, using a wavelet resolution of 64×64. The variance plot in the upper right corner clearly shows that our algorithm outperforms the other two.*

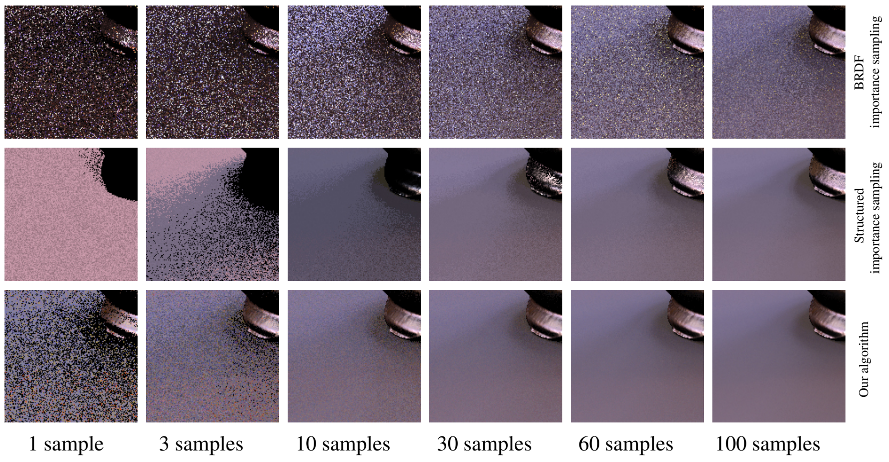


***Figure 9:** The diffuse floor from the scene shown in Figure 8, rendered using BRDF importance sampling, structured importance sampling of the environment, and wavelet importance sampling, respectively. Sampling according to the BRDF alone gives little guidance for diffuse materials, and taking the lighting into account is critical for fast convergence, as the bottom two rows show.*

time for creating the BRDFs is a few minutes, and is not included in the rendering times. For the pre-rotated lighting environment, we use a denser sampling of 64^4 or 128^4, as the lighting is typically more rapidly varying. In our implementation, the wavelet environment maps can take up to a couple of hours to create, mainly due to extensive super-sampling.

Figure 12 compares a dragon scene rendered using sample points warped on-the-fly by a tabulated set of 2D wavelet products and the same scene rendered using sample points warped as a pre-process by a full 4D×2D wavelet product. Both versions used 30 samples per pixel, but since no wavelet products were evaluated during render time with the 4D version, it rendered a factor 2.1 times faster.

Figure 13 shows a complex scene with 12 different measured BRDFs applied to two car models, illuminated by the eucalyptus grove light probe. This shows that with our technique, it is possible to render complex scenes under realistic lighting conditions with very few number of samples. The sampling handles highly glossy materials equally well as more diffuse ones.

8 Conclusions and Future Work

We have presented a new general tool for importance sampling products of complex functions. Our technique is not limited to a specific subset of functions, nor is it limited by the dimensionality of the functions. As an example we used the

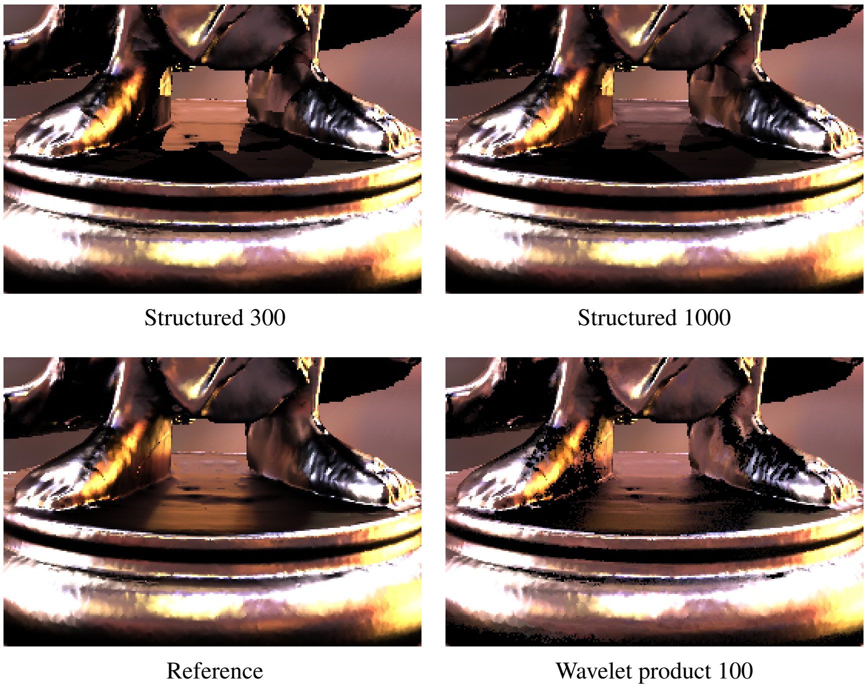


Figure 10: *Part of the Buddha rendered using structured importance sampling (without jittering) with 300 samples, 1000 samples, and wavelet sampling of the product using 100 samples per pixel and a wavelet resolution of 128×128. Even with a large number of samples, structured importance sampling fails to capture some of the more subtle details in the lighting. See for example the area between the model's feet.*

evaluation of the rendering equation, considering general BRDFs under direct illumination by an environment map. Wavelet importance sampling of the BRDF times the environment map proved to give superior sample distributions, enabling us to render essentially noise-free images using as few as 30–100 samples per pixel.

There are many possible applications for wavelet importance sampling of products. For example, our technique could be used for importance sampling of 4D surface light fields [21, 17] multiplied by general BRDFs, or even by 6D bidirectional texture functions [8]. Another new application is importance sampling over the time domain for rendering animations more efficiently. For example, the product of a time-varying environment map and a BRDF could be used to generate samples that are well distributed in both time and space. We are confident our work will be useful in a wide variety of problems involving importance sampling of complex functions.

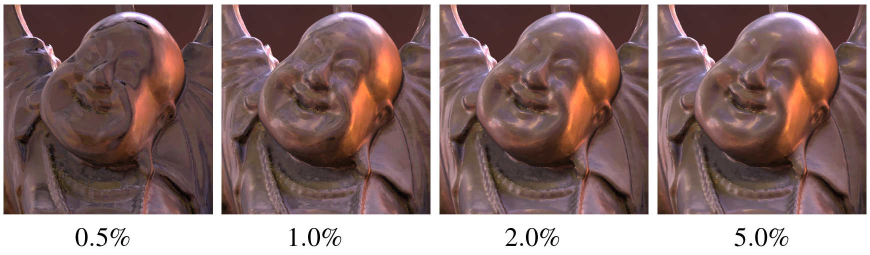


Figure 11: *Buddha with a measured BRDF (oxidized steel), rendered with different levels of BRDF compression. From left to right: 0.5%, 1%, 2%, and 5% sparsity. In practice, a sparsity of 1%–2% produces good results for a wide variety of materials.*

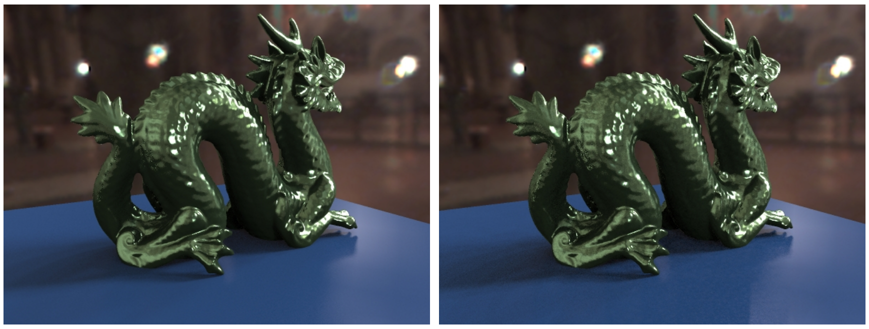


Figure 12: *Glossy dragon in Galileo's tomb rendered using 30 samples per pixel. The left image was rendered using on-the-fly 2D wavelet products and the right image used 4D wavelet products as a pre-process. The right image rendered 2.1 times as fast.*

In the future, we are planning to apply wavelet importance sampling to some of these problems, and further investigate the effect of different parameters on the result. A deeper more theoretical analysis of how the properties of the initial point distribution are affected by the warping would be useful.

Acknowledgements

Tomas was supported by the Hans Werthén Foundation, Carl Tryggers Foundation, and Ernhold Lundströms Foundation. Both Petrik and Tomas were supported by the Swedish Foundation for Strategic Research. This research was also supported by a Sloan Fellowship and the National Science Foundation under Grant No. 0305399.

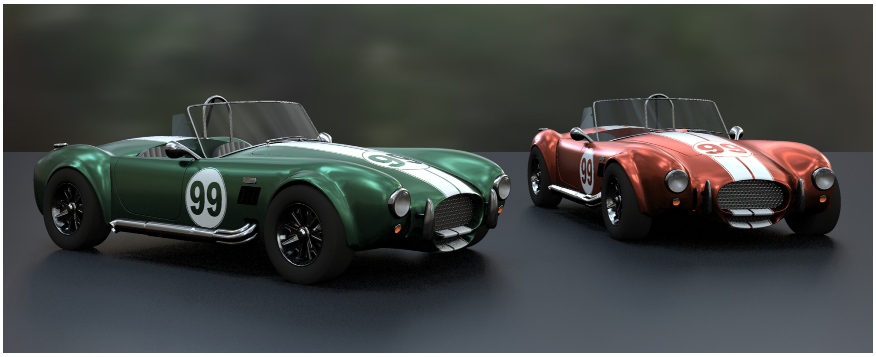


Figure 13: *Scene with 12 different measured BRDFs illuminated by the eucalyptus grove at UC Berkeley. The product of environment map and BRDF was sampled on-the-fly using 10 samples per pixel. The image was rendered in 804 seconds using ray tracing, at a resolution of 4000$\times$1600.*

A Multi-Dimensional Wavelet Product

In this appendix, we will show how to derive Equation 5. Recall that we want to compute the $\mathbf{i}^{\text{th}}$ basis coefficient of the product, $G = E \cdot F$, where G and E have n dimensions each, and F has m dimensions, and $0 < m \leq n$. Furthermore, assume we have the following vectors $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, $\bar{\mathbf{x}} = (x_1, x_2, \ldots, x_m)$, $m \leq n$. For an orthonormal basis, the $\mathbf{i}^{\text{th}}$ basis coefficient of $G(\mathbf{x}) = E(\mathbf{x}) \cdot F(\bar{\mathbf{x}})$ is computed by projecting G onto the $\mathbf{i}^{\text{th}}$ basis function:

$$
\begin{aligned}
G_{\mathbf{i}} &= \underbrace{\int \cdots \int}_{n} \Psi_{\mathbf{i}}(\mathbf{x}) G(\mathbf{x}) d\mathbf{x} = \int \cdots \int \Psi_{\mathbf{i}}(\mathbf{x}) E(\mathbf{x}) F(\bar{\mathbf{x}}) d\mathbf{x} \\
&= \int \cdots \int \Psi_{\mathbf{i}}(\mathbf{x}) \left(\sum_{\mathbf{j}} E_{\mathbf{j}} \Psi_{\mathbf{j}}(\mathbf{x}) \right) \left(\sum_{\mathbf{k}} F_{\mathbf{k}} \Psi_{\mathbf{k}}(\bar{\mathbf{x}}) \right) d\mathbf{x} \\
&= \sum_{\mathbf{j}} \sum_{\mathbf{k}} \left(E_{\mathbf{j}} F_{\mathbf{k}} \int \cdots \int \Psi_{\mathbf{i}}(\mathbf{x}) \Psi_{\mathbf{j}}(\mathbf{x}) \Psi_{\mathbf{k}}(\bar{\mathbf{x}}) d\mathbf{x} \right) \\
&= \sum_{\mathbf{j}} \sum_{\mathbf{k}} C_{\mathbf{ijk}} E_{\mathbf{j}} F_{\mathbf{k}}, \text{ where} \\
C_{\mathbf{ijk}} &= \underbrace{\int \cdots \int}_{n} \Psi_{\mathbf{i}}(\mathbf{x}) \Psi_{\mathbf{j}}(\mathbf{x}) \Psi_{\mathbf{k}}(\bar{\mathbf{x}}) d\mathbf{x}. \qquad (12)
\end{aligned}
$$

Note that $\Psi_{\mathbf{i}}$ and $\Psi_{\mathbf{j}}$ are n-dimensional basis functions, and $\Psi_{\mathbf{k}}$ is an m-dimensional basis function. The above reasoning works for an arbitrary orthonormal basis even though the focus of our work is on the normalized Haar basis. A crucial insight to generalizing Haar wavelet products to higher dimensions is that higher dimension Haar basis functions are separable. For example, in the two-dimensional case this means:

$$
\Psi_{\mathbf{i}} = \Psi^{l}_{\mathbf{t},\mathbf{f}}(x_1, x_2) = \begin{cases} \phi^{l}_{t_1}(x_1) \psi^{l}_{t_2}(x_2), & \text{if } \mathbf{f} = 01, \\ \psi^{l}_{t_1}(x_1) \phi^{l}_{t_2}(x_2), & \text{if } \mathbf{f} = 10, \\ \psi^{l}_{t_1}(x_1) \psi^{l}_{t_2}(x_2), & \text{if } \mathbf{f} = 11. \end{cases} \qquad (13)
$$

Here, the index $\mathbf{i}$ includes all information from l, $\mathbf{t}$, and $\mathbf{f}$. Also, note that $\mathbf{t} = (t_1, t_2)$ is a vector of translations, and $\mathbf{f} = (f_1, f_2)$ is an 2-bit vector that determines which combination of basis functions should be used. For n dimensions, this generalizes to vectors of n elements. A n-dimensional image, H, is then described as $H = \sum_{\mathbf{i}} H_{\mathbf{i}} \Psi_{\mathbf{i}}$ (see Equation 1). To simplify notation, we introduce the following function:

$$
\chi_{\alpha(\mathbf{i},q)}(x_q) := \begin{cases} \phi^{l}_{t_q}(x_q), & \text{if } f_q = 0, \\ \psi^{l}_{t_q}(x_q), & \text{if } f_q = 1, \end{cases} \qquad (14)
$$

where $\alpha(\mathbf{i}, q) = (l, t_q, f_q)$, i.e., it picks out the parameters related to the q:th dimension of $\mathbf{i}$. Thus, our shorthand for Equation 13 becomes $\Psi_{\mathbf{i}}(x_1, x_2) = \chi_{\alpha(\mathbf{i},1)}(x_1)$ $\chi_{\alpha(\mathbf{i},2)}(x_2)$. This reasoning generalizes to higher dimensions as well.

The n-dimensional Haar basis functions can be written as products of 1D scaling or wavelet functions: $\Psi_{\mathbf{i}}(\mathbf{x}) = \prod_{q=1}^{n} \chi_{\alpha(\mathbf{i},q)}(x_q)$, where q is simply a dimension index. This separability property is used below to prove that the tripling coefficient from Equation 12 can be computed as a product of one-dimensional integrals:

$$\begin{aligned}
C_{\mathbf{ijk}} &= \underbrace{\int \cdots \int}_{n} \Psi_{\mathbf{i}}(\mathbf{x})\Psi_{\mathbf{j}}(\mathbf{x})\Psi_{\mathbf{k}}(\overline{\mathbf{x}})d\mathbf{x} \\
&= \prod_{q=1}^{m}\left(\underbrace{\int \chi_{\alpha(\mathbf{i},q)}(x_q)\chi_{\alpha(\mathbf{j},q)}(x_q)\chi_{\alpha(\mathbf{k},q)}(x_q)dx_q}_{\text{1D tripling coefficient}}\right) \times \\
&\quad \prod_{p=m+1}^{n}\left(\underbrace{\int \chi_{\alpha(\mathbf{i},p)}(x_p)\chi_{\alpha(\mathbf{j},p)}(x_p)dx_p}_{\text{1D \textit{non-standard} coupling coefficient}}\right) \\
&= \prod_{q=1}^{m} c_{\alpha(\mathbf{ijk},q)} \times \prod_{p=m+1}^{n} \Delta_{\alpha(\mathbf{ij},p)}. \qquad (15)
\end{aligned}$$

In the last line in the equation above, $c_{\alpha(\mathbf{ijk},q)}$ is used to denote a one-dimensional tripling coefficient, and $\Delta_{\alpha(\mathbf{ij},p)}$ is, what we call, a one-dimensional *non-standard* coupling coefficient. Similar to before, the functions $\alpha(\mathbf{ijk},q)$ and $\alpha(\mathbf{ij},q)$ pick out the relevant parameters from **i**, **j**, and **k** for the q:th dimension. For the Haar basis, $\Delta_{\alpha(\mathbf{ij},p)} = 1$ if the corresponding two basis functions, identified by $\alpha(\mathbf{i},p)$ and $\alpha(\mathbf{j},p)$, are exactly the same[1]. If the two basis functions are overlapping and the finest of the basis functions is a scaling function, then $\Delta_{\alpha(\mathbf{ij},p)} = \pm 2^{(l_1-l_2)/2}$, where l_1 and l_2 are the levels of the two involved basis functions and $l_1 < l_2$. The sign is determined by the signs of the basis functions where they overlap.

B Proof: 1D Haar Tripling Coefficient Theorem

For this proof, we assume that the normalized Haar basis is used, and for that the wavelet basis functions have vanishing integrals. In the following, support of the three basis functions must overlap, otherwise $c_{\alpha(\mathbf{ijk},q)} = 0$.

Case 1: All basis functions at the same level Since the support of all basis functions at the same level are disjoint, the three basis functions must share the same translation, t. Due to orthonormality, the product of two identical basis functions must be a constant function, $2^{l/2} \times 2^{l/2} = 2^l$ with the same support as the terms in the product. The integral of a constant function times the third basis function is zero if the third basis function is a wavelet function due to vanishing integrals of the wavelets. If the third basis function is a scaling function, thus with height

[1]This is the same case as for the standard coupling coefficient, i.e., a Kronecker delta.

$2^{l/2}$, then $c_{\alpha(\mathbf{ijk},q)} = 2^l \times 2^{l/2} \times 2^{-l} = 2^{l/2}$, where l is the level of the three basis functions and 2^{-l} is the width of the basis functions.

Case 2: Exactly two basis functions at the same level We first assume that the basis functions sharing level are at a finer level than the third basis function. The third function will therefore be constant over the support of the two functions sharing level. Thus, the two functions sharing level need to be exactly the same due to orthonormality. Using similar reasoning as for case 1, the tripling coefficient becomes $c_{\alpha(\mathbf{ijk},q)} = \pm 2^{l/2}$, where l is the level of the third basis function (at a coarser level), and the sign is determined by the sign of the third basis function where the finer functions overlap. In all other cases, $c_{\alpha(\mathbf{ijk},q)} = 0$.
Next, we assume that the basis functions sharing level are at a coarser level than the third basis function. The product of the two coarser basis functions will be constant over the support of the finer basis function. Hence, due to vanishing integrals, $c_{\alpha(\mathbf{ijk},q)} = 0$, if the third basis function is a wavelet function. If the third basis function is a scaling function, then $c_{\alpha(\mathbf{ijk},q)} = \pm 2^{l_1/2} \times 2^{l_2/2} \times 2^{l_3/2} \times 2^{-l_1} = \pm 2^{(l_2+l_3-l_1)/2}$, where l_1 is the finest level, and $l_2 = l_3$ is the level of the basis functions sharing level.

Case 3: All three basis functions at different levels Due to orthonormality, the product of the two coarsest basis functions must be a constant function $\pm 2^{l_2/2} \times 2^{l_3/2}$ over the support of the finest basis function. Again, due to vanishing integrals, $c_{\alpha(\mathbf{ijk},q)} = 0$ if the basis function at the finest level is a wavelet function. Using similar reasoning to case 2, $c_{\alpha(\mathbf{ijk},q)} = \pm 2^{l_1/2} \times 2^{l_2/2} \times 2^{l_3/2} \times 2^{-l_1} = \pm 2^{(l_2+l_3-l_1)/2}$ when the basis function at the finest level is a scaling function. This concludes the proof.

Bibliography

[1] AGARWAL, S., RAMAMOORTHI, R., BELONGIE, S., AND JENSEN, H. W. Structured Importance Sampling of Environment Maps. *ACM Transactions on Graphics 22*, 3 (2003), 605–612.

[2] BURKE, D., GHOSH, A., AND HEIDRICH, W. Bidirectional Importance Sampling for Illumination from Environment Maps. In *ACM SIGGRAPH Technical Sketches* (2004).

[3] CABRAL, B., MAX, N., AND SPRINGMEYER, R. Bidirectional Reflection Functions from Surface Bump Maps. In *Computer Graphics (Proceedings of ACM SIGGRAPH 87)* (1987), pp. 273–281.

[4] CLAUSTRES, L., BOUCHER, Y., AND PAULIN, M. Wavelet Projection for Modelling of Acquired Spectral BRDF. *Optical Engineering 43*, 10 (2004), 2327–2339.

[5] CLAUSTRES, L., PAULIN, M., AND BOUCHER, Y. BRDF Measurement Modelling using Wavelets for Efficient Path Tracing. *Computer Graphics Forum 22*, 4 (2003), 701–716.

[6] COHEN, J., AND DEBEVEC, P. LightGen, HDRShop plugin. http://www.ict.usc.edu/ jcohen/lightgen/lightgen.html, 2001.

[7] COOK, R. L., PORTER, T., AND CARPENTER, L. Distributed Ray Tracing. In *Computer Graphics (Proceedings of ACM SIGGRAPH 84)* (1984), pp. 137–145.

[8] DANA, K. J., VAN GINNEKEN, B., NAYAR, S. K., AND KOENDERINK, J. J. Reflectance and Texture of Real-World Surfaces. *ACM Transactions on Graphics 18*, 1 (1999), 1–34.

[9] DEBEVEC, P. Rendering Synthetic Objects into Real Scenes: Bridging Traditional and Image-Based Graphics with Global Illumination and High Dynamic Range Photography. In *Proceedings of ACM SIGGRAPH 98* (1998), pp. 189–198.

[10] DUTRÉ, P., BEKAERT, P., AND BALA, K. *Advanced Global Illumination.* A K Peters, 2003.

[11] JENSEN, H. W. *Realistic Image Synthesis Using Photon Mapping.* A K Peters, 2001.

[12] KAJIYA, J. T. The Rendering Equation. In *Computer Graphics (Proceedings of ACM SIGGRAPH 86)* (1986), pp. 143–150.

[13] KOLLIG, T., AND KELLER, A. Efficient Illumination by High Dynamic Range Images. In *Eurographics Symposium on Rendering* (2003), pp. 45–50.

[14] LAFORTUNE, E. P., AND WILLEMS, Y. D. A 5D Tree to Reduce the Variance of Monte Carlo Ray Tracing. In *Eurographics Workshop on Rendering* (June 1995), pp. 11–20.

[15] LAFORTUNE, E. P. F., FOO, S.-C., TORRANCE, K. E., AND GREENBERG, D. P. Non-Linear Approximation of Reflectance Functions. In *Proceedings of ACM SIGGRAPH 97* (1997), pp. 117–126.

[16] LALONDE, P. *Representations and Uses of Light Distribution Functions.* PhD thesis, University of British Columbia, 1997.

[17] LALONDE, P., AND FOURNIER, A. Interactive Rendering of Wavelet Projected Light Fields. In *Proceedings of Graphics Interface '99* (1999), pp. 107–114.

[18] LAWRENCE, J., RUSINKIEWICZ, S., AND RAMAMOORTHI, R. Efficient BRDF Importance Sampling using a Factored Representation. *ACM Transactions on Graphics 23*, 3 (2004), 496–505.

[19] MALLAT, S. *A Wavelet Tour of Signal Processing.* Academic Press, 1998.

[20] MATUSIK, W., PFISTER, H., BRAND, M., AND MCMILLAN, L. A Data-Driven Reflectance Model. *ACM Transactions on Graphics 22*, 3 (2003), 759–769.

[21] MILLER, G. S. P., RUBIN, S. M., AND PONCELEON, D. B. Lazy Decompression of Surface Light Fields for Precomputed Global Illumination. In *Eurographics Workshop on Rendering* (1998), pp. 281–292.

[22] NG, R., RAMAMOORTHI, R., AND HANRAHAN, P. Triple Product Wavelet Integrals for All-Frequency Relighting. *ACM Transactions on Graphics 23*, 3 (2004), 477–487.

[23] NIEDERREITER, H. *Random Number Generation and Quasi-Monte Carlo Methods.* Society for Industrial and Applied Mathematics, 1992.

[24] OSTROMOUKHOV, V., DONOHUE, C., AND JODOIN, P.-M. Fast Hierarchical Importance Sampling with Blue Noise Properties. *ACM Transactions on Graphics 23*, 3 (2004), 488–495.

[25] PHARR, M., AND HUMPHREYS, G. *Physically Based Rendering: From Theory to Implementation.* Morgan Kaufmann, 2004.

[26] RAMAMOORTHI, R., AND HANRAHAN, P. Frequency Space Environment Map Rendering. *ACM Transactions on Graphics 21*, 3 (2002), 517–526.

[27] RUSINKIEWICZ, S. M. A New Change of Variables for Efficient BRDF Representation. In *Eurographics Workshop on Rendering* (June 1998), pp. 11–22.

[28] SHIRLEY, P. S. *Physically Based Lighting Calculations for Computer Graphics.* PhD thesis, University of Illinois at Urbana-Champaign, 1991.

[29] STOLLNITZ, E. J., DEROSE, T. D., AND SALESIN, D. H. *Wavelets for Computer Graphics: Theory and Applications.* Morgan Kaufmann, 1996.

[30] VEACH, E., AND GUIBAS, L. J. Optimally Combining Sampling Techniques for Monte Carlo Rendering. In *Proceedings of ACM SIGGRAPH 95* (1995), pp. 419–428.

[31] WARD, G. J. Measuring and Modeling Anisotropic Reflection. In *Computer Graphics (Proceedings of ACM SIGGRAPH 92)* (1992), pp. 265–272.

Paper II

Practical Product Importance Sampling for Direct Illumination

Petrik Clarberg Tomas Akenine-Möller

Lund University

Abstract

We present a practical algorithm for sampling the product of environment map lighting and surface reflectance. Our method builds on wavelet-based importance sampling, but has a number of important advantages over previous methods. Most importantly, we avoid using precomputed reflectance functions by sampling the BRDF on-the-fly. Hence, all types of materials can be handled, including anisotropic and spatially varying BRDFs, as well as procedural shaders. This also opens up for using very high resolution, uncompressed, environment maps. Our results show that this gives a significant reduction of variance compared to using lower resolution approximations. In addition, we study the wavelet product, and present a faster algorithm geared for sampling purposes. For our application, the computations are reduced to a simple quadtree-based multiplication. We build the BRDF approximation and evaluate the product in a single tree traversal, which makes the algorithm both faster and more flexible than previous methods.

Computer Graphics Forum (Proceedings of Eurographics 2008), 27(2):681–690, 2008.

1 Introduction

Despite more than thirty years of research, faster and more flexible methods for solving the *rendering equation* [14] are needed to meet the demands of the industry. For high quality images, the interaction of light and materials must be accurately simulated. To obtain realistic results, the incident lighting at a real set can be captured in a high-dynamic range image, and used for lighting the scene [8]. Although many different methods have been proposed, high quality rendering under environment map lighting is still a difficult problem, especially in scenes with realistic materials.

We focus on computing *direct illumination* using Monte Carlo integration, i.e., the integral of the rendering equation is estimated using stochastic point sampling. The integral involves a product over incident lighting, surface reflectance, and visibility. Too few samples or a poor sampling distribution results in undesired noise. With *importance sampling*, noise is reduced by sampling important directions more densely. This is, however, difficult as some parts of the integrand are unknown.

Recently, several methods for sampling according to the product of lighting and BRDF have been proposed. Clarberg et al. [5] presented a general framework for wavelet-based importance sampling of products. Our algorithm is inspired by their work, but we remove most of its limitations. In *two stage importance sampling* [7], heuristics are used to build a BRDF approximation per pixel. In the same spirit, we draw samples samples from the BRDF and build a hierarchical representation on-the-fly, and effectively avoid storing tabulated BRDFs. This is important, as many real world materials exhibit spatially varying reflectance (see Figure 1), and in the industry, complex procedural shaders and shading models with many parameters are common. The high dimensionality makes these materials expensive to precompute and store.

Importance sampling using hierarchical warping only requires the scaling coefficients, or *local averages*, of the product. Therefore, we first simplify the wavelet product to directly compute the product averages from the individual wavelet coefficients. Then, we show that in our application, we can further reduce the complexity. We build the BRDF approximation hierarchically, and at the same time, compute the product by multiplying leaf nodes and propagate the results up. Both these steps are performed in a *single* tree traversal, which makes our algorithm very fast.

By sampling in world space, it is sufficient to use a single two-dimensional environment map, which is stored as a mipmap hierarchy [34]. This enables very high resolution, uncompressed, lighting (e.g., 4k$\times$4k resolution), with practically no precomputation. The key contributions are:

- ▶ We build a hierarchical approximation of the BRDF per pixel, based on a small number of point samples. Any shader that supports BRDF importance sampling can be used, as we do not rely on heuristics [7].

- ► A fast quadtree-based method for computing the product is presented. Compared to wavelet importance sampling [5], our algorithm is faster, avoids precomputation, and supports high resolution lighting.

- ► We also present an optimized wavelet product, which can be useful in many other applications.

- ► Our prototype implementation compares favorably to previous state-of-the-art methods, and presents a viable alternative for production rendering, where precomputation of BRDFs is infeasible.

2 Related Work

At a high level, most algorithms for photo-realistic rendering can be classified as either deterministic, stochastic, or a combination of the two. For a general overview, we refer to popular books on the subject [21, 11]. Veach [28] gives an excellent overview of Monte Carlo methods for light transport problems. In the following, we limit our discussion to methods for computing the direct illumination.

Importance sampling reduces the variance by taking known information about the integrand into account to guide the sampling efforts. High-intensity regions have a larger impact on the result, and hence more samples should be placed here. There are several algorithms that sample according to only one of the involved functions, e.g., environment map sampling [1, 20], and BRDF importance sampling [24, 33, 2, 18, 6, 17]. Work has also been done on linearly combining estimators from multiple importance functions [29].

Recent methods have approached the problem of sampling the *product* of lighting and surface reflectance. One approach is to first draw samples from only one of the terms, and then adjust these samples to (approximately) follow the product distribution. This can be done by *importance resampling* [4, 27], where the initial samples are assigned weights and resampled into a smaller set, or by *rejection sampling* [4], where unimportant samples are discarded. Similar to our algorithm, these methods supports spatially varying BRDFs. However, they may be inefficient when both the lighting and material have complex high-frequency features, since many samples will be useless.

In *wavelet importance sampling* [5], the lighting and BRDF are stored as sparse Haar wavelets, and multiplied on-the-fly using the wavelet product [19]. The product is sampled by hierarchically transforming a uniform point set (e.g., Halton points) into the desired distribution, using a *warping* process. The resulting samples are of high quality, but due to memory constraints, their method is limited to relatively low resolution lighting. In addition, the use of tabulated materials is a severe restriction in many applications. Cline et al. [7] remove some of the limitations by hierarchically splitting the environment map based on peaks in the

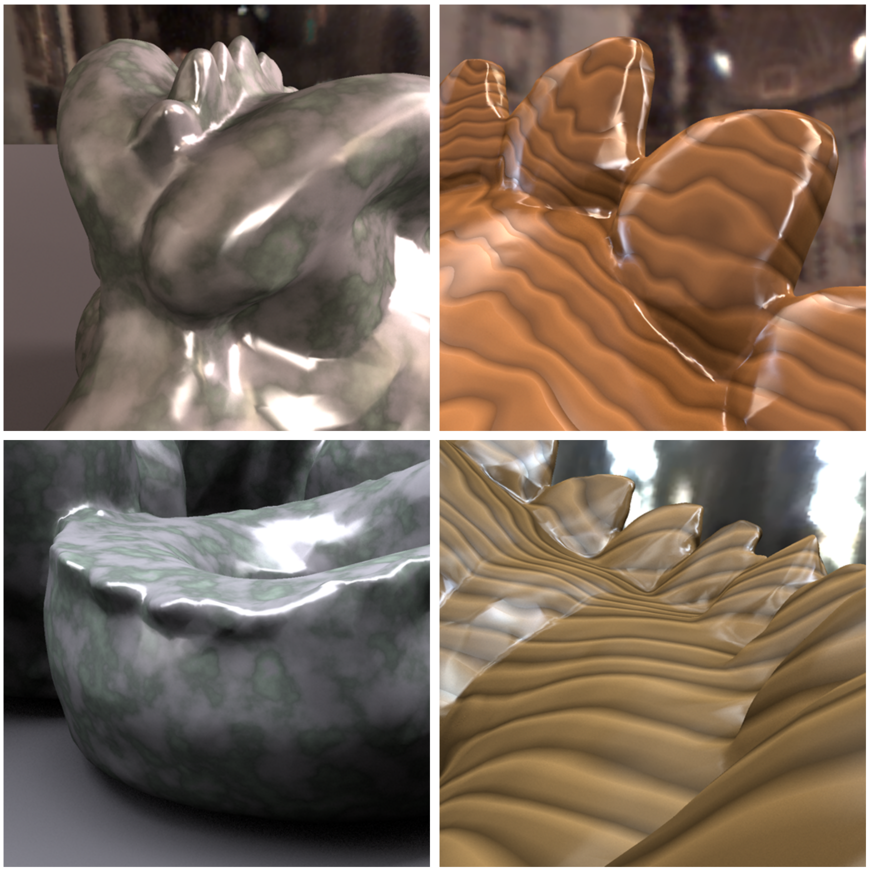


Figure 1: *Examples of procedural shaders with varying diffuse, specular, and shininess coefficients, rendered with our algorithm. Methods using precomputed BRDFs cannot easily handle these types of materials, as the high dimensionality would lead to long precomputation times and excessive memory usage. Our method samples the BRDF and builds the importance function on-the-fly, thus supporting all kinds of spatially varying materials without precomputation.*

BRDF. The product is approximated using summed area tables, and sampled with the previously mentioned warping.

A number of other methods exist, which are not directly based on importance sampling. Ghosh and Heidrich [12] exploit visibility information to lower the variance. They use bidirectional importance sampling to find partially occluded pixels, and then apply Metropolis-Hastings mutations to reduce the noise in these regions. Donikian et al. [9] use adaptive importance sampling. The image is divided into small blocks, and for each block, the sampling density and a pixel estimate are updated until convergence is achieved. The *lightcuts* framework [31, 30] splits the rendering integrals into sets of gather points and light points. A product traversal of these sets together with conservative termination criteria make for efficient rendering. In the same spirit, Hašan et al. [13] formulate the problem as a many-light problem. They sparsely sample the transfer matrix on the GPU, and obtain impressive results.

3 Algorithmic Overview

The outgoing radiance, L_o, in a direction ω_o at a point in the scene, is given by the integral over all incident directions, ω, as follows [14]:

$$L_o(\omega_o) = \int L(\omega) B(\omega_o, \omega) V(\omega) d\omega, \tag{1}$$

where L is the incident illumination by an environment map, B is the reflectance function (see Equation 13), and V is a binary visibility term. The unbiased Monte Carlo estimator, $\hat{L}_o$, is given by:

$$\hat{L}_o = \frac{1}{N} \sum_{i=1}^{N} \frac{L(\omega_i) B(\omega_i) V(\omega_i)}{p(\omega_i)}, \tag{2}$$

where N is the number of samples, and ω_i the sampling directions. For clarity, we have omitted ω_o.

In our algorithm, we build a piecewise constant approximation, $\tilde{B}$, of the reflectance function on-the-fly, and use the product with the *exact* lighting, $L \cdot \tilde{B}$, as our importance function. Thus, the probability density function is equal to:

$$p(\omega) = \frac{L(\omega)\tilde{B}(\omega)}{L_{ns}}, \tag{3}$$

where $L_{ns} = \int L(\omega)\tilde{B}(\omega) d\omega$. The normalization by L_{ns} is necessary since, per definition, $\int p(\omega) d\omega = 1$. The value of L_{ns} is given by the root node in hierarchy of the product $L \cdot \tilde{B}$. When working with wavelets, this is equal to the first scaling coefficient of the wavelet product. Combining Equations 2 and 3, we arrive at:

$$\hat{L}_o = \frac{L_{ns}}{N} \sum_{i=1}^{N} \frac{B(\omega_i) V(\omega_i)}{\tilde{B}(\omega_i)}, \tag{4}$$

As we can see, the only remaining variance comes from the visibility, and the relative inaccuracy of our BRDF approximation, $B(\omega)/\tilde{B}(\omega)$. The main difference to wavelet importance sampling [5], is that we use the exact lighting instead of a wavelet approximation, $\tilde{L}$. Hence, we avoid the variance introduced by $L(\omega)/\tilde{L}(\omega)$, which can be significant with high-resolution environment maps. We also build $\tilde{B}$ on-the-fly, thereby removing the requirement of precomputed materials.

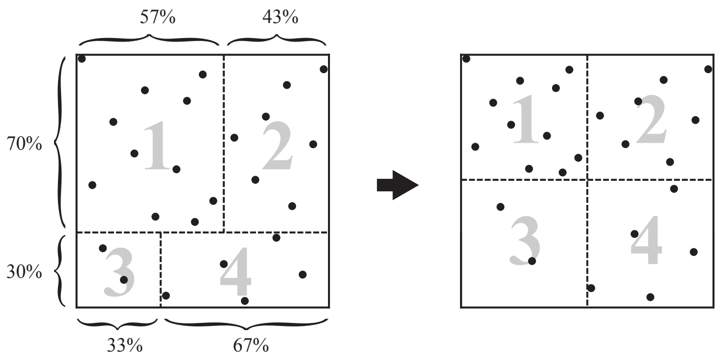


***Figure 2:** The sample warping algorithm by Clarberg et al. [5]. A uniform point set (left) is split according to the relative intensity of each sub-quad, and the points rescaled into the desired distribution (right). When repeated hierarchically, we get a simple algorithm for sampling any importance function with a quadtree structure.*

Equation 4 assumes all functions are scalar-valued. In practice, the lighting and the reflectance are usually in RGB color. We perform all computations on the three color channels, but use the *luminance* of the result as importance function. This is standard practice, but it should be noted that it adds some chrominance noise.

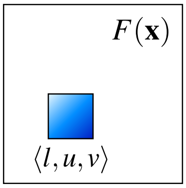
4 Fast Product Evaluation and Sampling

In wavelet importance sampling [5], the individual importance functions are stored as compressed Haar wavelets, and multiplied in the wavelet domain [19]. However, for importance sampling using sample warping (Figure 2), all we need are the *averages* of the quadtree nodes of the product. It is unnecessary to first compute the wavelet coefficients, and then reconstruct the averages.

This is a key observation, which we exploit to make the computations faster. We introduce two novel algorithms; an optimized wavelet product (Section 4.1) and a quadtree-based product (Section 4.2). All results were generated using the latter, so the reader may want to skim through the next section. Please refer to Appendix A and B for an introduction to wavelets and an overview of the terminology.

4.1 Fast Wavelet Product

Let $q = \langle l, u, v \rangle$ be a quad in an image $F(\mathbf{x})$, as illustrated on the right. We want to compute the *average* over q, when F is a product of two images: $F = G \cdot H$. We call this value the *parent sum* of F, or $p_{sumF}(q)$ for short, following the convention of Ng et al. [19]. The terms p_{sum} and "average" are used interchangeably, as they represent the same thing.



The p_{sum} of a node at $\langle l,u,v\rangle$ is the sum of the coefficients of all overlapping basis functions at strictly coarser scales, $k<l$, scaled by $\pm 2^k$. The sign depends on in which quadrant of the basis function that $\langle l,u,v\rangle$ lies. We note that $p_{sumF}(q)$ can also be computed by integrating over $G\cdot H$ restricted to q. We define the *restricted basis*, $\Psi^{\langle q\rangle}$, of a node q as:

$$\Psi^{\langle q\rangle} = \left\{\phi^l_{uv}, \psi_{k_1}, \ldots, \psi_{k_N}\right\}, \tag{5}$$

where k is the set of indices of all wavelet basis functions that are under the support of q, and exist at the same or finer scales. The basis $\Psi^{\langle q\rangle}$ is orthonormal due to the properties of the Haar basis, and represents a subtree of basis functions with an extra scaling function appended to its root, see Figure 3. The expansion of an image $F(\mathbf{x})$ in the basis $\Psi^{\langle q\rangle}$ represents the restriction of F to q:

$$F|_q = \sum_i f_i^{\langle q\rangle}\Psi_i^{\langle q\rangle}(\mathbf{x}) = \begin{cases} F(\mathbf{x}), & \text{if } \mathbf{x}\in q,\\ 0, & \text{if } \mathbf{x}\notin q,\end{cases} \tag{6}$$

$$f^{\langle q\rangle} = \left\{2^{-l}p_{sumF}(q), f_{k_1}, \ldots, f_{k_N}\right\}, \tag{7}$$

where the scaling coefficient is equal to the node average (p_{sum}) scaled by 2^{-l}, and the detail coefficients are the same as the corresponding coefficients of the complete wavelet decomposition (Equation 16).

When F is a product of two functions, i.e., $F = G\cdot H$, we can compute the average over q by integrating over the restrictions of G and H, and obtain the following:

$$\begin{aligned} p_{sumF}(q) &= \frac{1}{A_q}\int G|_q\cdot H|_q\, d\mathbf{x}\\ &= \frac{1}{A_q}\int\left(\sum_i g_i^{\langle q\rangle}\Psi_i^{\langle q\rangle}\right)\left(\sum_j h_j^{\langle q\rangle}\Psi_j^{\langle q\rangle}\right)d\mathbf{x}\\ &= \frac{1}{A_q}\sum_i\sum_j g_i^{\langle q\rangle}h_j^{\langle q\rangle}\int\Psi_i^{\langle q\rangle}\Psi_j^{\langle q\rangle}d\mathbf{x}\\ &= \frac{1}{A_q}\sum_j g_j^{\langle q\rangle}h_j^{\langle q\rangle}, \end{aligned} \tag{8}$$

due to orthonormality. By inserting the area of q, which is $A_q = 2^{-2l}$, and the coefficients (Equation 7), we arrive at:

$$p_{sumF}(q) = p_{sumG}(q)\cdot p_{sumH}(q) + 2^{2l}\underbrace{\sum_{i=1}^{N} g_{k_i}h_{k_i}}_{c_{sumGH}(q)} \tag{9}$$

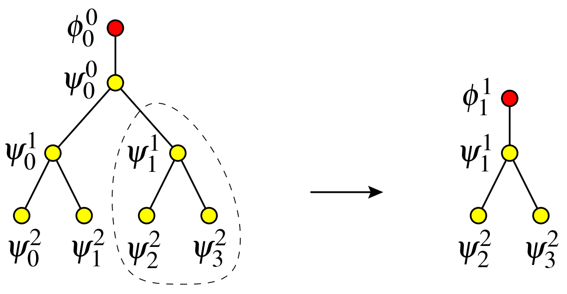


Figure 3: *On the left, a complete 1D wavelet basis tree, and on the right, a restricted basis consisting of a subtree of basis functions plus a scaling function. In two dimensions, each node has three basis functions and four children.*

Here, we have introduced the notation $c_{sumGH}(q)$ for the sum of the product of all coefficients in G and H, for which the wavelet functions are identical and exist under the support of q, at the same or finer scales. We call this the *children sum* of the product $G \cdot H$.

From a practical point of view, Equation 9 greatly simplifies the evaluation of the wavelet product. Given two wavelet trees, we can traverse them in parallel and precompute all c_{sum} values in a single tree traversal. As only coefficients for identical basis functions contribute, the recursion is terminated whenever a leaf node is found in one of the two trees. After computing a tree of c_{sum} values, importance sampling is reduced to hierarchically evaluating Equation 9 and warping the samples according to the intensities at each level.

4.2 Bottom-up Quadtree Product

In this section, we look at the problem of multiplying two quadtree representations, G and H. That is, the p_{sum} values are known, but not the wavelet coefficients. Our application presented in Section 5 is an important example of a case where this is useful. We build a quadtree approximation of the BRDF on-the-fly, and we wish to multiply it with an environment mipmap hierarchy, in order to sample the result.

First, we note that for all *leaf* nodes in G and H, we can compute the product average by simply multiplying the individual averages, as follows:

$$p_{sumF}(q) = p_{sumG}(q) \cdot p_{sumH}(q), \tag{10}$$

where q is a leaf node in G and/or H. This follows from the fact that a quadtree leaf node is per definition constant. Hence, all its wavelet coefficients under the support of q are zero, and we can drop the c_{sumGH} term in Equation 9. Similarly, in all of H's children nodes, q_c, under the support of a leaf node q in G, the product is given by:

$$p_{sumF}(q_c) = p_{sumG}(q) \cdot p_{sumH}(q_c), \tag{11}$$

and vice versa. Thus, we can multiply a leaf node with any other node under its support, to get the corresponding product average.

For *interior* nodes, Equation 10 does not hold. However, the product average of a node, q, can always be expressed as the average of its four immediate children nodes in the product tree, as follows:

$$p_{sumF}(q) = \frac{1}{4}\sum_{i=1}^{4} p_{sumF}(q+i), \tag{12}$$

where we assume $q+i$ denotes the i^{th} child of q. This leads to a simple bottom-up algorithm for computing the product quadtree. We traverse the trees of G and H in parallel, and for all leaves, we compute the product using Equation 10, and then propagate the result up using Equation 12. This can be done in a single depth-first traversal.

Once the product tree is setup, we sample it using hierarchical sample warping [5], starting at its root. When a leaf is reached, we proceed by evaluating Equation 11 only for the nodes where it is needed, i.e., for nodes with one or more samples. This algorithm gives an efficient way to sample the product of two quadtree representations, and in addition, we completely avoid the added complexity of using wavelets. Pseudo-code and more details on our application are given in the following section.

5 Implementation

Next, we discuss the implementation of our algorithm for direct illumination under environment map lighting.

5.1 Sampling in World Space

A fundamental limitation of the wavelet product and the quadtree product, is that the functions must be defined over the same domain. Environment map lighting is given in world space, while a general BRDF is a 4D function in local space. Reparameterization to world space gives a 7D function (or 6D for isotropic materials), which is clearly impractical. Clarberg et al. [5] solve the problem by pre-rotating the lighting into local space for a dense set of directions. This limits their algorithm to low resolution lighting, e.g., 128^2 or 256^2 ($\sim$1GB compressed).

Using a low resolution approximation of the lighting to guide the sampling, introduces a substantial amount of noise when a higher resolution environment map is used, as we will see in Section 6. For realistic high-frequency lighting, environment maps of 1k$\times$1k to 4k$\times$4k resolution are common. As pre-rotation of such large maps is currently infeasible due to memory usage and precomputation time, importance sampling must be performed directly in world space. This also requires the BRDF to be in world space.

One option is to rotate a 2D slice of the BRDF into world space on-the-fly using wavelet rotation matrices [32]. This can be made fast enough by exploiting the sparsity of the matrices and the BRDF, but the memory is still a limiting factor. For example, with 64^2 distinct rotations, and a source and target resolution of 64^2, the rotation matrices require about 2GB. We have tried this approach, but the results were satisfactory only for diffuse materials. For glossy BRDFs, the misalignment of the specular peak due to the discretization introduces a large amount of noise.

The solution we settled for, is instead to build a BRDF approximation in world space on-the-fly, based on a small number of point samples. The approximation is multiplied by the environment map using the fast quadtree-based product, and the result is sampled using warping.

5.2 Environment Map

The environment map is stored as a single uncompressed high resolution image, together with its mipmap hierarchy. Each pixel in the hierarchy stores the average over its four immediate children pixels. With the quadtree-based product (Section 4.2), we do not need to store the wavelet coefficients. Hence, the memory requirement is only 33% larger than the environment map itself. As we do not rely on tabulated materials, the total setup time is reduced to computing a single mipmap hierarchy. For a 4k$\times$4k map, this takes less than one second.

To represent directions on the sphere, we use a mapping from the sphere to a *single* quad, which is based on the octahedral map [22], but with an area-preserving parameterization. The details are given in Appendix C.

5.3 BRDF Approximation

For a specific viewing direction, ω_o, the *bidirectional reflectance distribution function* (BRDF) is a two-dimensional function over all incident directions, ω. We define the local *reflectance* function, B, as the BRDF, f_r, times the cosine term in world space, as follows:

$$B(\omega) = \begin{cases} f_r(\omega_o, \omega)(\omega \cdot n), & \omega \cdot n > 0, \\ 0, & \omega \cdot n \leq 0, \end{cases} \tag{13}$$

where n is the surface normal. Assume we have a set of point samples, $S = \{B(\omega_1), \ldots, B(\omega_N)\}$, taken from the reflectance function. In the next section, we will describe how S is chosen. The problem is to reconstruct a continuous surface, $\tilde{B}$, which is a reasonable approximation to B. This is a scattered data interpolation problem, and ideally, we would like to use higher-order techniques. However, in our framework, we are limited to a piecewise constant quadtree approximation.

We build a quadtree by recursively dividing the set of samples, S, until only one sample per node remains. This is illustrated in Figure 4. Each internal node stores the average (*brdf_psum*) of its four children, and empty leaf nodes are assigned

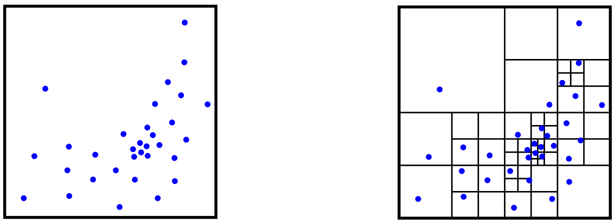


Figure 4: *Based on a set of point samples mapped onto the unit square (left), we build a quadtree approximation of the reflectance function by recursively subdividing the set until only one sample per node remains (right).*

the value of their nearest parent. We also augment our quadtree nodes with a field storing the product average (*prod_psum*) over the node, as follows:

```
struct node
    brdf_psum : BRDF parent sum (average)
    prod_psum : product parent sum (average)
    ch[4] : children pointers
end
```

Pseudo-code for building the BRDF approximation and computing the product tree is given in Algorithm 1. Input to the algorithm is the set of all BRDF samples, $S = \{B(\omega_i)\}$, and the root node of the tree, located at $q = \langle 0,0,0 \rangle$.

5.4 Obtaining the BRDF Samples

The quality of $\tilde{B}$ naturally depends on how the samples, S, are chosen. Dense sampling results in a finer subdivision, and the area of each leaf node is approximately proportional to the inverse of the local sampling density. We assume the samples are drawn from some probability density $p_B(\omega)$.

Uniform sampling works well for diffuse materials. However, for more specular BRDFs, we would likely miss high-frequency features. Therefore, we adopt an importance sampling strategy. The error in the reconstruction, ε, is equal to the difference between the original and the approximated function. Since $\tilde{B}$ will be used as an importance function, it is desirable to distribute the approximation error evenly. It is likely that ε is larger in high-intensity regions than in regions with low intensity. Hence, ideally, we want the sampling density to be proportional to the reflectance function, i.e., $p_B(\omega) \propto B(\omega)$. This way, we get higher precision around important features, such as bright specular peaks, and less resolution in smoother regions.

However, for many analytical reflectance models, sampling according to the BRDF times the cosine term is non-trivial. We often have to choose $p_B(\omega) \propto f_r(\omega_o, \omega)$.

```
function BUILDPRODUCT(quad $q$, node $n$, samples $S$)
    if size($S$) = 1 then
        $n.brdf\_psum = S[1].value$
        $n.prod\_psum = S[1].value \times L.psum(q)$
    else
        $n.ch[1..4]$ = new node()          // initialized to null
        Split $S$ into $bin[1..4]$ based on $S[i].position$
        $j$ = indices of non-empty bins
        for $i \in j$
            BUILDPRODUCT($q+i$, $n.ch[i]$, $bin[i]$)
        $n.brdf\_psum$ = avg( $n.ch[j].brdf\_psum$ )
        for $i \notin j$
            $n.ch[i].brdf\_psum = n.brdf\_psum$
            $n.ch[i].prod\_psum = n.brdf\_psum \times L.psum(q+i)$
        $n.prod\_psum$ = avg( $n.ch[1..4].prod\_psum$ )
    return
```

Algorithm 1: *Recursive function for building the approximation, $\tilde{B}$, based on a set of point samples, S, while simultaneously computing all product averages in a single tree traversal. The mipmap hierarchy of the environment map is denoted L, and the* avg() *function computes the average of the supplied values. The current node is identified by q, and $q+i$ is assumed to identify the i^{th} child of q.*

The exact choice of sampling density is not critical as it does not affect the correctness of our algorithm, but only the quality of the resulting importance function. As the sampling is done on-the-fly, it is in many cases preferable to choose a slightly inferior, but faster, sampling strategy.

Some examples of BRDFs for which analytical sampling is well known include the modified Phong model [15], and the anisotropic Ward and Ashikhmin models [33, 2] among others. Many of these sampling strategies are already implemented in existing rendering packages. This is a great advantage, as it makes the implementation of our algorithm a relatively easy task.

For measured materials, a number of sampling methods exist. We can use, for example, wavelet-based methods [16, 18, 6] or factorization [17]. Also note that the BRDF sampling step can be precomputed for all materials that are not spatially varying. We discretize the outgoing direction, and for each direction, a set of samples in local space is computed. At runtime, the samples are rotated into world space, which is very fast. We have found that a few hundred point samples is enough to build a quadtree approximation with a quality equivalent to, or better than, wavelet-compressed tabulated BRDFs [5].

```
function WARPPRODUCT(points $P$, quad $q$, node $n$)
    if $q.level = max\_level$ then
        Compute probability density for each $P[i]$
        Store $P$ in sample array
        return

    for $i = 1$ to 4
        if is_leaf($n$) then
            $w[i]$ = intensity($n.brdf\_psum \times L.psum(q+i)$)
        else
            $w[i]$ = intensity( $n.ch[i].prod\_psum$ )
    Compute splitting planes based on $w[1..4]$
    Warp $P$ into $bin[1..4]$

    for $i = 1$ to 4
        if size($bin[i]$) > 0 then
            if is_leaf($n$) then
                WARPPRODUCT($bin[i]$, $q+i$, $n$)
            else
                WARPPRODUCT($bin[i]$, $q+i$, $n.ch[i]$)
    return
```

Algorithm 2: *Recursive algorithm for warping an initially uniform point set, P, according to the product quadtree. When a leaf node is reached, the sampling continues according to the environment map, L, up to its full resolution. The* intensity() *function computes the luminance of an RGB color.*

5.5 Product Sampling

After computing the product quadtree (Algorithm 1), we proceed by sampling it using sample warping [5]. At each level, a horizontal and two vertical splitting planes are computed based on the product averages of the four immediate children, and the samples are rescaled accordingly, as illustrated in Figure 2.

When a leaf node in the product tree is reached, the sampling continues according to the environment mipmap hierarchy. Thus, although the BRDF approximation is of limited resolution, the sample warping is not terminated until the full resolution of the environment map is reached. Pseudo-code is given in Algorithm 2.

6 Results

We have implemented the algorithm described in Section 5 in a custom ray tracer loosely based on `pbrt` [21]. All images are unbiased and were rendered on a

MacBook Pro with Intel Core 2 Duo 2.40GHz (using only *one* core). The current implementation does not use any SIMD-optimizations. However, this would be fairly straightforward to add, since many of our operations are performed on four children nodes in parallel. For all tests, the BRDFs were sampled on-the-fly, but it should be noted that precomputed samples (Section 5.4) can be used to further improve the performance in many cases.

Low-discrepancy points with good spectral properties are essential for lowering the variance in any Monte Carlo technique. We use the method of Dunbar and Humphreys [10] to quickly generate Poisson-disk points that are fed to the sampling algorithm.

Figure 5 compares our algorithm with two state-of-the-art methods for product importance sampling: *wavelet importance sampling* (WIS) by Clarberg et al. [5], using the unbiased version of their algorithm, and Cline et al.'s [7] *two stage importance sampling*. Both these methods are based on sampling the product of lighting and BRDF. The scene is lit by a 1k$\times$1k light probe featuring a small strong light source: the sun. The dragon uses a Phong shader, and the ground plane is purely diffuse. For sampling the BRDFs, 256 point samples each were used for the diffuse and specular lobes. The rendering times at 1024$\times$768 pixels resolution, using one primary ray per pixel and varying number of visibility samples, were (*min:sec*):

#samples	Cline et al.	Clarberg et al.	Our algorithm
10	1:00	2:40	1:39
30	2:22	3:16	2:15
100	7:22	5:12	4:09
300	19:13	10:15	9:14

This scene presents a major challenge for WIS, which is limited to a low resolution wavelet approximation of the lighting (e.g., 128^2 or 256^2). The error introduced by the approximation, i.e., $L(\omega)/\tilde{L}(\omega)$, significantly increases the variance, especially in unoccluded regions. In addition, their method shows banding in noisy regions, which comes from the varying accuracy of the importance function due to bilinear interpolation of the wavelet terms. We completely avoid these problems by sampling in world space.

Two stage importance sampling handles this scene much better, and similar to our algorithm, it supports spatially varying reflectance functions. However, for equal variance, our algorithm gives a $1.5\times-2.7\times$ reduction in the number of visibility samples, as shown by the plot in Figure 5. It should be noted that a direct comparison of the results is difficult, as the rendering systems differ slightly and the exact speed depends on what type of shaders are used. The reported timings suggest that our ray tracer is faster than Cline et al.'s, although the acceleration data structure is the same (`pbrt`). Figure 6 shows an equal-time comparison for a simple scene lit by the "kitchen" light probe.

Our algorithm is essentially a two step method. First, the shader is sampled, and then the product is computed and sampled. An important consideration is the allo-

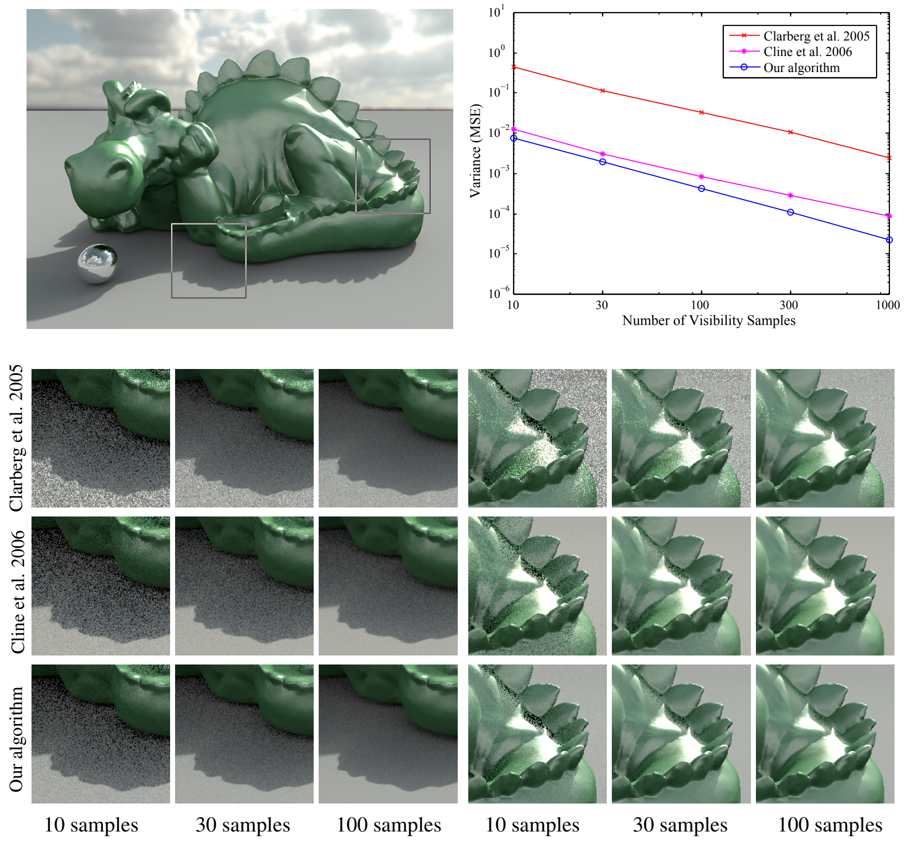


Figure 5: *We compare our method (bottom row) with wavelet importance sampling (top row) [5], and two stage importance sampling (middle row) [7], using 10, 30, and 100 visibility samples/pixel. For this test, we used 256 BRDF samples for the diffuse floor, and 512 BRDF samples for the glossy green material. All images are unbiased, and ground truth is shown on the top left. Clarberg et al.'s method is noisy in unoccluded areas, as their low-resolution lighting approximation fails to capture the precise location of the small bright light (the sun). Cline et al.'s method gives better results, but exhibits more noise than our algorithm in shadow regions due to their more crudely approximated importance functions. For equal variance, our method gives a 1.5×–2.7× reduction in the number of visibility samples compared to their method. The variance was measured on the rendered HDR images before tone-mapping. The light probe is courtesy of Paul Debevec.*

cation of samples between the two steps. In Figure 7, we have varied the number of BRDF samples (specular+diffuse), while keeping the rendering time constant by adjusting the number of visibility samples. The optimal ratio is, of course,

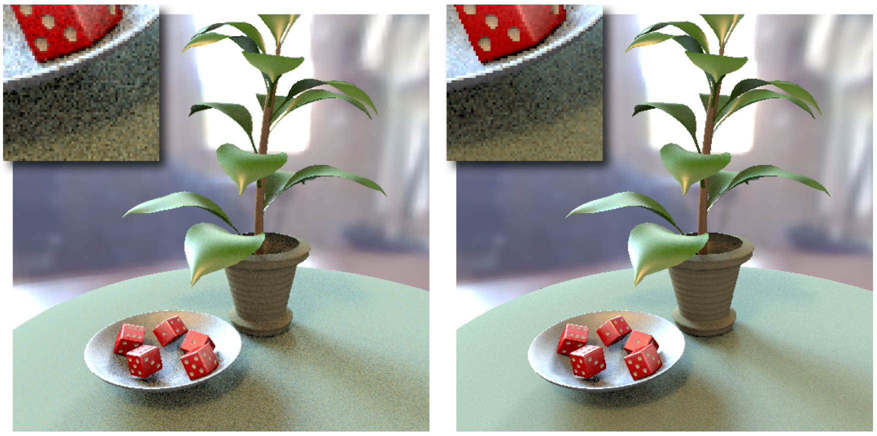


Figure 6: *Two images rendered in equal time (28 seconds) using two stage importance sampling (left) [7], and our algorithm (right). The noise is significantly reduced.*

determined by the relative speed of BRDF evaluations versus ray tracing. In our implementation, peak performance is reached with about 100–500 BRDF samples. Interestingly, Figure 7 also shows that the cost of constructing and sampling the importance function grows only linearly with the number of point samples used for approximating the material.

7 Discussion

We have presented several practical improvements to wave-let importance sampling [5]. To avoid the limitations of tabulated materials, we build a BRDF approximation on-the-fly. We also replace the wavelet product with a simpler quadtree-based product, computed in a single traversal, and thus effectively avoid wavelets altogether. The precomputation is reduced to a creation of a mipmap hierarchy for the lighting, and the memory requirements are very modest.

The proposed algorithm has much in common with Cline et al.'s method [7]. The main differences are the evaluation of the product (quadtree *vs* summed area table) and the construction of the BRDF approximation. We rely on BRDF importance sampling, while Cline et al. use heuristics specifically designed for each supported BRDF. Their approach does not require importance sampling of the shader, which is a big advantage, but finding good heuristics for general materials can be tedious. On the other hand, our algorithm may be difficult to use with some complex shaders, for which none or only poor sampling strategies exist.

The two methods produce comparable results, although the noise in occluded regions is lower with our algorithm. This indicates that our quadtree-based BRDF

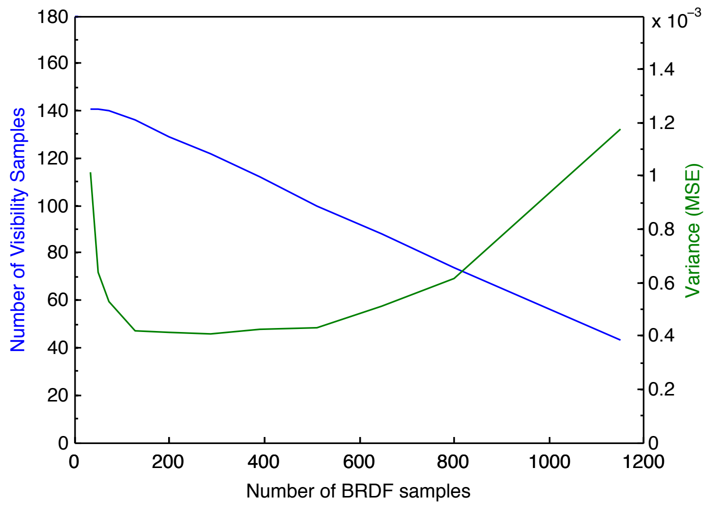


Figure 7: *Equal-time comparison for different sample allocations using the scene in Figure 5. The green line shows the variance for different numbers of BRDF samples (x-axis) and visibility samples (blue line). In this example, the best results are obtained with approximately 100–500 BRDF samples. This shows that our algorithm is robust with respect to the choice of sampling rates, and a reasonable default value (e.g., 256 BRDF samples) works well in most cases.*

approximation is slightly more accurate. In production rendering, the main bottleneck is currently the shading. Methods for creating good BRDF approximations based on a minimal number of shader evaluations are needed. We hope our work will stimulate research in that direction.

Acknowledgements

We would like to thank the anonymous reviewers for their valuable feedback, Jacob Munkberg for the scene in Figure 14, and the people in our lab for their support. This work was supported by the Swedish Foundation for Strategic Research and by Intel Corporation.

A Wavelet Primer: The 2D Haar Basis

The two-dimensional (nonstandard) Haar basis is an orthonormal basis made up of translations and dilations of mother basis functions defined over the unit square. The normalized *scaling basis functions* and *wavelet basis functions* are defined as [25]:

$$\begin{aligned} \phi_{uv}^{l}(x,y) &= 2^{l}\phi(2^{l}x-u,2^{l}y-v), && (14)\\ {\psi_M}_{uv}^{l}(x,y) &= 2^{l}\psi_M(2^{l}x-u,2^{l}y-v), && (15) \end{aligned}$$

where u,v are integer translations in $[0,2^l-1]$, and l is a positive integer representing the *scale*, which goes from coarse to fine. The mother wavelet functions, ψ_M, are defined in Figure 8, and the mother scaling function is $\phi(x,y)=1$ for $x,y \in [0,1]^2$, and 0 elsewhere. A two-dimensional image, F, with $2^k \times 2^k$ elements can be exactly represented in the basis consisting of the first scaling function and all wavelet functions up to scale $k-1$. For convenience, we denote this basis $\Psi = \{\phi_0, \psi_1, \ldots, \psi_N\}$, where $\phi_0 = \phi_{0,0}^{0}$ and ψ_j are the wavelet functions, sequentially numbered. The expansion of F can be written:

$$F = \sum_{i=0}^{N} f_i \Psi_i, \tag{16}$$

where f_i are the wavelet coefficients, which are given by the inner product: $f_i = \int F(x,y)\Psi_i(x,y)dxdy$. We call f_0 the *scaling* coefficient, and the rest *detail* coefficients.

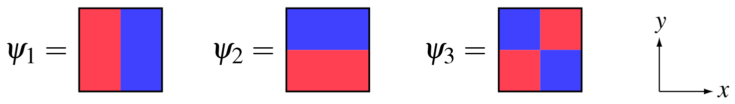


Figure 8: *The two-dimensional Haar mother wavelet basis functions, ψ_M, are defined over the unit square with the values $+1$ where red, -1 where blue, and 0 elsewhere.*

B Quadtree Encoding and Wavelet Products

The scale and translation of a basis function uniquely identifies the *wavelet square* in which it resides. We use $\langle l,u,v \rangle$ to denote a square at scale l and offset u,v. All squares at the same level are disjoint, and each has an area of $A = 2^{-2l}$. Since a square has four children, it is natural to encode 2D wavelet coefficients in a *quadtree* structure [3, 26]. With sparse wavelet representations, one or more of the coefficients and/or children of a node may be missing. Hence, empty interior nodes are possible.

The product of two 2D images, $F = G \cdot H$, can be efficiently computed directly in the wavelet basis [19]. The theory has later been generalized to higher dimensions [5], and to include multiple terms [26]. The quadtree structure of the Haar basis is essential for reducing the cost. Sun and Mukherjee [26] showed that the product can be described as directed paths through the tree of basis functions. In the same spirit, we exploit the structure to optimize the sampling of wavelet products.

C Area-Preserving Mapping of the Sphere

To simplify the sampling, we need an area-preserving mapping that, ideally, maps a single square to the sphere, with low distortion and fast analytical forward and inverse transforms. We combine the octahedral map [22] with the parametrization of Shirley and Chiu [23] to obtain a mapping with all the desired properties, as illustrated in Figure 9. As the mapping is an important practical aspect of our implementation, we repeat the formulas here.

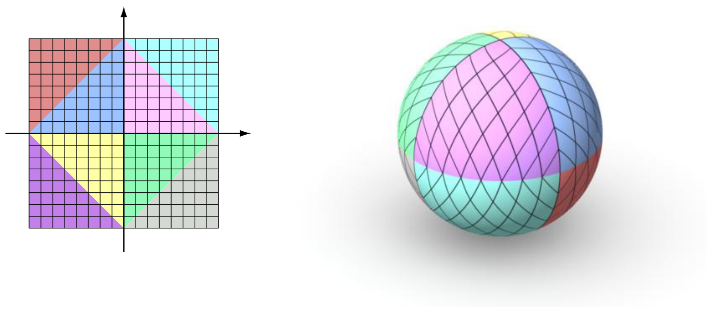


Figure 9: The square is divided into $n \times n$ pixels, which are mapped to the same number of irregularly shaped quads with equal area on the sphere.

The "inner" quad (rotated by 45°) maps to the northern hemisphere, while the outer four triangles are folded down to cover the southern hemisphere. Shirley and Chiu map a square to the unit disk, and then to the hemisphere, to obtain an area-preserving mapping. Figure 10 illustrates the square-to-disk mapping for the inner triangle of the first quadrant. Given a point (u, v) in the triangle, the lengths of a and b are $a = (u+v)/\sqrt{2}$ and $b = \sqrt{2}v$. The mapping to the disk is:

$$\begin{aligned} r &= \sqrt{2}a = u + v, \\ \phi &= \frac{\pi}{4}\frac{b}{a} = \frac{\pi}{2}\frac{v}{u+v}. \end{aligned} \tag{17}$$

Similar transforms apply to the other quadrants. The point (r, ϕ) is then projected onto the northern hemisphere, while preserving fractional area, as follows [23]:

$$(x, y, z) = (r\sqrt{2 - r^2}\cos\phi,\ r\sqrt{2 - r^2}\sin\phi,\ 1 - r^2). \tag{18}$$

This mapping uses the same number of trigonometric operations as the cylindrical equal-area projection, but the distortion is much more well-behaved. The inverse transform is well-defined and fast to compute. Interpolation across the seams is also easy due to the boundary symmetry of the octahedral map [22].

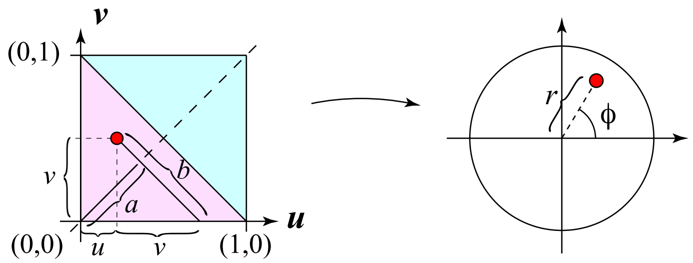


Figure 10: *The mapping of the first quadrant to the disk.*

Bibliography

[1] AGARWAL, S., RAMAMOORTHI, R., BELONGIE, S., AND JENSEN, H. W. Structured Importance Sampling of Environment Maps. *ACM Transactions on Graphics, 22*, 3 (2003), 605–612.

[2] ASHIKHMIN, M., AND SHIRLEY, P. An Anisotropic Phong BRDF Model. *Journal of Graphics Tools, 5*, 2 (2000), 25–32.

[3] BERMAN, D. F., BARTELL, J. T., AND SALESIN, D. H. Multiresolution Painting and Compositing. In *Proceedings of ACM SIGGRAPH* (1994), pp. 85–90.

[4] BURKE, D., GHOSH, A., AND HEIDRICH, W. Bidirectional Importance Sampling for Direct Illumination. In *Eurographics Symposium on Rendering* (2005), pp. 147–156.

[5] CLARBERG, P., JAROSZ, W., AKENINE-MÖLLER, T., AND JENSEN, H. W. Wavelet Importance Sampling: Efficiently Evaluating Products of Complex Functions. *ACM Transactions on Graphics, 24*, 3 (2005), 1166–1175.

[6] CLAUSTRES, L., PAULIN, M., AND BOUCHER, Y. BRDF Measurement Modelling using Wavelets for Efficient Path Tracing. *Computer Graphics Forum, 22*, 4 (2003), 701–716.

[7] CLINE, D., EGBERT, P. K., TALBOT, J. F., AND CARDON, D. L. Two Stage Importance Sampling for Direct Lighting. In *Eurographics Symposium on Rendering* (2006), pp. 103–113.

[8] DEBEVEC, P. Rendering Synthetic Objects into Real Scenes: Bridging Traditional and Image-Based Graphics with Global Illumination and High Dynamic Range Photography. In *Proceedings of ACM SIGGRAPH* (1998), pp. 189–198.

[9] DONIKIAN, M., WALTER, B., BALA, K., FERNANDEZ, S., AND GREENBERG, D. P. Accurate Direct Illumination Using Iterative Adaptive Sampling. *IEEE Transactions on Visualization and Computer Graphics, 12*, 3 (2006), 353–364.

[10] DUNBAR, D., AND HUMPHREYS, G. A Spatial Data Structure for Fast Poisson-disk Sample Generation. *ACM Transactions on Graphics, 25*, 3 (2006), 503–508.

[11] DUTRÉ, P., BEKAERT, P., AND BALA, K. *Advanced Global Illumination*, second ed. A K Peters, 2006.

[12] GHOSH, A., AND HEIDRICH, W. Correlated Visibility Sampling for Direct Illumination. *The Visual Computer, 22*, 9 (2006), 693–701.

[13] HAŠAN, M., PELLACINI, F., AND BALA, K. Matrix Row-Column Sampling for the Many-Light Problem. *ACM Transactions on Graphics, 26*, 3 (2007), 26.

[14] KAJIYA, J. T. The Rendering Equation. *Computer Graphics (Proceedings of ACM SIGGRAPH), 20*, 4 (1986), 143–150.

[15] LAFORTUNE, E. P., AND WILLEMS, Y. D. Using the Modified Phong BRDF for Physically Based Rendering. Tech. Rep. CW197, Katholieke Universiteit Leuven, 1994.

[16] LALONDE, P. *Representations and Uses of Light Distribution Functions*. PhD thesis, University of British Columbia, 1997.

[17] LAWRENCE, J., RUSINKIEWICZ, S., AND RAMAMOORTHI, R. Efficient BRDF Importance Sampling using a Factored Representation. *ACM Transactions on Graphics, 23*, 3 (2004), 496–505.

[18] MATUSIK, W. *A Data-Driven Reflectance Model*. PhD thesis, MIT, 2003.

[19] NG, R., RAMAMOORTHI, R., AND HANRAHAN, P. Triple Product Wavelet Integrals for All-Frequency Relighting. *ACM Transactions on Graphics, 23*, 3 (2004), 477–487.

[20] OSTROMOUKHOV, V., DONOHUE, C., AND JODOIN, P.-M. Fast Hierarchical Importance Sampling with Blue Noise Properties. *ACM Transactions on Graphics, 23*, 3 (2004), 488–495.

[21] PHARR, M., AND HUMPHREYS, G. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann, 2004.

[22] PRAUN, E., AND HOPPE, H. Spherical Parametrization and Remeshing. *ACM Transactions on Graphics, 22*, 3 (2003), 340–349.

[23] SHIRLEY, P., AND CHIU, K. A Low Distortion Map between Disk and Square. *Journal of Graphics Tools, 2*, 3 (1997), 45–52.

[24] SHIRLEY, P. S. *Physically Based Lighting Calculations for Computer Graphics*. PhD thesis, University of Illinois at Urbana-Champaign, 1991.

[25] STOLLNITZ, E. J., DEROSE, T. D., AND SALESIN, D. H. *Wavelets for Computer Graphics: Theory and Applications*. Morgan Kaufmann, 1996.

[26] SUN, W., AND MUKHERJEE, A. Generalized Wavelet Product Integral for Rendering Dynamic Glossy Objects. *ACM Transactions on Graphics, 25*, 3 (2006), 955–966.

[27] TALBOT, J., CLINE, D., AND EGBERT, P. Importance Resampling for Global Illumination. In *Eurographics Symposium on Rendering* (2005), pp. 139–146.

[28] VEACH, E. *Robust Monte Carlo Methods for Light Transport Simulation*. PhD thesis, Stanford University, 1997.

[29] VEACH, E., AND GUIBAS, L. J. Optimally Combining Sampling Techniques for Monte Carlo Rendering. In *Proceedings of ACM SIGGRAPH* (1995), pp. 419–428.

[30] WALTER, B., ARBREE, A., BALA, K., AND GREENBERG, D. P. Multidimensional Lightcuts. *ACM Transactions on Graphics, 25*, 3 (2006), 1081–1088.

[31] WALTER, B., FERNANDEZ, S., ARBREE, A., BALA, K., DONIKIAN, M., AND GREENBERG, D. P. Lightcuts: A Scalable Approach to Illumination. *ACM Transactions on Graphics, 24*, 3 (2005), 1098–1107.

[32] WANG, R., NG, R., LUEBKE, D., AND HUMPHREYS, G. Efficient Wavelet Rotation for Environment Map Rendering. In *Eurographics Symposium on Rendering* (2006), pp. 173–182.

[33] WARD, G. J. Measuring and Modeling Anisotropic Reflection. *Computer Graphics (Proceedings of ACM SIGGRAPH), 26*, 2 (1992), 265–272.

[34] WILLIAMS, L. Pyramidal Parametrics. *Computer Graphics (Proceedings of ACM SIGGRAPH), 17*, 3 (1983), 1–11.

Paper III

Exploiting Visibility Correlation in Direct Illumination

Petrik Clarberg Tomas Akenine-Möller

Lund University

Abstract

The visibility function in direct illumination describes the binary visibility over a light source, e.g., an environment map. Intuitively, the visibility is often strongly correlated between nearby locations in time and space, but exploiting this correlation without introducing noticeable errors is a hard problem. In this paper, we first study the statistical characteristics of the visibility function. Then, we propose a robust and unbiased method for using estimated visibility information to improve the quality of Monte Carlo evaluation of direct illumination. Our method is based on the theory of control variates, and it can be used on top of existing state-of-the-art schemes for importance sampling. The visibility estimation is obtained by sparsely sampling and caching the 4D visibility field in a compact bitwise representation. In addition to Monte Carlo rendering, the stored visibility information can be used in a number of other applications, for example, ambient occlusion and lighting design.

Computer Graphics Forum (Proceedings of EGSR 2008), 27(4):1125–1136, 2008.

1 Introduction

In photo-realistic rendering, the lighting is often divided into direct and indirect illumination, which are evaluated separately. The *indirect* illumination is typically low-frequency, as it is the accumulated result of a long (infinite) series of bounces, and efficient algorithms for sparsely sampling and reusing indirect illumination exist. The *direct* illumination is given by an integral over the incident lighting, surface reflectance, and *visibility* [16]. As we will see in Section 4, the visibility is often strongly correlated between nearby points. Our goal is to exploit this correlation to obtain faster Monte Carlo (MC) rendering with higher quality.

The direct illumination often shows high-frequency features such as sharp shadows, bright specular reflections, and so on. Due to this high-frequency behavior, the algorithms used for indirect illumination, e.g., photon mapping [15], (ir)radiance caching [33, 19, 11], and statistical PCA-based filtering [22], are not directly applicable. These algorithms essentially perform a low-pass filtering of the radiance field, which may lead to visible artifacts such as blurred shadows when applied to direct illumination.

Current state-of-the-art methods for Monte Carlo evaluation of direct illumination are based on importance sampling the product of lighting and reflectance [3, 27, 5, 6, 4]. However, none of these methods take visibility into account. A simple example where product sampling gives poor results is a glossy surface reflecting a bright light source, which is partially occluded. Product sampling directs most samples toward the occluded light, but occasionally a sample hits the light, resulting in a locally high noise level. An example is shown in Figure 1.

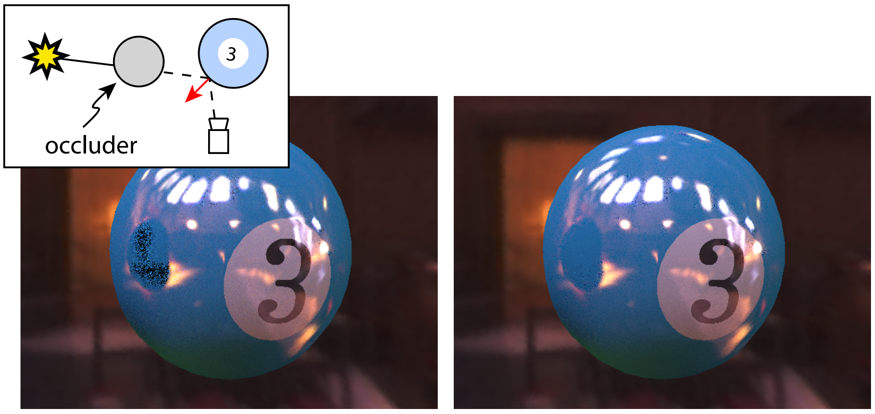


Figure 1: *Techniques for importance sampling only the product of the lighting and BRDF suffer, in general, from excessive noise in occluded regions (left). By adding a control variate term taking visibility into account, we can significantly reduce the problem (right). Both images are unbiased and rendered using 10 samples/pixel.*

We propose a simple, yet efficient, method to exploit correlation in the visibility function to improve the results. Our approach is based on *control variates* [17], which is a classic MC technique. The idea is to subtract a correlated approximation from the function we want to integrate, and thus reduce the variance of the remaining part, as illustrated in Figure 2. The difference between the two functions is sampled, and the integral of the approximation is added back as a correction term. In our case, we estimate the visibility from nearby samples, and use the triple product of lighting, reflectance and visibility as a control variate term.

Our technique presents an unbiased way of reusing visibility information to improve the rendering quality. The key question is how to obtain the visibility approximation. We have opted for a strategy inspired by radiance caching [19]. We sample the 4D visibility field sparsely over all visible surfaces, and store the information as 2D visibility maps at selected positions. These maps are interpolated to yield a visibility approximation for the current pixel. It is important to stress that, although we cache visibility data, the primary purpose is to reduce the number of shader evaluations. By spending some extra effort on evaluating the visibility (which is relatively cheap), we reduce the number of samples needed in the importance sampling step, thus keeping the number of expensive shader evaluations at a minimum.

2 Related Work

The method of control variates, or *correlated sampling*, has been used in several computer graphics papers. Lafortune and Willems [20] used a constant ambient term as control variate. Later, they stored an approximation of the incident radiance in a 5D tree [21] in order to reduce the variance in path tracing. Szirmay-Kalos et al. [25] used a radiosity solution as a control variate for a subsequent Monte Carlo step. Szécsi et al. [24] combine correlated and importance sampling to improve the quality of direct illumination. This approach is similar to ours, but their approximation ignores visibility and assumes a diffuse BRDF. Fan et al. [8] combine samples from multiple functions, which are used as control variates. This can be seen as a generalization of multiple importance sampling [28].

A few techniques for exploiting coherence in Monte Carlo evaluation of direct illumination exist. Ghosh et al. [13] exploit *temporal* coherence in the illumination to efficiently render scenes under animated environment map lighting. Samples are generated for the first frame using traditional methods, and then updated for subsequent frames using a sequential Monte Carlo sampling strategy. In other work, Ghosh and Heidrich [12] target *spatial* coherence in visibility. First, bidirectional importance sampling is used to find pixels that are partially occluded. In a second step, they apply Metropolis-Hastings mutations to reduce the noise in these regions. This method only works for occluded regions. In contrast, our approach lowers the variance overall, and exploits both temporal and spatial coherence. Donikian et al. [7] divide the image into 8×8 blocks, and use adaptive importance

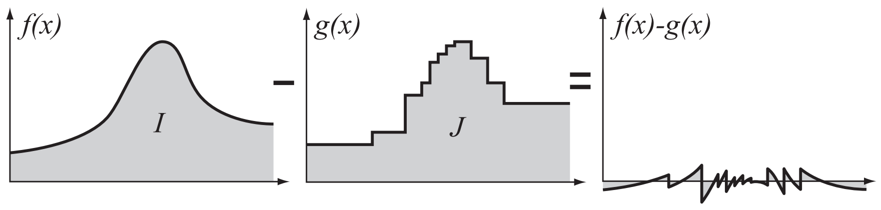


Figure 2: *The idea behind control variates is to subtract a function, g, with known integral, J, from the function, f, we want to integrate, and thus reduce the variance of the remaining part,* $f - g$.

sampling to iteratively refine probability density functions (pdf) for the blocks and pixels. In early iterations, more emphasis is put on the block pdf, thereby automatically exploiting spatial coherence. Their algorithm has many clever twists, but since they start without prior knowledge of the lighting and BRDF, many shadow rays are needed (often above 1000 rays/pixel).

Methods for reducing the number of shadow tests have a long history in computer graphics (see, e.g., [31, 23, 10]). The goal of these algorithms is to select a small set of representative lights out of a large number of light sources, usually by means of sorting or spatial data structures, but without taking the BRDF into account. Recent algorithms for many-light rendering [30, 29] share the same goal, but do not explicitly exploit visibility correlation.

Hart et al. [14] use lazy visibility evaluation to compute direct illumination in scenes with many area light sources. They build a “blocker map” for each pixel, exploiting spatial visibility coherence by a flood-fill algorithm in screen space. Similarly, Agrawala et al. [1] exploit image-space correlation to render soft shadows from area light sources. Ben-Artzi et al. [2] recently presented a clever method for reducing the number of shadow rays needed for environment map illumination. They partition the environment map into a set of preintegrated lights, and use a coarse-to-fine evaluation of the image. By sharing occluder information between neighboring pixels and directions, they can significantly reduce the number of rays traced. These methods are somewhat similar to our visibility cache, but do not exploit temporal coherence. We also use the original unapproximated environment map.

The main difference to previous methods is that we try to solve the combined problem, taking both reflectance, lighting, and visibility into account, while exploiting visibility correlation. Our algorithm builds on existing work on importance sampling and is an *unbiased* Monte Carlo technique, which means it can be used for reference renderings and with very high-resolution environment maps. Our algorithm also has a broader applicability, and can be used for efficient *ambient occlusion* and *lighting design*, i.e., fast preview of complex lighting and shadowing. An important contribution is also our statistical analysis of the visibility function.

3 Mathematical Foundation

The outgoing radiance, L_o, due to direct illumination is given by an integral over the incident lighting, the reflectance (BRDF times a cosine-term), and the visibility [16]. We denote these terms L, B, and V, respectively. The traditional Monte Carlo estimator for L_o using importance sampling is:

$$\langle L_o \rangle = \frac{1}{N}\sum_{i=1}^{N}\frac{LBV}{p(\omega)}, \tag{1}$$

where $p(\omega)$ is the importance function. Note that we have omitted the arguments of L, B and V for clarity. In the following, we denote approximations of the exact functions by adding a tilde, e.g., $\tilde{B}$. Several recent sampling algorithms explicitly compute and sample an approximation of the product LB. For example, in two-stage importance sampling [6] and quadtree-based product sampling [4], hierarchical approximations of the reflectance, $\tilde{B}$, are multiplied with the exact lighting L, i.e., $p(\omega) \propto L\tilde{B}$.

We propose to include a binary estimation of the visibility, $\tilde{V}$, to lower the variance. A natural approach would be to use the triple product $L\tilde{B}\tilde{V}$ for importance sampling. However, this requires $\tilde{V} \neq 0$ wherever $V \neq 0$, as zeroes in $\tilde{V}$ effectively stop the exploration of those regions of the integral and may lead to bias. Since we do not know which parts of V are truly zero, $\tilde{V}$ has to be nonzero over the entire domain to guarantee correctness. One approach would be to add a small constant, ε, and sample according to $\tilde{V}+\varepsilon$. However, in regions where $\tilde{V}=0$, but $V=1$, we would get excessive noise as we divide by $p(\omega)$ in Equation 1, which in this case is very small. We avoid these problems by sampling according to the product $p(\omega) \propto L\tilde{B}$ as before, and use $L\tilde{B}\tilde{V}$ as a *control variate*. Equation 1 is rewritten as follows:

$$\langle L_o \rangle = \frac{1}{N}\sum_{i=1}^{N}\frac{LBV-\alpha L\tilde{B}\tilde{V}}{p(\omega)} \;+\; \alpha \underbrace{\int L\tilde{B}\tilde{V}\,d\omega}_{J}. \tag{2}$$

It is easy to show that this is an unbiased estimator for L_o, as the MC evaluation of $\alpha L\tilde{B}\tilde{V}$ converges to αJ as $N \to \infty$.

With existing importance sampling methods that already compute hierarchical representations of $L\tilde{B}$ [6, 4], including a third term, $\tilde{V}$, in the product is relatively inexpensive. More details will be given in Section 5. One interpretation of Equation 2 is that we compute a rough estimate, J, of the direct illumination based on approximations, and then evaluate the *difference* between this and the correct solution using Monte Carlo integration. The rest of this paper deals with the computation of $\tilde{V}$.

3.1 Variance Analysis

In the following, we assume that our algorithm is implemented on top of a scheme for importance sampling the product of lighting and reflectance. In practice, we use the quadtree-based method of Clarberg et al. [4]. For this case $p(\omega) = L\tilde{B}/L_{ns}$, where L_{ns} represents the unoccluded illumination, and Equation 2 can be rewritten:

$$\langle L_o \rangle = \frac{L_{ns}}{N} \sum_{i=1}^{N} \underbrace{\left[\frac{B}{\tilde{B}} V - \alpha \tilde{V} \right]}_{Y} + \alpha J. \tag{3}$$

We use the variables Y as defined above, $Z = BV/\tilde{B}$ and $W = \tilde{V}$ to represent random observations of the respective functions, and note that the variance of the estimator in Equation 3 is equal to: $L_{ns}^2 \sigma_Y^2 / N$, where σ_Y^2 denotes the variance of Y given by:

$$\sigma_Y^2 = \sigma_Z^2 + \alpha^2 \sigma_W^2 - 2\alpha \sigma_{ZW}^2. \tag{4}$$

Here, σ_{ZW}^2 denotes the *covariance* of Z and W. The variance is minimized when $\alpha = \sigma_{ZW}^2 / \sigma_W^2$, in which case:

$$\sigma_Y^2 = \sigma_Z^2 (1 - \rho^2), \tag{5}$$

where ρ is the statistical correlation (Pearson's product-moment coefficient) between Z and W. This is defined as follows:

$$\rho = \frac{\sigma_{ZW}^2}{\sigma_Z \sigma_W}, \qquad -1 \leq \rho \leq 1. \tag{6}$$

The correlation coefficient, ρ, is a convenient measure of similarity, as it always lies in the range $[-1, 1]$, where $\rho = 1$ indicates a positive linear relationship, $\rho = -1$ a negative linear relation, and values in-between indicate a weaker correlation.

Equation 5 implies that, assuming we know the value of α, introducing a control variate term reduces the variance of the original estimator [4] by up to a factor $1 - \rho^2$. Using control variates is attractive since the variance goes to zero when the correlation ρ is ± 1, while in the worst case there is no correlation $\rho = 0$, and the variance is unchanged.

However, finding the optimal α is difficult in practice, as the values of σ_{ZW}^2 and σ_W^2 are unknown. The variance of the visibility approximation can be computed, but σ_{ZW}^2 has to be estimated based on samples. In our tests, the additional computation was not motivated by a large enough quality improvement. In addition, bias is introduced if the same samples are used for estimating σ_{ZW}^2 as for evaluating the integral. Instead, we use $\alpha = 1$, which works well because $\tilde{V}$ is reasonably close to $BV/\tilde{B}$.

4 The Visibility Function

The visibility function, V, measures the visibility between a point, $\mathbf{x}$, and a light source, L. This can be written as a function of world-space direction, ω, as follows:

$$V(\mathbf{x},\omega) = \begin{cases} 1, & \text{if } L \text{ is visible,} \\ 0, & \text{otherwise.} \end{cases} \tag{7}$$

By definition, the lower hemisphere is always occluded. If we restrict $\mathbf{x}$ to the surfaces of a scene, V becomes a four-dimensional function. Our goal is to exploit coherence in V to improve the rendering quality. An important question is, how much coherence is there in a typical scene? To answer this, we have performed extensive statistical measurements.

4.1 Correlation Measurements

Intuitively, looking at two random points, $\mathbf{x}_i$ and $\mathbf{x}_j$, the correlation between $V_i = V(\mathbf{x}_i,\omega)$ and $V_j = V(\mathbf{x}_j,\omega)$ should be stronger the smaller the distance, $d = ||\mathbf{x}_i - \mathbf{x}_j||$, is between them. An important factor is also the angular difference, $\theta = \cos^{-1}(n_i \cdot n_j)$, between the surface normals at the two points. As the lower hemisphere of V is always occluded, the similarity between V_i and V_j is likely to be smaller if the surface frames are rotated relative to each other.

We have estimated the *average correlation*, $\bar{\rho}$, as a function of distance and normal difference, i.e., $\bar{\rho}(d,\theta)$, on a suite of test scenes. This was done by distributing a set of sample positions (about 700k) over the visible surfaces, and then measuring the correlation between the visibility functions using a large number of randomly chosen pairs of positions, $\{\mathbf{x}_i,\mathbf{x}_j\}$. Each measurement gives an estimate of ρ for a specific d and θ. These were stored in bins representing small discretized intervals of d and θ. Finally, for each bin, the mean and standard deviation of the correlation estimates were computed. In total, $1.9-3.2\cdot 10^{10}$ correlation estimates per scene were done, each based on 96×96 visibility samples over the sphere. The results are summarized in Figures 3–6.

4.2 Test Scenes

Four different test scenes were used (see Figure 4). The scenes (a) and (b) represent different camera angles in a garden scene (available on the Autodesk Maya 2008 DVD) tessellated to about 2 million triangles. This scene features extremely difficult occlusion due to the small and highly detailed geometry. The third scene (c) is simpler, although it contains a number of plants not visible in the image, while the last (d) is even simpler with mostly flat surfaces. As we measure the correlation as a function of world space distance, it is important to know the relative sizes of objects. In (a) and (b), the tulips have a diameter of about 0.8 units, while the average height of the grass is 3.8 units. The mushrooms in (c) have a height and diameter of about 1.3, while the cubes in (d) have a side length of 5.0 units.

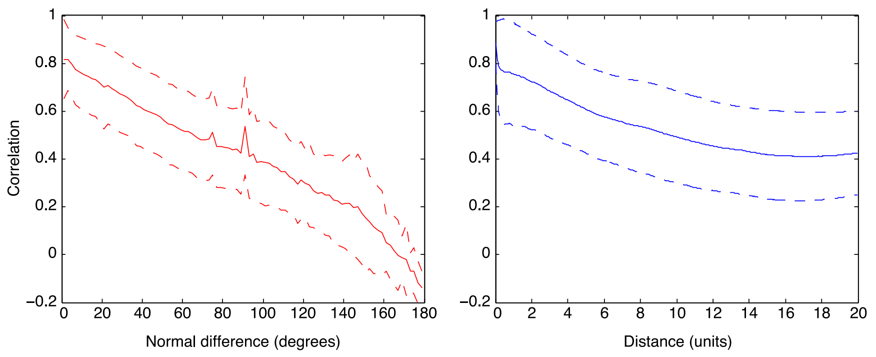


Figure 3: *Analysis of the visibility correlation in scene (a) from Figure 4. All pairs of measurement points were sorted into bins based on their distance and normal difference. The left graph shows the correlation for points that are nearby in distance ($d < 0.2$), and the right graph shows the correlation for points nearby in direction ($\theta < 2^\circ$). The dashed lines represent the standard deviation ($\pm\sigma$).*

4.3 Discussion

The average correlation, $\bar{\rho}(d,\theta)$, provides an interesting footprint of the statistical properties of V. In scene (d), for example, the geometry mainly consists of flat surfaces, i.e., the cubes' faces. Each combination of two such surfaces gives a distinct horizontal line in the correlation plot, as θ is constant, and the range of d is limited by the location and extent of the two surfaces. As expected, the overall correlation is smaller in scenes with "difficult" visibility. However, even for the garden scene, there is a significant amount of correlation between nearby locations. In general, the visibility at surface points with similar normals is highly correlated, even when they are relatively far from each other.

To visualize the spatial distribution of the visibility correlation, we compute the average correlation within a local neighborhood around each point. Only visibility samples within a distance $d_{\max}$, and with $\theta < \theta_{\max}$ were considered. The values of $d_{\max}$ and $\theta_{\max}$ were set as to include mainly the part of $\bar{\rho}$ where the correlation is high (red/orange). This can be seen as cutting out the top-left part of the correlation plot, and computing its average value at different positions in the scene.

Our results show that the visibility is often highly correlated over large smooth surfaces. More interestingly, we found that the correlation can be high even in places of very complex geometry, e.g., the grass in the background in (a) and deep inside the vegetation in (b). In some parts of (c) and (d), the average correlation is surprisingly low, even on smooth surfaces. This happens when the search distance, $d_{\max}$, is too large and samples from nearby surfaces with different visibility are included in the average. This shows that, in order to approximate the visibility function from nearby samples, a uniform distribution is not enough. We must be able to refine the approximation where needed.

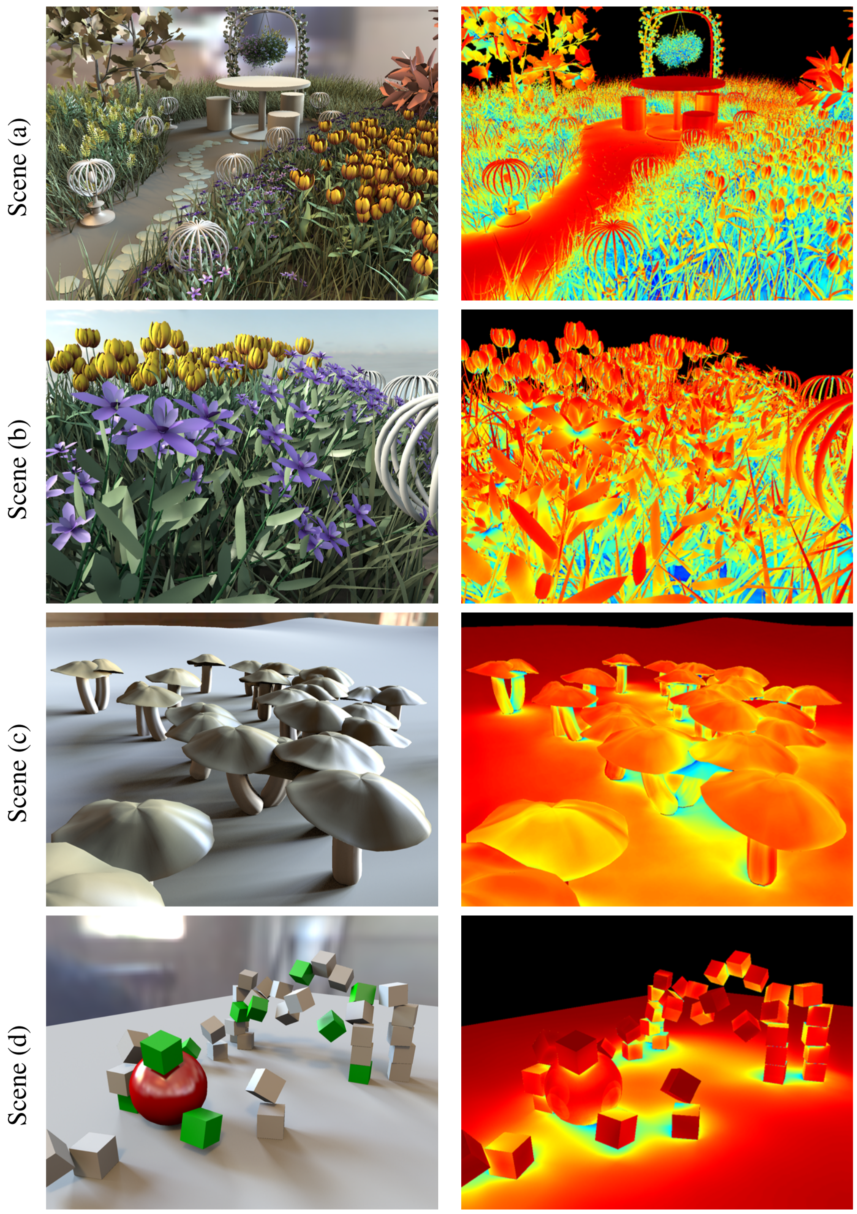


Figure 4: *Measurements of the visibility correlation in four different scenes (a)–(d). The four false-color renderings on the right show the spatial distribution of the visibility correlation. For each pixel, we have computed the average correlation to all nearby points within a certain distance and normal difference,* $(d_{\max}, \theta_{\max})$, *represented by a box in the correlation plots in Figure 5 and 6.*

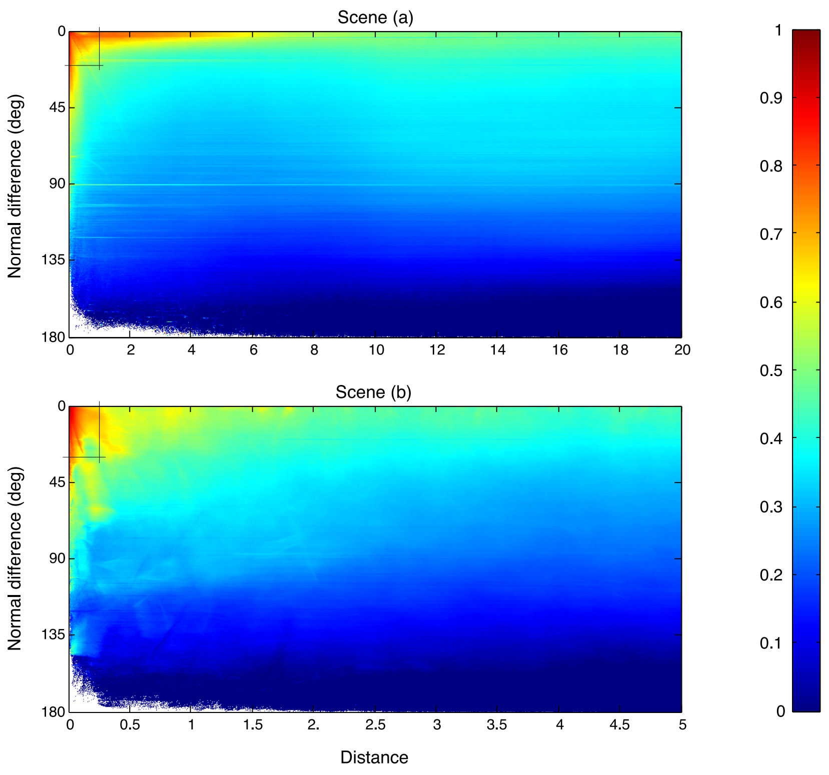


Figure 5: *The average visibility correlation as a function of normal difference, θ, and world space distance, d, in scenes (a) and (b). All correlation values have been clamped to $\rho \in [0,1]$ to make the differences more noticeable, as the correlation is rarely below 0. White represents combinations (d,θ) for which there was not enough data. In total, about $1.7-3.0\cdot 10^{14}$ pairs of visibility samples were considered for each image. As expected, the correlation is close to 1 in the top-left corners.*

5 Exploiting Coherence: The Visibility Cache

Our goal is to construct an approximation, $\tilde{V}$, which is as close to V as possible. For this purpose, we evaluate V at a sparse set of locations $\{\mathbf{x}_i\}$, and store the resulting 2D visibility maps in a median-balanced *kd*-tree for efficient range search. We call this structure the *visibility cache* as the concept is similar to (ir)radiance caching [33, 19].

Each visibility map is called a *cache record*, and encodes the visibility over the sphere in world space. Essentially, these maps are just low-resolution black and white images representing the visibility of the background. A weighted average of

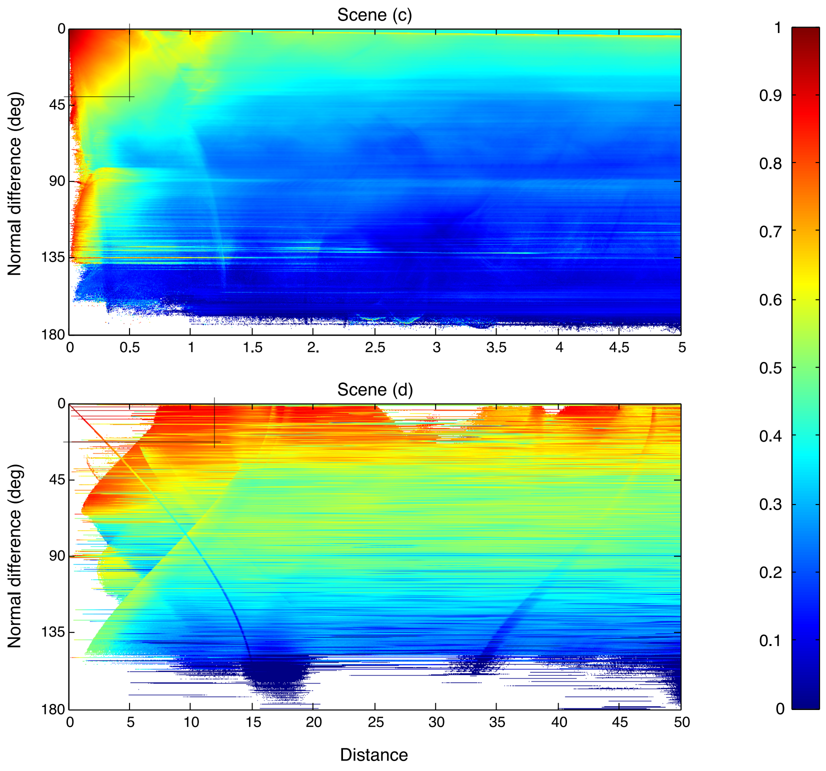


Figure 6: *The average visibility correlation as a function of normal difference, θ, and world space distance, d, in scenes (c) and (d).*

visibility maps is multiplied by the environment map and the reflectance at each pixel. This gives a good approximation of the direct illumination, which is used as a control variate term.

There are several advantages with our representation over working in the local surface frame. First, we need to quickly integrate our cache records against $L\tilde{B}$ obtained by the importance sampling algorithms [6, 4], which operate in world space. Second, interpolation becomes trivial, as we do not have to perform rotations. The approximated visibility at a point, $\mathbf{x}$, is obtained as a linear combination of n nearby cache records:

$$\tilde{V}(\mathbf{x},\omega) = \sum_{i=1}^{n} \beta_i \tilde{V}(\mathbf{x}_i,\omega) = \sum \beta_i \tilde{V}_i. \tag{8}$$

As a first step, we send a small number, N_{startup}, of rays (e.g., 1000) uniformly distributed over the image, and insert a cache record at the first hit along each ray. This bootstrapping of the cache completes very quickly and is only done for the first frame in an animation. Otherwise, the algorithm is a one-pass method, and new records are inserted along the way. Next, we will describe how the weights, β_i, are found.

5.1 Cache Lookups

In order to find a small set of n (typically 3–4) cache records that are representative for the visibility at $\mathbf{x}$, we start by performing a range search in the *kd*-tree. The search is restricted to the m (typically 10–20) nearest records that are within a distance $d_{\max}$, and that have a similar surface orientation, i.e., $\theta_i < \theta_{\max}$, where θ_i denotes the angle between the normals at $\mathbf{x}$ and $\mathbf{x}_i$. We compute a weight, $w_i = w_g \cdot \hat{w}(d, \theta)$, for each record and then pick the n records that score the highest. These weights are then used as β_i after normalization so that $\sum \beta_i = 1$.

The purpose of $\hat{w}$ is essentially to relate d to θ in a sensible way. In our measurements, we have seen that the correlation often falls off near linearly with increasing θ, while there is a quicker dropoff in the beginning with increasing distance, d, and slower at the end. Figure 3 shows a typical example of this. We have designed a function w, which mimics this behavior (see Figure 7), and is given by:

$$w(d,\theta) = (1-\theta/\pi)\left(1-\frac{x}{1+\lambda x}\right), \quad \text{where } x = \frac{d}{d_{\max}}, \tag{9}$$

and λ controls how steep the initial dropoff is (we use $\lambda = 5$). Equation 9 gives a weight in the range $[\underline{w}, 1]$, where the lower limit, $\underline{w}$, can easily be derived from $d_{\max}$ and $\theta_{\max}$. Before use, we normalize w to the unit interval by setting $\hat{w} = (w - \underline{w})/(1 - \underline{w})$.

The motivation for first looking at a larger neighborhood, and then picking a smaller set of records, is that we want to find records with matching surface orientation. Statistically seen, these should be good approximations to V, at least if they are reasonably close. In Figure 8, we illustrate a case where a larger search region is beneficial. On a rough surface, e.g., using displacement mapping, our method only includes records with similar normals, while ignoring others, even if they are

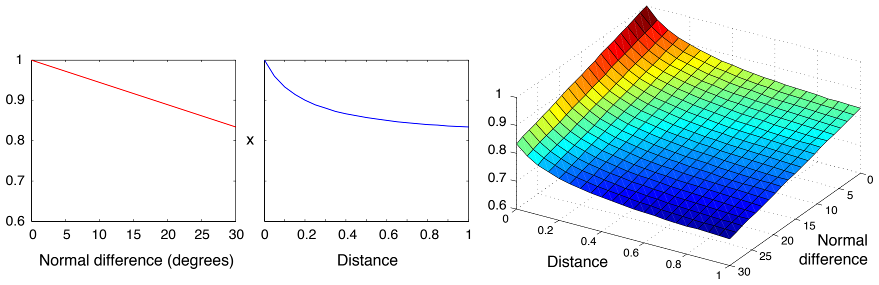


Figure 7: *The weighting function used in cache lookups.*

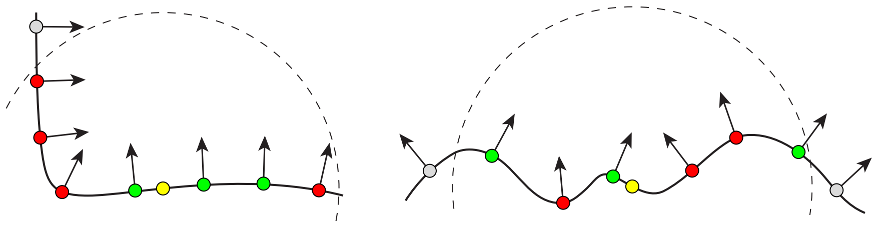


Figure 8: *To create a visibility approximation at the yellow points, we locate the m nearest cache records, and then select only the ones with similar normal directions (green).*

closer. To keep the cache density approximately constant in screen space, we also compute the area of the current pixel projected onto the plane passing through $\mathbf{x}$ with normal n. The maximum search range, $d_{\max}$, is then set to a fixed constant times the square root of the projected pixel area [26].

Finally, we include a geometric term, w_g, to reduce the weights of records that lie "in front" or "behind" the current point $\mathbf{x}$. Consider the two cases shown in Figure 9. It is desirable to reduce the weight for the record at $\mathbf{x}_i$ in the rightmost case, as part of the hemisphere at $\mathbf{x}$ must be occluded by the (unknown) geometry holding $\mathbf{x}_i$. The same holds true for the inverse case; if $\mathbf{x}$ is in front of $\mathbf{x}_i$, part of $\mathbf{x}_i$'s hemisphere must be occluded. The angle, ϑ, between the normal and the vector, v, pointing toward the record, gives an indication of the induced error. We use the simple formula $w_g = \sqrt{1 - |n \cdot v|}$ as an approximation.

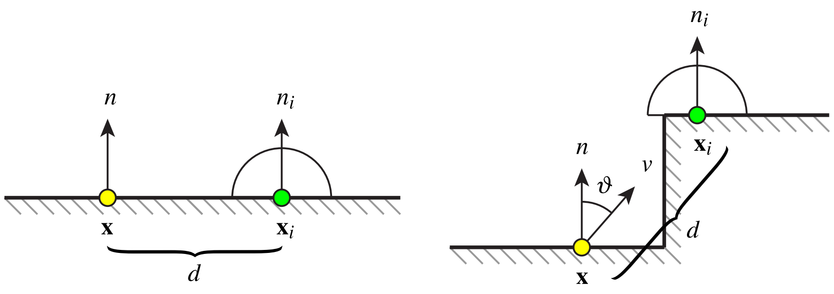


Figure 9: *If we only consider the distance and normals, the weight for the record at* $\mathbf{x}_i$ *would be the same in both these cases, as the distance, d, and the normals, n and* n_i*, are the same. However, the information stored at* $\mathbf{x}_i$ *is clearly less relevant at* $\mathbf{x}$ *for the case on the right. Thus, we include a geometric term, which takes the angle* ϑ *into account.*

5.2 Adaptive Refinement

During rendering, we insert a new cache record whenever the local cache density is too low. Our strategy is somewhat similar to adaptive refinement in (ir)radiance caching [18]. We first use a geometry-based criteria, inserting a new record when the weight of the highest ranked record according to Equation 9 is below a predetermined threshold, i.e., $\max(w_i) < w_{\min}$. This will locally increase the cache density on curved surfaces.

Second, heuristics are used to decide whether to use the existing records or insert a new. We measure the *average difference* between the n nearby records found as described in the previous section. The difference, δ_{ij}, between two visibility approximations, $\tilde{V}_i$ and $\tilde{V}_j$, is defined as the probability of a random sample returning different values. As $\tilde{V}$ are discrete visibility maps, this is simply the number of pixels with different values in $\tilde{V}_i$ and $\tilde{V}_j$, divided by the total number of pixels. The average difference, $\bar{\delta}$, is the mean over all combinations of the n records, which is fast to compute as n is very small. If the difference is above a certain threshold, i.e., $\bar{\delta} > \bar{\delta}_{\max}$, we insert a new cache record at the current position, $\mathbf{x}$.

As we explicitly compare the stored visibility information, our method automatically inserts new cache records in regions of difficult occlusion. This happens even if the surface, on which the records themselves are placed, is simple. Records near an occluder will “see” different things, and hence the cache will be locally refined until the difference is below the threshold. The results of our two refinement criteria, the geometry-based and the average visibility difference, are visualized in Figure 10.

The insertion of a new record in the *kd*-tree is done by adding it to the leaf node enclosing its position. If a leaf gets too large (e.g., more than four records), we split it along its median. As each insertion makes the tree progressively more unbalanced, we occasionally rebalance the whole tree. As the number of stored records is rather small (in the order of 10,000), this only takes a few milliseconds. Note that we use a standard 3D *kd*-tree based on position only. To further speed up the proximity search, it would be possible to use a higher-dimensional tree, splitting on both position and normal orientation. Our rendering application is multi-threaded, so we also protect all accesses to the *kd*-tree with a read-write lock. This way, multiple threads may read simultaneously, but only one at the time can write.

5.3 Exploiting Temporal Coherence

A major advantage of the visibility cache is that we can reuse our world-space cache records over time, thereby reducing the computations performed per frame. In many cases, changes in visibility are local and do not affect the whole scene, and for the case of just a moving camera, the visibility field does not change at all. As we use the visibility approximations as control variates, reusing slightly inaccurate visibility information will *not* introduce bias or artifacts, only increase

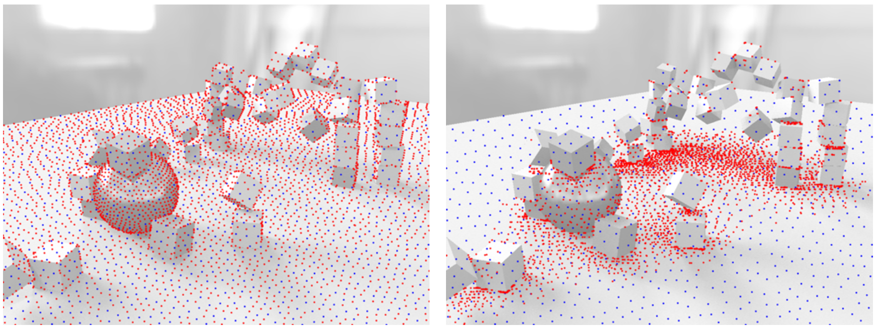


Figure 10: The red dots show the locations of cache records. The geometry-based criteria (left) puts more records on surfaces of high curvature, while the heuristic based on measuring the average visibility difference (right) focuses on regions of difficult occlusion. Note that this method efficiently finds the regions of low correlation (see Figure 4).

the noise.

To avoid the visibility information from deteriorating, we automatically remove inaccurate cache records. In computing the outgoing radiance using Equation 2, we have to evaluate both the exact visibility, V, and the approximation, $\tilde{V}$, for each of the sampling directions obtained by importance sampling. We count the number of rays for which the visibility approximation is off, and accumulate this value in the cache records. At the end of each frame, we remove all records above a predefined threshold, $\varepsilon_{\max}$, e.g., 5% misses. Another method is to sort the records according to their error rate and remove the k worst performing. To keep the cache small, we also remove all records that have not been used for a certain number (e.g., 10) of frames.

5.4 Setting the Thresholds

The different parameters controlling the behavior of the visibility cache can be classified based on which feature they control, as shown in Table 1. This makes their setup more intuitive. Some of the parameters are non-critical and reasonable default values work well, e.g., $N_{\text{startup}} = 1000$ and $\theta_{\max} = 30°$. Others can be derived automatically, e.g., $d_{\max} = 5\%$ of the image width in pixels. The thresholds controlling the insertion ($w_{\min}$ and $\bar{\delta}_{\max}$) and removal ($\varepsilon_{\max}$) of cache records have a direct impact on the size of the cache, and have to be manually set.

In general, we have seen that the cache adapts well to different scenes. With the same parameters, a scene with complex occlusion will get more cache records than a scene with simple geometry. This also means that for a dynamic animation sequence, more records will be allocated to difficult frames. We have found it useful to have a couple of predefined setups (i.e., low/medium/high scene complexity),

Category	*Parameter*	*Description*
initialization	N_{startup}	Number of initial records
search	$d_{\max}$	Max search distance
	$\theta_{\max}$	Max normal difference
insertion	$w_{\min}$	Minimum weight allowed
	$\bar{\delta}_{\max}$	Max average visibility difference between nearby records
removal	$\varepsilon_{\max}$	Max number of wrong visibility queries

Table 1: *The parameters controlling the visibility cache.*

and then manually adjust the threshold values only if needed.

It would be interesting to look into ways of automatically determining good starting values for the different parameters. We could perhaps use statistics gathered during the initial bootstrapping phase to estimate the scene complexity. Some of the work by Feixas et al. on analyzing scene complexity [9] could potentially be used for this purpose. This has been saved for future work.

5.5 Implementation

To create a visibility map, we divide the domain into $2^M \times 2^M$ pixels, and evaluate the visibility through ray tracing with one ray per pixel. The sample locations are stratified within the cells, and to improve the blue-noise characteristics of the sampling pattern, we perform a small number (10–30) of iterations of Lloyd-relaxation. We use an area-preserving mapping [4] of the sphere, which is based on the octahedral map.

Since V is a binary function, we have opted for a compact *bitwise* representation of the cache records. In total we need 2^{2M} bits to store an uncompressed visibility map. The bits are encoded using the Z-order (Morton-order) space-filling curve. The index, z, of a two-dimensional coordinate, (x,y), is found by bitwise interleaving the binary representations of its coordinates, $x = (x_{M-1} \dots x_0)_2$ and $y = (y_{M-1} \dots y_0)_2$ respectively, as follows:

$$z = (y_{M-1}x_{M-1} \dots y_1 x_1 y_0 x_0)_2. \tag{10}$$

Due to its locality-preserving behavior, this encoding implicitly provides a quadtree representation of the visibility. The bits representing a node are always consecutive, so the visibility can be found by simple bit shifts and logical operations, e.g., if a node contains all zeroes, it is fully occluded. At higher levels, we use a secondary hierarchy with 2-bit values indicating full/partial/no visibility. See Figure 11. Another advantage of using a bitwise representation is that we can very quickly find the mean difference, $\delta_{ij} = \sum |\tilde{V}_i - \tilde{V}_j| / 2^{2M}$, between two visibility

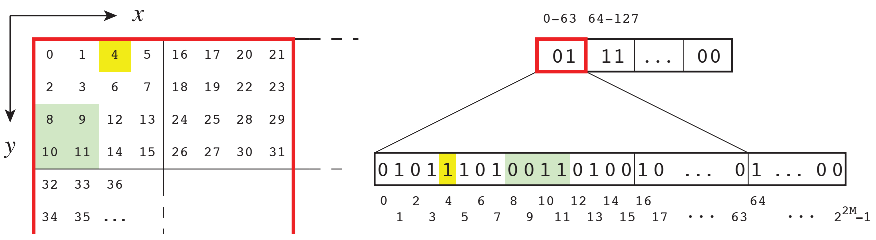


Figure 11: *We encode the binary visibility map as a two-level Z-order hierarchy, which gives an implicit quadtree representation.*

maps. We compute δ_{ij} as follows:

$$\delta_{ij} = \frac{\#\text{nonzero bits in } \tilde{V}_i \oplus \tilde{V}_j}{2^{2M}}, \qquad 0 \le \delta_{ij} \le 1, \tag{11}$$

i.e., we *xor* the binary visibility representations and count the number of nonzero bits in the result. This can be done very efficiently using SIMD-optimized code.

To quickly find $J = \int L\tilde{B}\tilde{V}\,d\omega$ (see Equation 2), which serves as an approximation to the sought-after integral, we traverse the quadtree representation of $L\tilde{B}$ [4], starting at its root. For each node, we look at the visibility, and stop the recursion if the node is occluded. If it is fully visible, we add up the integral of $L\tilde{B}$ over the node, which is already computed in the importance sampling step, and if the node is only partially visible, we recursively traverse its four children.

6 Other Applications

The visibility cache is an adaptively and sparsely sampled representation of the 4D visibility field. Our prime application is Monte Carlo rendering. However, the same framework can be used in a number of other applications.

6.1 Ambient Occlusion

Ambient occlusion [34] is a widely used technique for adding realism to local shading models. The ambient occlusion term, A, is the integral of the visibility function over the hemisphere, taking solid angle into account, as follows:

$$A(\mathbf{x}) = \int_{\Omega} V(\mathbf{x},\omega)(n\cdot\omega)\,d\omega. \tag{12}$$

This can be evaluated using ray tracing, but for high-quality results without banding, up to 1000 rays/pixel are needed. By replacing V by our visibility approximation, $\tilde{V}$, obtained from the cache, we get a quick approximation of A. For example,

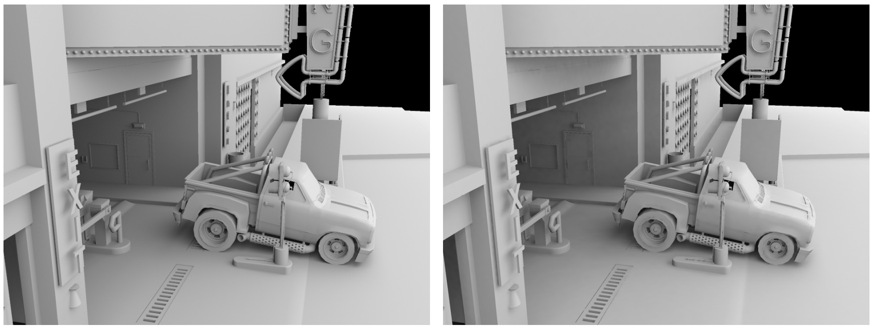


Figure 12: *The left image shows a ray traced ambient occlusion reference image using 4 samples/pixel and 1024 rays per shading point. For the right image, we have used our visibility cache for evaluating the ambient occlusion term, using on average 17.7 rays per shading point. The speedup is a factor 63× for this scene. However, some artifacts are visible, e.g., around the door, and the overall appearance is a bit softer.*

for full HD rendering (resolution 1920×1080) and a visibility cache with 20k records of resolution 32×32, the amortized cost is only 5 rays/pixel. Since the cache is adaptively refined, the method handles regions of difficult occlusion very well.

We compute the ambient occlusion term, A, during the creation of each new visibility record, and store the value in the cache record. For shading a pixel, we locate the m nearest records and compute a weighted average of their preintegrated A values, rather than first selecting a smaller set of records as before. To get a smoother solution, we use a Gaussian filter, whose width is set based on the projected pixel area. We also compute the distance, R_i, to the nearest intersection at each cache record in order to better detect small geometric features, similar to [26]. If $\max[w_i \cdot (1-d/R_i)] < w_{\text{min}}$, then a new record is inserted. Figure 12 shows the result for a typical scene containing about 259k triangles. A major strength of our approach is that the solution is noise-free, although a few artifacts due to the sparse sampling exist.

6.2 Lighting Design

The integral, J, over the control variate term (see Equation 2), is an approximation of the outgoing radiance based on the exact lighting and approximations of the reflectance and visibility, $\tilde{B}$ and $\tilde{V}$ respectively. By directly visualizing J, we get a very quick *preview* of the direct illumination. This can be useful for, e.g., fine-tuning the lighting in a scene before starting a production-quality rendering.

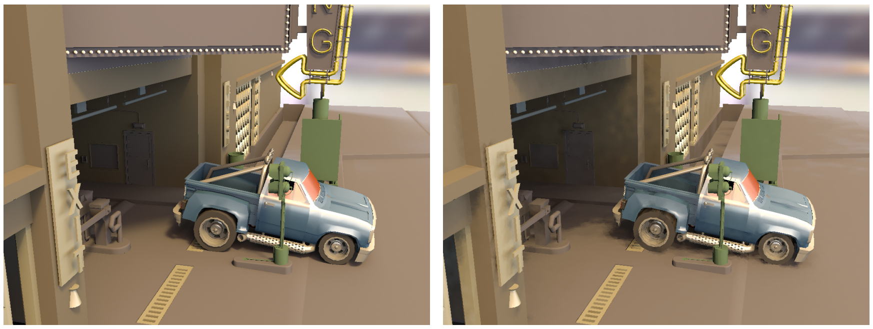

Reference Preview (12,500 cache records)

Figure 13: *Direct visualization of the control variate term gives a quick preview without doing any per-pixel sampling. After the BRDF quadtrees have been setup, the environment map can be replaced and/or rotated freely, providing an almost instant preview of the shading, including glossy effects.*

Figure 13 shows a preview image computed using 12,500 cache records. The output is fairly blotchy, but the quality is good enough to judge where shadows and highlights fall. More advanced interpolation strategies should improve the quality, and we plan to develop this idea further. The main execution cost currently lies in computing $\tilde{B}$ per pixel. A system for caching and interpolating $\tilde{B}$ (i.e., a shader cache), similar to our visibility cache, would be one way of decreasing the cost.

7 Results

We have implemented our algorithm for direct illumination on top of a recent technique for product importance sampling [4], which samples the product of a distant area light source (e.g., an environment map) and the local BRDF. All parts of their algorithm are carried out unchanged. The only difference is that, for each pixel, we perform a lookup in our visibility cache, and evaluate the obtained visibility approximation for each of the sampling directions. These visibility approximations are subtracted from the estimator, and finally the integral of the triple product, $J = \int L\tilde{B}\tilde{V}$, is computed and added to the result, as described by Equation 3. All results were generated on a Mac Pro with an Intel "Penryn" 45nm processor at 3.2GHz, using 2 cores.

The performance was evaluated on three scenes of different complexity; *garden1* and *dynamics* from Figure 4, and the *garage* scene in Figure 13. For most tests we have used a visibility map resolution of 32×32 pixels. In simple scenes where ray tracing is fast, a higher resolution of 64×64 pixels can be used, but the extra cost is usually not motivated by a large enough quality improvement. The settings used for our three test scenes are summarized in Table 2, and the rendering results are

Scene	**garage**	**dynamics**	**garden1**
#Records	14,300	6,200	17,100
Record size	32×32	64×64	32×32
Avg rays/pixel	3.81	6.61	4.56

Table 2: *The number of cache records and resolutions used for three of our test scenes. The last row shows the amortized cost of the visibility cache, measured as the average number of shadow rays/pixel at 1600×1200 pixels resolution.*

garage

#Samples	10	30	100	300
Time [4]	103.0	155.3	325.1	779.1
Time [our] (s)	124.9	178.3	346.8	800.9
Overhead (s)	21.9	23.0	21.7	21.8
Overhead (%)	21.3%	14.8%	6.7%	2.8%
Variance ratio	0.212	0.339	0.497	0.633

dynamics

#Samples	10	30	100	300
Time [4]	65.0	82.1	135.2	280.5
Time [our] (s)	85.0	102.6	157.9	303.4
Overhead (s)	20.0	20.5	22.7	22.9
Overhead (%)	30.8%	24.9%	16.8%	8.1%
Variance ratio	0.405	0.533	0.641	0.734

garden1

#Samples	10	30	100	300
Time [4]	229.1	451.8	1211.7	3220.6
Time [our] (s)	309.2	534.7	1307.1	3306.8
Overhead (s)	80.1	82.9	95.4	86.2
Overhead (%)	34.9%	18.3%	7.9%	2.7%
Variance ratio	0.775	0.997	1.190	1.244

Table 3: *Statistics for three of our scenes rendered at 1600×1200 pixels resolution.*

presented in Table 3. We present the variance reduction as the ratio of variance in the images rendered with our algorithm to the ones rendered using only importance sampling [4].

For the *garage* scene there is a significant variance reduction, ranging from almost 5× at 10 samples/pixel to 37% at 300 samples/pixel. The lighting in this scene is representable for scenes where ordinary product importance sampling fails to give good results. The majority of the light is coming from a few large, bright light sources, which are occluded by the building. Hence, most rays are occluded, and

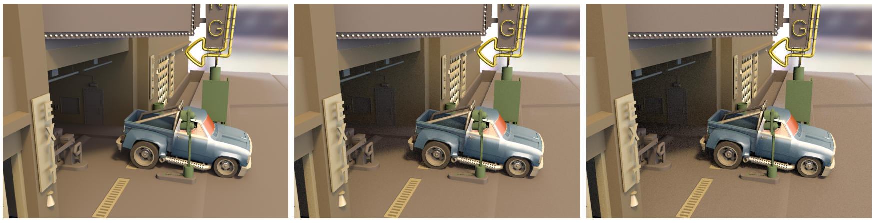


Figure 14: *Equal-time comparison between our algorithm and that of Clarberg et al. [4], using 30 and 39 samples/pixel respectively. The rendering time is 178 s at resolution 1600×1200, and the variance is reduced by 51.4%, i.e., to less than half.*

there is strong noise in the shadow regions. This is similar to what happens in Figure 1. In these types of scenes, our algorithm gives a large improvement at a modest cost. Figure 14 and 15 show a comparison between our method and that of Clarberg et al. [4]. At equal rendering time (178 s), the variance is reduced to less than half (variance ratio 0.489).

The *dynamics* scene is a hard case as it has very little occlusion. The control variate term gives a noise reduction also in unoccluded regions, but we have found the effect to be much stronger in shadows. However, our algorithm still gives a 37–60% reduction of variance, at a rendering time overhead of only 8% to 31%. There is a net win, but the improvement is not as large as we had hoped for.

Finally, the *garden1* scene presents a worst-case scenario with near-random visibility. Our analysis in Section 4 shows that there is a rather weak correlation in the visibility function. In order to exploit this, a large number of cache records would be needed. Using a reasonable number of records (17,100), we achieve only a very modest variance reduction at 10 and 30 samples/pixel. At the higher sampling rates, there is actually an *increase* in variance. This may seem counter-intuitive, since Equation 5 states that the variance can only decrease with control variates. However, that is under the assumption that the optimal value of α is known (Section 3), which is not the case.

The memory overhead of our visibility cache is very modest. Each cache record of resolution 32×32 occupies 172 bytes (including additional book-keeping data), which means a cache with 14,300 records (as we used for the garage scene) uses 2.4 MB memory. This fits well within the L2 cache on modern CPUs. In general, we have found the algorithm to complement existing methods for product sampling very well. A strong feature of our algorithm is that it provides a more robust solution than product sampling alone, as illustrated by Figure 1. The largest quality improvement is achieved in scenes with heavy occlusion, and especially in scenes with bright light sources that are occluded. Both these cases are difficult for product importance sampling. For more results, we refer to the supplemental video.

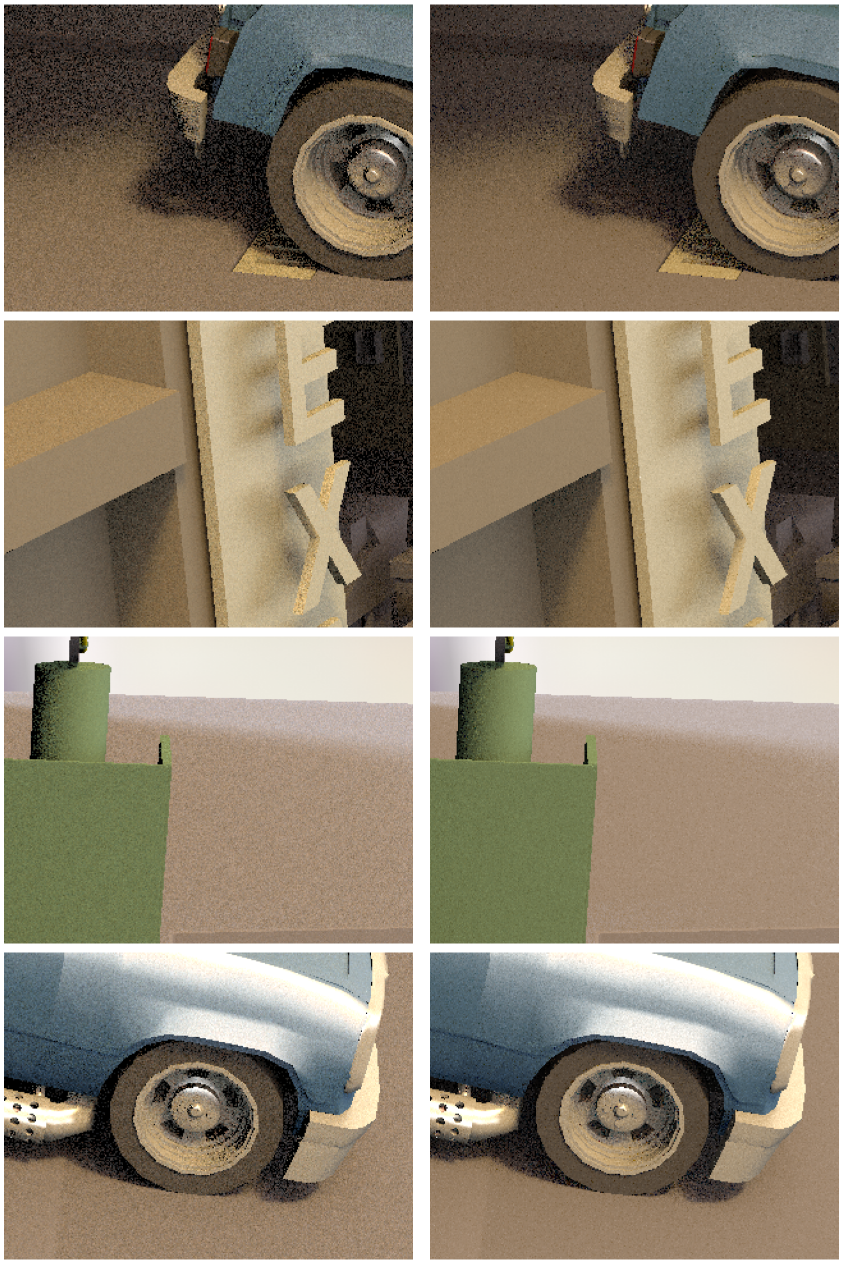


Figure 15: *Equal-time comparison with crops from the full resolution images in Figure 14.*

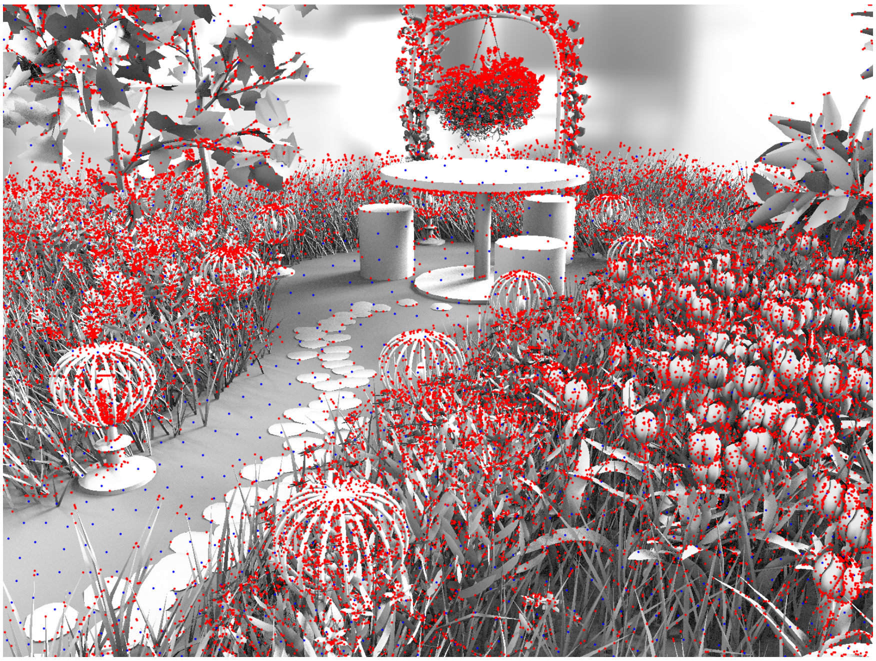


Figure 16: *The distribution of cache records for the garden1 scene, which presents a difficult case for our algorithm.*

8 Limitations and Discussion

We have restricted the analysis and algorithm to *binary* visibility. This makes an efficient bitwise implementation possible, but it also implies that only non-local lighting can be used, i.e., light sources that lie outside the convex hull of the scene. Our algorithm is, in theory, not limited to only distant lighting, but it makes most sense when combined with recent methods for importance sampling under environment map illumination [6, 4]. These algorithms exploit the fact that the lighting, L, does not change per pixel.

By storing the distance to the nearest occluder rather than just a binary value, we could adapt our method to handle local lighting. This would require substantial changes to the implementation, and the size of the visibility records would grow considerably, making its usefulness questionable. It would, however, be interesting to perform a similar analysis as in Section 4 to the case of local visibility. For binary visibility, nearby points and points with similar normals have highly correlated visibility. We expect the same to be true for local visibility, but new questions arise, e.g., how occluder distance correlates with normal/positional differences. It would also be interesting to perform a statistical analysis of the other terms in the rendering equation. This could be useful for improving, e.g., indirect illumination.

Implementation-wise, there are a couple of issues we would like to address. The direct visualization of visibility in Section 6 reveals that the interpolation is far from perfect. With better weights, we believe most of the artifacts in the ambient occlusion and lighting design examples can be removed. This would also further reduce the noise in MC rendering. An approach similar to irradiance gradients [32] should be possible. There are also a number of optimization to do, e.g., caching the visibility difference between nearby records instead of recomputing them for each lookup, using SIMD in the integration of the triple product, and so on.

9 Conclusion

The contribution of this paper is twofold. First, we study the statistical properties of the visibility function. Our insights here can be useful when designing algorithms taking visibility into account. We believe this is the next logical step in photo-realistic rendering, as many existing algorithms only consider the product of lighting and reflectance.

Second, we propose to use control variates to incorporate a visibility approximation in MC rendering. The key idea is to evaluate the difference between the exact function and the estimation from our visibility cache. The method is attractive in that it can exploit both spatial and temporal coherence, without introducing bias. We believe the same concept can be applied to other applications, e.g., radiance caching.

Acknowledgements

The garage scene was modeled by Christophe Desse and Matthew Thain, and the dynamics scene was created by Jacob Munkberg. We would also like to thank all the anonymous reviewers. This work was supported by the Swedish Foundation for Strategic Research and by Intel Corporation.

Bibliography

[1] AGRAWALA, M., RAMAMOORTHI, R., HEIRICH, A., AND MOLL, L. Efficient Image-Based Methods for Rendering Soft Shadows. In *Proceedings of ACM SIGGRAPH 2000* (2000), pp. 375–384.

[2] BEN-ARTZI, A., RAMAMOORTHI, R., AND AGRAWALA, M. Efficient Shadows from Sampled Environment Maps. *Journal of Graphics Tools, 11*, 1 (2006), 13–36.

[3] BURKE, D., GHOSH, A., AND HEIDRICH, W. Bidirectional Importance Sampling for Direct Illumination. In *Eurographics Symposium on Rendering* (2005), pp. 147–156.

[4] CLARBERG, P., AND AKENINE-MÖLLER, T. Practical Product Importance Sampling for Direct Illumination. *Computer Graphics Forum (Proceedings of Eurographics 2008), 27*, 2 (2008), 681–690.

[5] CLARBERG, P., JAROSZ, W., AKENINE-MÖLLER, T., AND JENSEN, H. W. Wavelet Importance Sampling: Efficiently Evaluating Products of Complex Functions. *ACM Transactions on Graphics, 24*, 3 (2005), 1166–1175.

[6] CLINE, D., EGBERT, P. K., TALBOT, J. F., AND CARDON, D. L. Two Stage Importance Sampling for Direct Lighting. In *Eurographics Symposium on Rendering* (2006), pp. 103–113.

[7] DONIKIAN, M., WALTER, B., BALA, K., FERNANDEZ, S., AND GREENBERG, D. P. Accurate Direct Illumination Using Iterative Adaptive Sampling. *IEEE Transactions on Visualization and Computer Graphics, 12*, 3 (2006), 353–364.

[8] FAN, S., CHENNEY, S., HU, B., TSUI, K.-W., AND LAI, Y.-C. Optimizing Control Variate Estimators for Rendering. *Computer Graphics Forum (Proceedings of Eurographics 2006), 25*, 3 (2006), 351–357.

[9] FEIXAS, M., DEL ACEBO, E., BEKAERT, P., AND SBERT, M. An Information Theory Framework for the Analysis of Scene Complexity. *Computer Graphics Forum (Proceedings of Eurographics 1999), 18*, 3 (1999), 95–106.

[10] FERNANDEZ, S., BALA, K., AND GREENBERG, D. P. Local Illumination Environments for Direct Lighting Acceleration. In *Eurographics Workshop on Rendering* (2002), pp. 7–14.

[11] GAUTRON, P., BOUATOUCH, K., AND PATTANAIK, S. Temporal Radiance Caching. *IEEE Transactions on Visualization and Computer Graphics, 13*, 5 (2007), 891–901.

[12] GHOSH, A., AND HEIDRICH, W. Correlated Visibility Sampling for Direct Illumination. *The Visual Computer, 22*, 9 (2006), 693–701.

[13] GHOSH, A., AND HEIDRICH, W. Sequential Sampling for Dynamic Environment Map Illumination. In *Eurographics Symposium on Rendering* (2006), pp. 115–126.

[14] HART, D., DUTRÉ, P., AND GREENBERG, D. P. Direct Illumination with Lazy Visibility Evaluation. In *Proceedings of ACM SIGGRAPH 99* (1999), pp. 147–154.

[15] JENSEN, H. W. *Realistic Image Synthesis Using Photon Mapping*. A K Peters, 2001.

[16] KAJIYA, J. T. The Rendering Equation. *Computer Graphics (Proceedings of ACM SIGGRAPH 86), 20*, 4 (1986), 143–150.

[17] KALOS, M. H., AND WHITLOCK, P. A. *Monte Carlo Methods*. John Wiley & Sons, 1986.

[18] KŘIVÁNEK, J., BOUATOUCH, K., PATTANAIK, S. N., AND ŽÁRA, J. Making Radiance and Irradiance Caching Practical: Adaptive Caching and Neighbor Clamping. In *Eurographics Symposium on Rendering* (2006), pp. 127–138.

[19] 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.

[20] LAFORTUNE, E. P., AND WILLEMS, Y. D. The Ambient Term as a Variance Reducing Technique for Monte Carlo Ray Tracing. In *Eurographics Workshop on Rendering* (1994), pp. 168–176.

[21] LAFORTUNE, E. P., AND WILLEMS, Y. D. A 5D Tree to Reduce the Variance of Monte Carlo Ray Tracing. In *Eurographics Workshop on Rendering* (1995), pp. 11–20.

[22] MEYER, M., AND ANDERSON, J. Statistical Acceleration for Animated Global Illumination. *ACM Transactions on Graphics, 22*, 3 (2006), 1075–1080.

[23] SHIRLEY, P., WANG, C., AND ZIMMERMAN, K. Monte Carlo Techniques for Direct Lighting Calculations. *ACM Transactions on Graphics, 15*, 1 (1996), 1–36.

[24] SZÉCSI, L., SBERT, M., AND SZIRMAY-KALOS, L. Combined Correlated and Importance Sampling in Direct Light Source Computation and Environment Mapping. *Computer Graphics Forum (Proceedings of Eurographics 2004), 23*, 3 (2004), 585–594.

[25] SZIRMAY-KALOS, L., CSONKA, F., AND ANTAL, G. Global Illumination as a Combination of Continuous Random Walk and Finite-Element Based Iteration. *Computer Graphics Forum (Proceedings of Eurographics 2001), 20*, 3 (2001), 288–298.

[26] TABELLION, E., AND LAMORLETTE, A. An Approximate Global Illumination System for Computer Generated Films. *ACM Transactions on Graphics, 23*, 3 (2004), 469–476.

[27] TALBOT, J., CLINE, D., AND EGBERT, P. Importance Resampling for Global Illumination. In *Eurographics Symposium on Rendering* (2005), pp. 139–146.

[28] VEACH, E., AND GUIBAS, L. J. Optimally Combining Sampling Techniques for Monte Carlo Rendering. In *Proceedings of ACM SIGGRAPH 95* (1995), pp. 419–428.

[29] WALTER, B., ARBREE, A., BALA, K., AND GREENBERG, D. P. Multidimensional Lightcuts. *ACM Transactions on Graphics, 25*, 3 (2006), 1081–1088.

[30] WALTER, B., FERNANDEZ, S., ARBREE, A., BALA, K., DONIKIAN, M., AND GREENBERG, D. P. Lightcuts: A Scalable Approach to Illumination. *ACM Transactions on Graphics, 24*, 3 (2005), 1098–1107.

[31] WARD, G. Adaptive Shadow Testing for Ray Tracing. In *Eurographics Workshop on Rendering* (1991), pp. 11–20.

[32] WARD, G. J., AND HECKBERT, P. Irradiance Gradients. In *Eurographics Workshop on Rendering* (1992), pp. 85–98.

[33] WARD, G. J., RUBINSTEIN, F. M., AND CLEAR, R. D. A Ray Tracing Solution for Diffuse Interreflection. *Computer Graphics (Proceedings of ACM SIGGRAPH 88), 22*, 4 (1988), 85–92.

[34] ZHUKOV, S., IONES, A., AND KRONIN, G. An Ambient Light Illumination Model. In *Eurographics Workshop on Rendering* (1998), pp. 45–55.

Paper IV

Efficient Product Sampling using Hierarchical Thresholding

Fabrice Rousselle[*][†] Petrik Clarberg[‡] Luc Leblanc[†]
Victor Ostromoukhov[†] Pierre Poulin[†]

[*]Ecole Polytechnique Fédérale de Lausanne
[†]University of Montreal
[‡]Lund University

ABSTRACT

We present an efficient method for importance sampling the product of multiple functions. Our algorithm computes a quick approximation of the product on-the-fly, based on hierarchical representations of the local maxima and averages of the individual terms. Samples are generated by exploiting the hierarchical properties of many low-discrepancy sequences, and thresholded against the estimated product. We evaluate direct illumination by sampling the triple product of environment map lighting, surface reflectance, and a visibility function estimated per pixel. Our results show considerable noise reduction compared to existing state-of-the-art methods using only the product of lighting and BRDF.

The Visual Computer (Proceedings of CGI 2008), 24(7-9):465–474, 2008.

1 Introduction

Monte Carlo methods are widely used in photo-realistic rendering, but many samples are needed for noise-free results. Importance sampling is a popular way to improve the performance by concentrating the sampling efforts to important regions. Ideally, the sampling density should be proportional to the function itself, but this is hard to achieve in practice. In this paper, we focus on integrating the direct illumination under environment map lighting. The problem involves a product of the lighting, surface reflectance, and local visibility. This product has to be computed on-the-fly for each pixel, as precomputation is infeasible due to the large amounts of data.

We store hierarchical representations of the local *maxima* and *averages* of the involved functions. For any interval, the product of the functions' individual maxima is always a conservative estimate of the product's local maximum. This can be used for rejection sampling, but many samples would be rejected in regions where the maximum is overly conservative. Instead, we compute an approximation of the product by hierarchically multiplying the local averages. The estimation is then refined in regions of potential high contribution, indicated by the local maximum. Samples are generated by thresholding against this approximated product,

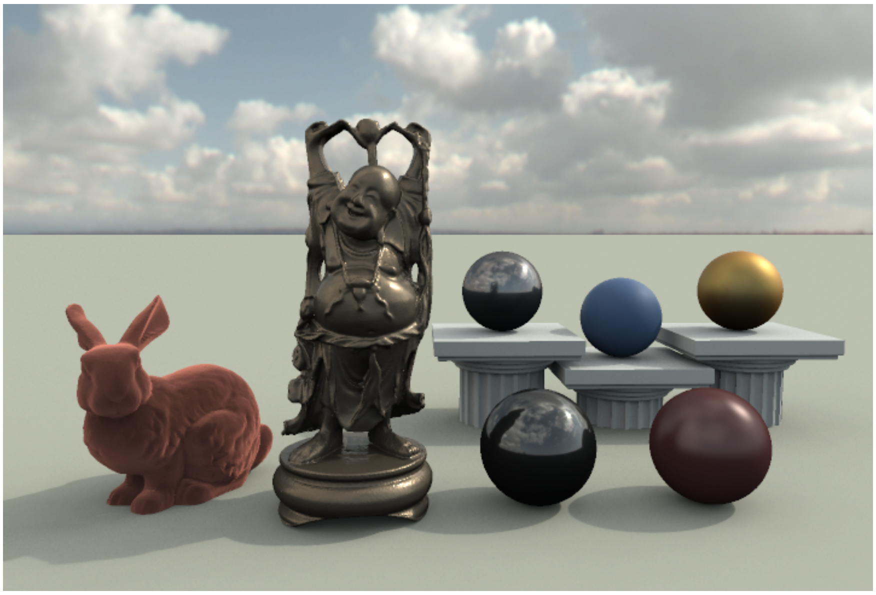


Figure 1: *Our algorithm can sample the product of multiple functions exhibiting a very wide range of frequencies. This image results from sampling the* triple *product of environment map lighting, surface reflectance, and estimated visibility.*

exploiting the hierarchical properties of many low-discrepancy sequences.

Unlike many previous methods, we aim for quickly generating samples approximately following the product distribution. In terms of overall performance, this is better than going through great effort to create a small set of near optimal samples. This is particularly true today, when ray tracing has reached interactive speeds. The complexity of our product approximation grows only linearly with the number of terms. Hence, it is possible to use more than two functions at a small extra cost. As a proof of concept, we conservatively approximate the visibility per pixel, and directly sample the *triple* product of lighting, BRDF, and visibility.

2 Previous Work

In this paper, we concentrate on computing the direct illumination using Monte Carlo integration. For a general overview of Monte Carlo methods, we refer to [11, 7]. In this context, the rendering equation [12] involves an integral over the product of lighting, material reflectance, and visibility. These are all potentially high-frequency, which makes it expensive to compute their product. In addition, the exact visibility is unknown, and must be locally estimated.

Many techniques exist for importance sampling only one of the three functions. Numerical BRDF models can often be analytically inverted, e.g., [2, 25], and measured materials can be efficiently sampled [15, 16]. Another example is environment map sampling [1, 14, 19]. It is also possible to draw samples from a weighted combination of multiple functions [23]. However, none of these methods take the product of the functions into account.

Talbot et al. [22] suggest *importance resampling*, where an initial set of samples is first drawn from a simpler distribution. Then, by giving the samples appropriate weights and resampling the initial set, they obtain samples approximately following the product distribution. Burke et al. [3] generate a large set of samples according to one of the product's individual terms, and use either rejection sampling or resampling to pick out the most important ones based on the remaining terms.

Other work has focused on explicitly estimating and sampling the product. Clarberg et al. [4] precompute wavelet representations of the lighting and materials. These are multiplied on-the-fly, and uniform points are *warped* into the desired distribution. Cline et al. [5] avoid the precomputation by using a summed-area table for the light source, which is hierarchically divided into smaller regions based on peaks in the BRDF. A major advantage is that spatially varying materials are supported.

We compute an approximation of the product by hierarchically multiplying precomputed representations of the *maxima* and *averages* of the individual terms. The approximation is adaptively refined, and samples are placed by hierarchical thresholding of low-discrepancy sequences, similar to [19]. Quasi-random numbers [17] are crucial for reducing the variance, and have a long history in computer graphics [13]. Our simple approach has a number of advantages. First, it gives a very fast algorithm. Second, our method allows inexpensive on-the-fly rotations of the involved functions (e.g., the BRDF), which means we avoid storing redundant pre-rotated data, and thus reduce memory requirements. Third, we can include additional terms in the product at a small cost. This opens up for novel sampling strategies.

As a proof of concept, we include a third importance function, representing an estimation of the visibility. We use a conservative approximation of the geometry with inner spheres [24], and build a low resolution visibility map per pixel. By including a visibility term, we effectively avoid sampling in directions guaranteed to be occluded. Very few other techniques exist, which exploit visibility to reduce the variance. Ghosh et al. [9] propose a two-pass method. First, they apply bidirectional importance sampling [3] to compute an initial estimate and to identify partially occluded pixels. Then, the noise is reduced by redistributing the variance from nearby pixels using Metropolis sampling.

3 Approximate Product Importance Sampling

Our sampling method is based on *hierarchical thresholding* of candidate points against an estimate of the function to be sampled. The idea is to gradually fill the sampling domain, and then perform a rejection test. Thresholding, or *rejection sampling* (see Figure 2), is a classic Monte Carlo technique for sampling an arbitrary function. First, we extend the method to efficiently handle the product of multiple functions, by using a conservative estimate of the product's maximum. Then, we introduce a fast approximation of the product to avoid a large number of slow evaluations of the individual functions. We use 1D examples throughout for clarity, but our method generalizes to any dimension.

3.1 Hierarchical Sample Generation

We generate candidate samples using low-discrepancy sequences that can be hierarchically constructed. By this we mean any sequence where samples can be iteratively added, while being uniformly distributed. One example is the van der Corput (VDC) sequence [17, 21]. Using a VDC sequence in base b, a sample's position, X_i, is defined as the *radical inverse* of its index i. Any positive integer i in base b can be expressed as a sequence of digits $d_m \dots d_2 d_1$ uniquely defined by $i = \sum_{j=1}^{m} d_j b^{j-1}$. The radical inverse is obtained by reflecting these digits about the decimal point. For instance, if $b = 2$, the radical inverse of $4 = (100)_2$ is $(0.001)_2 = 0.125$.

This sequence can be constructed recursively, in which case each subdivision multiplies the number of samples by b. We set a maximum level of subdivision L, which will generate $N = b^L$ samples, and define a sample's *threshold* value as

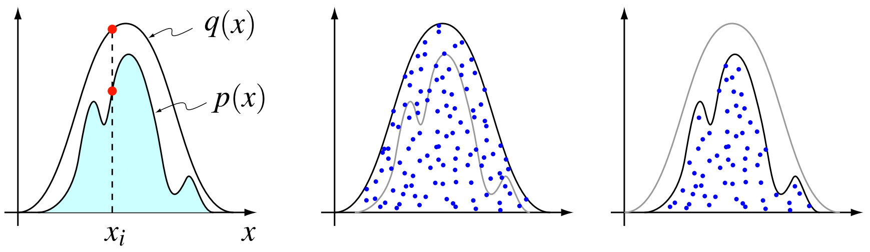


Figure 2: *In rejection sampling, samples are first drawn from a simpler* envelope *distribution, $q(x)$, and then randomly thresholded against the importance function $p(x)$. This can be illustrated as filling the space under $q(x)$ with random points, and then rejecting all points above $p(x)$. The probability of accepting a sample, x_i, is equal to $p(x_i)/q(x_i)$.*

$Y_i = i/N$. Now, for $f(x)$, such that $0 \leq f(x) < 1$, a sample is rejected if $f(X_i) < Y_i$. This process is illustrated in Figure 3. Note that our definition of threshold values assures that, as samples are generated, their threshold values are strictly increasing. This is a key point that will be exploited in the following section.

3.2 Sampling using Max-trees

The efficiency of rejection sampling relies on how well the envelope function approximates the desired importance function. A bad fit means more tested samples, many of which will be discarded. Consider the case of sampling a product with multiple terms:

$$f(x) = \prod_i f_i(x). \tag{1}$$

The construction of a good envelope function for Equation 1 can be a difficult problem. For example, in the case of direct illumination, the individual terms represent lighting, BRDF, and visibility.

We propose to precompute a hierarchical representation of the *maximum* for each individual term, which we call the *max-tree*. The max-tree is created by recursively subdividing the domain, storing the maximum over each child in a tree structure. For discrete functions, e.g., environment maps, finding the maximum over a region is straightforward. For analytical functions, such as many BRDFs, it is in some cases possible to derive an expression for the maximum. However, to remain general, we rely on point sampling of the function, and use extensive oversampling to reduce the risk of missing peaks. This assumes the function is reasonably smooth, but it gives no guarantee of finding the true maximum.

By multiplying together the individual maxima, we get an upper bound for the product. More formally, for any region $[a, b]$ in the function domain, the following

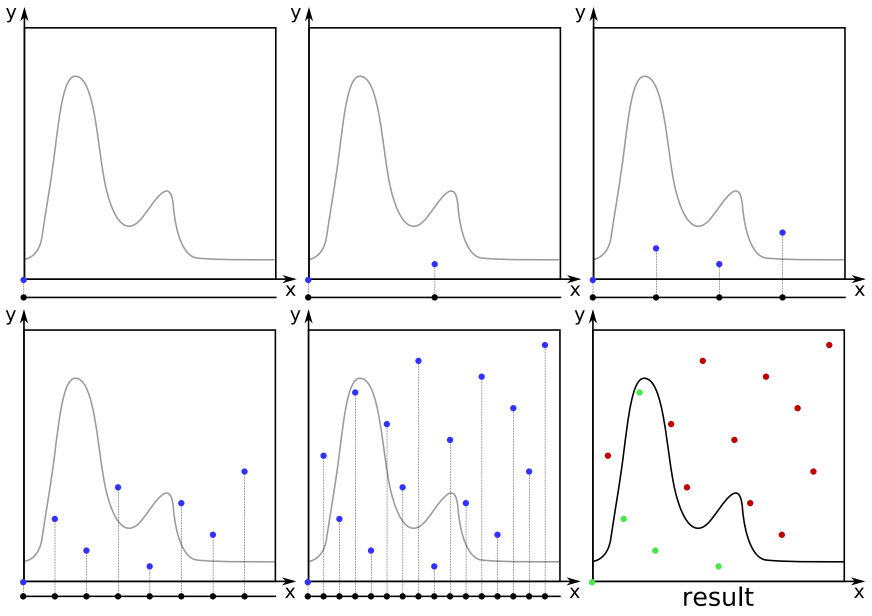


Figure 3: *Sample generation gradually filling the space using a van der Corput sequence. Note that $Y_j > Y_i$ for $j > i$.*

holds true:

$$\max_{x \in [a,b]} f(x) \leq \prod_i \left(\max_{x \in [a,b]} f_i(x) \right). \tag{2}$$

Note that this upper bound gives a tighter fit at finer subdivisions. We generate samples by recursively subdividing the domain, while evaluating Equation 2 at each level. Since samples are hierarchically generated and have ever increasing threshold values, we can safely stop the recursion as soon as a sample threshold value is larger than the local maximum, as illustrated in Figure 4. This effectively limits the number of generated samples.

3.3 Product Approximation

To speed up the sampling, we compute an *approximation* of the product, against which potential samples are thresholded. Hence, we avoid the expensive evaluation of the involved functions (e.g., environment map and BRDF) for each candidate point. On the negative side, we get samples only approximately following the target distribution, which increases the variance. However, in our application, faster sampling more than enough makes up for this, in terms of overall quality *vs* time.

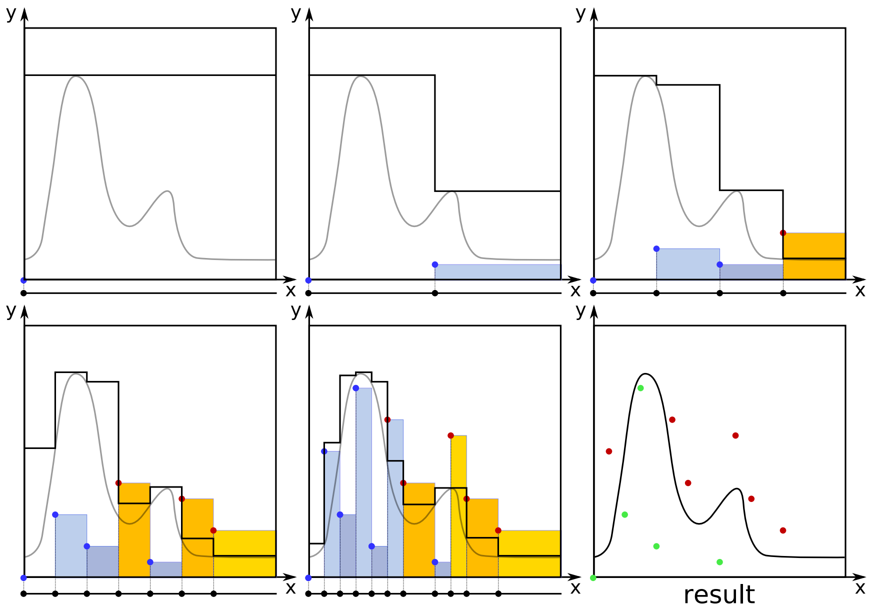


Figure 4: *Using the local maximum (solid black line), we can prune branches where all subsequent samples will be rejected (orange boxes), and thereby reduce the rejection rate.*

Previous work on importance sampling for direct illumination has used product approximations based on, e.g., wavelets [4] and summed-area tables [5]. We take a simpler approach, and multiply the local *averages* of the individual terms. Looking at an interval $[a,b]$, we use:

$$\frac{1}{b-a}\int_a^b \prod_i f_i(x)dx \approx \prod_i \left(\frac{1}{b-a}\int_a^b f_i(x)dx\right), \tag{3}$$

where the individual local averages, $\int_a^b f_i(x)dx/(b-a)$, are precomputed and stored in an *average-tree*, similar to our *max-tree*. We denote the piecewise constant importance function obtained this way by $h(x)$.

Equation 3 is clearly a crude approximation to the product, especially at coarse levels in the hierarchy. However, it converges towards the correct result at finer subdivisions. Note that, by construction, we adaptively refine the approximation where the function maximum is large, thereby minimizing the approximation error in regions having a significant contribution to the integral. This is a key point of our method. A challenging example is shown in Figure 5.

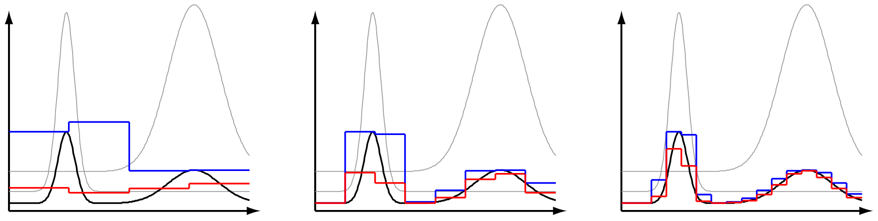


Figure 5: *Two non-overlapping peaks present a challenging case. Our product approximation (red) fails at coarse levels, as seen on the left. However, the large maximum (blue) around the left peak, will trigger further subdivision until the missing peaks are found, as shown on the right.*

3.4 Avoiding Bias

The deterministic nature of low-discrepancy sequences implies a fixed distribution of sample positions. To avoid bias, we use *scrambling* [20, 8] and randomly permute the assignment of sample positions when subdividing. The discretization of threshold values is another source of bias, which we address by adding a random offset in the range $[0, 1/N)$ to each threshold value. This is a well-known technique [11], which ensures that, on average, the correct number of samples is selected.

The use of a low-discrepancy sequence based on the radical inverse yields samples at fixed positions, aligned on a grid defined by the level of subdivision. As the samples only cover a subset of the domain, the solution will be biased. To address this fact, all results in this paper were generated using the VDC-sequence with jittering on sample positions. However, it is possible (but more costly) to achieve better blue noise properties by taking the local neighborhood into account using structural indices, as in, e.g., Polyomino-based sampling [18].

3.5 Sample Count

As mentioned in Section 3.1, the number of candidate samples is $N = b^L$, where b is the subdivision factor and L is the maximum level of subdivision. Since candidates are uniformly distributed, the average number of accepted samples $\bar{n}$ is equal to N scaled by the integral of the importance function, $H = \int h(x)dx$, as follows:

$$\bar{n} = H \times b^L. \tag{4}$$

To obtain n samples on average, we scale h by a factor c, which is given by:

$$c = \frac{n}{H \times b^L}. \tag{5}$$

It is essential that $c \cdot h(x)$ remains in $[0, 1)$ for our rejection test to be valid. This is ensured by increasing L up to the point where $c \cdot h(x) < 1$. Note that c varies per pixel and cannot be precomputed.

As $h(x)$ is defined through the sampling process, its integral H is initially unknown. In order to estimate the number of samples that will be generated, we approximate H by performing two subdivisions and then computing the average of $h(x)$ at that level. If the final number of samples widely differs from n, we refine c and repeat the process. In our application, using a 20% tolerance, the average number of sampling iterations rarely exceeds 1.3.

3.6 Unbiased Monte Carlo Integration

In importance sampling, the probability density function does, by definition, integrate to 1. We draw samples from the scaled product approximation, $c \cdot h(x)$, which must be divided by its integral, $c \cdot H$, in order to meet this criteria. Since $h(x)$ is piecewise constant, its integral can easily be computed by summing the contribution of all leaf nodes during the sampling process:

$$H = \frac{1}{b^L} \sum_{i=1}^{m} h_i \times b^{L-l_i}, \tag{6}$$

where h_i is the value of $h(x)$ over the i^{th} node, and $l_i \in [0, L]$ is the node's level in the hierarchy. The size of the i^{th} node with respect to the maximum level of subdivision, L, is equal to b^{L-l_i}. The resulting *unbiased* Monte Carlo estimator, $\langle F \rangle$, for the integral $F = \int f(x)dx$ is:

$$\langle F \rangle = \frac{H}{n} \sum_{i=1}^{n} \frac{f(x_i)}{h(x_i)}, \tag{7}$$

where H is computed using Equation 6. Note that the term H plays the same role as L_{ns} (i.e., "exitant radiance in the absence of shadows") used in [3, 4].

4 Application – Direct Illumination

The computation of direct illumination from distant HDR environment map lighting [6] is a problem that has generated considerable interest in recent years. The outgoing radiance is given by [12]:

$$L_o(\mathbf{x} \to \omega_o) = \int_{\Omega} L(\mathbf{x} \leftarrow \omega_i) \rho(\omega_i \leftrightarrow \omega_o) V(\omega_i) d\omega, \tag{8}$$

where the lighting (L), reflectance (ρ), and visibility (V), are integrated over the hemisphere. We define ρ as the BRDF weighted by the cosine of the incident angle, i.e., $\rho = f_r(\omega_i \leftrightarrow \omega_o)(\omega_i \cdot \mathbf{N})$, as commonly done.

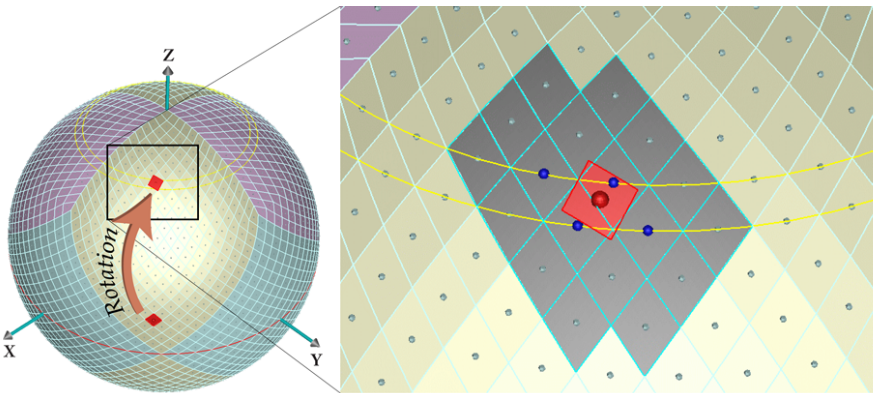


Figure 6: *After a rotation in the HEALPix mapping, the source quad usually overlaps a number of quads at the destination. The precomputed average at the red dot is linearly interpolated from the nearest neighbors, marked with blue dots. Similarly, we ensure that the local maximum is conservative by implicitly considering all quads marked dark gray.*

Our algorithm can be used to efficiently sample the product $L \cdot \rho$. We also show that an approximated visibility term, $\tilde{V}$, can be included. By sampling according to the triple product $L \cdot \rho \cdot \tilde{V}$, we further reduce noise in regions with large occlusion. Results with and without the visibility term are presented in Section 5.

4.1 HEALPix Mapping

All involved functions are defined over the (hemi)sphere, while our algorithm depends on hierarchical subdivision of the domain into quads. We use the HEALPix (Hierarchical Equal Area isoLatitude Pixelization) mapping [10], and divide the sphere into 12 faces (see Figure 6), as described by Gorski et al. [10]. Each face is a curvilinear quad, which can be recursively subdivided into 2×2 smaller quads of equal area. We apply our sampling scheme on each face separately.

The HEALPix mapping has a number of desirable properties: (1) hierarchical representation, (2) area preservation, and (3) low distortion. The preservation of area simplifies our implementation, as we do not have to compute form factors. Low distortion is important when rotating between different domains (see Section 4.3).

4.2 BRDF Representation

A general BRDF is a 4D function parameterized over incident and outgoing directions, (θ_i, ϕ_i) and (θ_o, ϕ_o) respectively. Isotropic BRDFs, currently implemented in our system, are reduced to 3D functions depending only on θ_i, θ_o, and $|\phi_i - \phi_o|$.

We store isotropic materials as 2D *slices*, i.e., one 2D reflectance map $(\theta_i, |\phi_i - \phi_o|)$ for each θ_o. Each slice is first encoded as a mipmap image, and then mapped to the HEALPix representation. Only the data for the upper hemisphere is stored.

All materials in the scene are resampled into this representation as a precomputation step. To avoid missing features, we use oversampling and assume the BRDF is reasonably smooth. This approach is taken by most algorithms using tabulated materials, and rarely presents a problem. The reflectance maps (as well as the environment map) are stored in RGB color, and the local maxima and averages are computed per channel. During sampling, we threshold against the luminance, Y, computed using the perceptual weighting: $Y = 0.299R + 0.587G + 0.114B$.

4.3 Rotations

In our application, the lighting is given by a 2D environment map in world space, while material reflectance is defined in the local surface frame. Hence, a rotation between the two domains must be performed. This can be precomputed as in [4], but the increased memory requirements would limit us to low-resolution representations. Instead, our algorithm was designed to support fast *on-the-fly* rotation of the importance functions.

The estimations of a quad's maximum and average (Equations 2 and 3) are both *local* operators, depending only on the values of the corresponding nodes in each term. Hence, we can simply rotate the coordinates used for locating a quad in the hierarchical representation. However, after rotation, a quad usually covers multiple quads at the destination. The low distortion in the HEALPix mapping helps reduce the overlap, but special care has to be taken to ensure that our estimation of the maximum remains conservative. We proceed as illustrated in Figure 6.

4.4 Visibility Approximation

One of the advantages of our method is that we can inexpensively include additional terms in the product. A natural extension is to use an estimated visibility map to steer samples away from occluded directions.

We use a visibility estimation inspired by [24]. Each object in the scene is approximated by a set of inner spheres. These spheres are aggregated into a hierarchy, but only leaves act as occluders (to preserve inner conservativity). To create a visibility map, we traverse the sphere hierarchy and the HEALPix hierarchy in parallel. If the cone enclosing a quad of the HEALPix mapping is completely occluded by a leaf sphere, the quad is marked as occluded (zero visibility). Other quads are set to fully visible (one). Then, we propagate these values upwards and store their maxima and averages.

The cost of building the visibility approximation is independent of the number of samples. Hence, the amortized cost is smaller for high-quality rendering, where more samples are used. In our implementation, a visibility map is built per pixel, which is expensive with many occluders. Another approach is to re-use the visibility estimate over several pixels, or use adaptive updates. We have saved this for future work. Other occluder primitives can also be added. In addition to spheres, our implementation supports infinite planes.

The visibility term gives a large variance reduction where the information is accurate. In other parts, e.g, along shadow edges, the effect is smaller. To handle such regions, we have experimented with adaptive sampling. We increase the number of samples for each pixel where the number of occluded visibility rays is above a certain threshold, e.g., 50%, and repeat the sampling process.

4.5 Early Termination and Biased Evaluation

In our algorithm presented in Section 3, samples are always placed in leaf nodes at the maximum level of subdivision. This gives an importance function that is accurate, but at a higher cost. Here, we present two optional extensions for increasing the performance.

First, we propose to terminate the recursion as early as possible. For this, we identify branches with *at most* one sample. Our sample threshold values are, by definition, strictly increasing. Thus, when we reach a point where the *next* threshold value is larger than the local maximum, only the current sample can possibly be accepted. Instead of traversing the hierarchy up to the leaf level, we place the sample in the current node and terminate. This gives a faster algorithm, but at the expense of a small increase in variance.

Second, we propose a biased version of our algorithm. Instead of point sampling the exact L and ρ in evaluating the rendering equation, we use the local averages as sample values for samples placed in large nodes. To avoid visual artifacts, e.g., with highly specular materials, we combine unbiased and biased evaluations. This is best explained with an example. Consider the case where the BRDF and the environment map are precomputed up to levels 5 and 8 respectively. If a sample is placed in a node at level 7, we compute the exact BRDF value, but use the precomputed average of the environment over the node. The sample is then assigned the product of these values. Note that this approach, although biased, is *consistent*, i.e., it converges towards the correct solution.

We have found the combination of these two extensions to be extremely useful. When a sample is placed at lower levels in the hierarchy (large nodes), the average-tree approximately gives us the integral over the node, instead of a single point-sampled value. This significantly reduces the variance. In Section 5, we present results using both the unbiased and the biased versions.

5 Results

All results were obtained on a MacBook Pro with an Intel Core 2 Duo 2.4GHz (using 1 core), and all functions were stored uncompressed in quadtrees of different depths. At depth n, each of the HEALPix mapping's 12 faces contains $2^n \times 2^n$ quads. In all images, the environment map was stored at depth 8, and occupies 24 MB. For a 4k$\times$4k angular map, the setup time was 1.46 s.

Isotropic BRDFs are represented with 50 slices computed for outgoing angles uniformly distributed in $[0, \pi/2]$. The only user set parameter is the BRDF resolution. In practice, we use depth 5 for most materials, while diffuse materials (e.g., the ground in Figure 1) are stored at depth 4, and highly specular materials (e.g., the sphere in Figure 8) are stored at depth 6. The precomputation times for measured materials [16] are:

Depth n	Memory	Precomputation
4	2.55 MB	0.36 s
5	9.78 MB	1.35 s
6	38.45 MB	5.31 s

The estimated visibility function is computed at depth 4 and occupies 8 KB. Figure 7 illustrates a practical situation where a Buddha is approximated by 110 inner spheres. It should be noted that the cost of estimating visibility is independent of the number of samples, and the cost of including a third term in the product is marginal. For this scene, adding visibility increases the rendering time by about 14 seconds. The rendering times for the left image in Figure 7, at resolution 256×256, were:

Number of Samples	No Visibility	Non-adaptive Visibility	Adaptive Visibility
16	4.1 s	17.8 s	18.3 s
128	14.7 s	28.5 s	33.0 s
512	49.8 s	64.0 s	81.0 s

Figure 8 and 9 show a comparison against several recent techniques. Results with sampling of only the BRDF or the environment map are also included. The material is a normalized Phong with diffuse and specular lobes. The shininess coefficient is 5000 for the sphere and 10 for the plane. These values were chosen to illustrate a full range of frequencies. Strong light blocked by occlusion results in a high noise level with algorithms sampling only the product of lighting and BRDF. This effect is diminished by taking visibility into account.

This scene presents an extremely easy case for our visibility estimation, only increasing the rendering time by 11% (with 64 samples). This is not representative for the general case, but it shows, as a proof of concept, that including a visibility approximation can dramatically lower the noise. More sophisticated algorithms

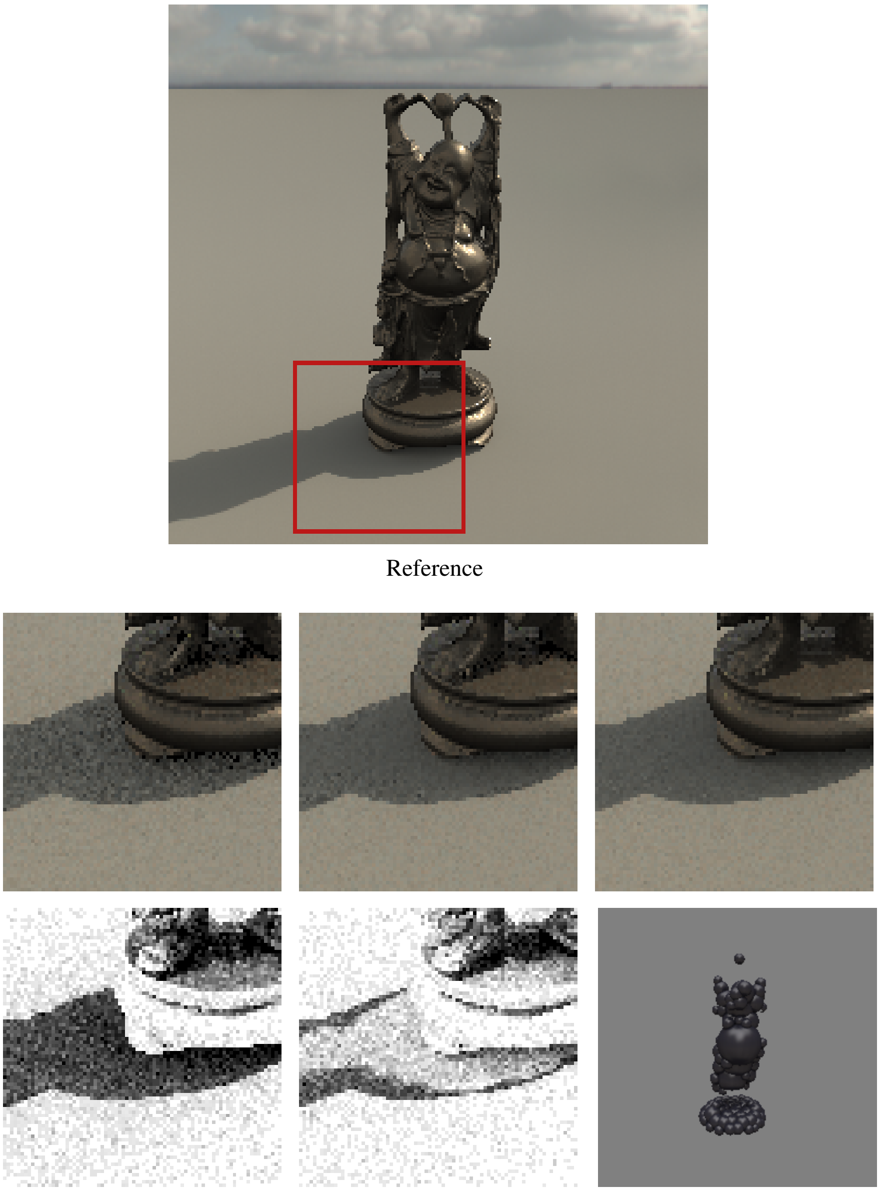


Figure 7: *Images illustrating the impact of adding visibility information and adaptive sampling, using 16 samples per pixel and biased rendering. The two gray scale images (bottom row) show the percentage of occluded rays, where black is 100% and white 0%. The bottom right image shows the sphere approximation used for the Buddha. The topmost image is a reference.*

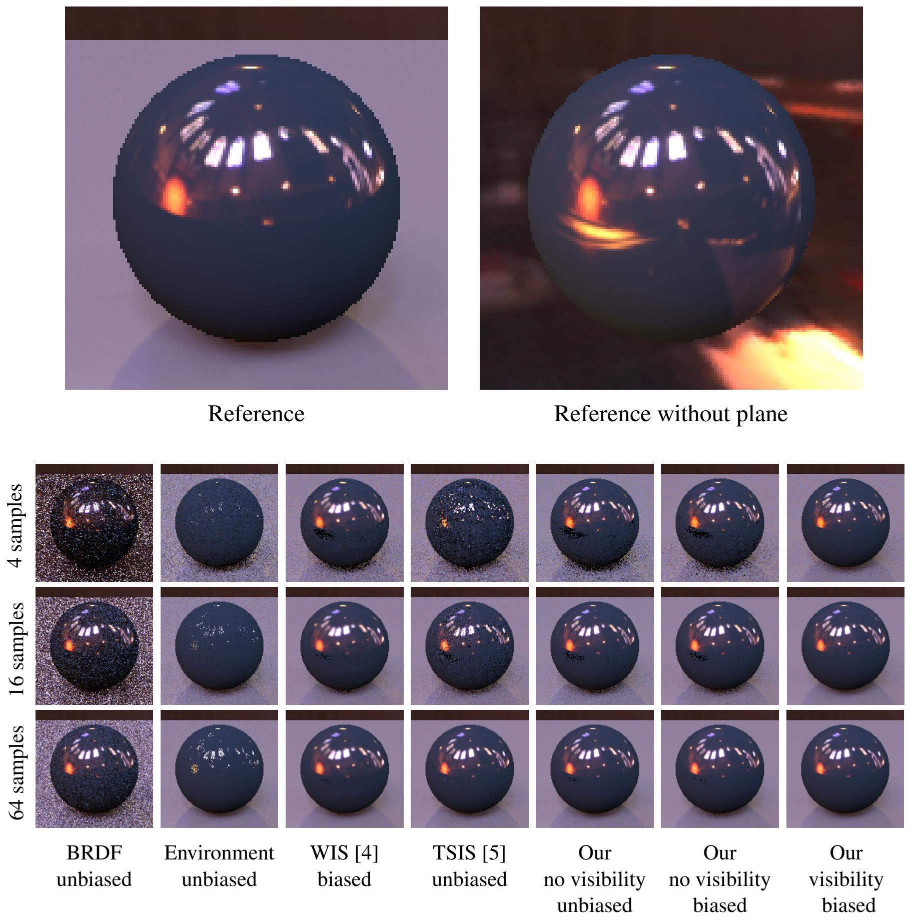


Figure 8: *Comparison with other recent methods. For WIS, we used wavelets of* 128^2 *resolution, 2–5% sparsity and Poisson points as input. For TSIS, we did not use adaptive sampling. BRDF sampling was done analytically, and environment map sampling with the unbiased version of our algorithm. For our algorithm, the BRDFs for the plane and the sphere were precomputed to HEALPix levels 5 and 6 respectively, and we did not use adaptive sampling. Ground truth is shown at the top. Visibility sampling effectively removes noise due to occlusion.*

for estimating the visibility term have been left for future work. The images also show that our biased extension gives strong noise reduction, while being consistently closer to the reference. The rendering times were: [1]

[1]Note that the timings are not directly comparable as the algorithms were implemented in different ray tracers. However, all systems show a performance similar to `pbrt` [21].

#Samples	4	16	64	256
HT unbiased	5.1 s	7.3 s	10.0 s	20.4 s
HT biased	2.3 s	3.2 s	4.8 s	9.9 s
HT+Vis biased	2.8 s	3.7 s	5.3 s	10.3 s
WIS [4]	6.4 s	6.7 s	7.8 s	11.3 s
Two-stage [5]	0.8 s	1.8 s	5.8 s	19.3 s

Figure 1 illustrates the robustness of our method and its applicability to a wide range of materials, ranging from diffuse to highly specular, in a lighting environment with an extreme dynamic range.

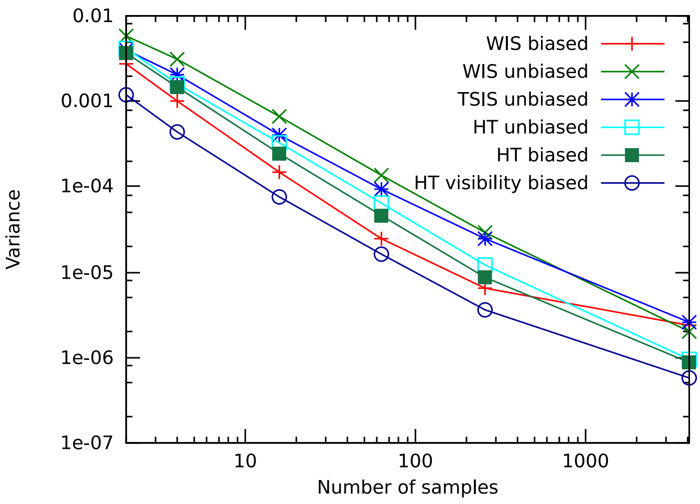


Figure 9: *Variance as a function of the number of samples for the methods compared in Figure 8. The variance was measured after tone-mapping.*

6 Discussion and Future Work

Our sampling scheme is based on hierarchical thresholding against an approximated importance function. The approximation is computed from hierarchical representations of the local maxima and averages, and enables several important features. First, we can sample products of multiple functions, including rotations between different domains. Second, many useful optimizations are possible, e.g., early termination and biased integration, which improve speed and reduce noise. Although our method requires the involved functions to be smooth and bounded, we have found it to be very robust.

For estimating the direct illumination, our results compare favorably to existing state-of-the-art methods. As a proof of concept, we include a visibility term estimated per pixel, and show that it is possible to significantly reduce the noise due to occlusion. Exploiting visibility information to speed up the computation of direct illumination is an interesting direction of research. We would also like to remove some of the limitations of our algorithm, most importantly the precomputation step. One possibility could be to compute the necessary data on-the-fly directly from analytical or factorized BRDFs. This would allow spatially varying materials, which is important in a number of applications.

Acknowledgements

We thank Nicolas Tittley for the inner spheres, as well as Tomas Akenine-Möller and Henrik Wann Jensen for their insightful comments. This work was supported by the Natural Sciences and Engineering Research Council of Canada, and the Swedish Foundation for Strategic Research. Lastly, we thank Paul Debevec who provided us with the environment maps.

Bibliography

[1] AGARWAL, S., RAMAMOORTHI, R., BELONGIE, S., AND JENSEN, H. W. Structured Importance Sampling of Environment Maps. *ACM Trans. Graphics, 22*, 3 (2003), 605–612.

[2] ASHIKHMIN, M., AND SHIRLEY, P. An Anisotropic Phong BRDF Model. *Journal of Graphics Tools, 5*, 2 (2000), 25–32.

[3] BURKE, D., GHOSH, A., AND HEIDRICH, W. Bidirectional Importance Sampling for Direct Illumination. In *Eurographics Symposium on Rendering* (2005), pp. 147–156.

[4] CLARBERG, P., JAROSZ, W., AKENINE-MÖLLER, T., AND JENSEN, H. W. Wavelet Importance Sampling: Efficiently Evaluating Products of Complex Functions. *ACM Trans. Graphics, 24*, 3 (2005), 1166–1175.

[5] CLINE, D., EGBERT, P. K., TALBOT, J. F., AND CARDON, D. L. Two Stage Importance Sampling for Direct Lighting. In *Eurographics Symposium on Rendering* (2006), pp. 103–113.

[6] DEBEVEC, P. Rendering Synthetic Objects into Real Scenes: Bridging Traditional and Image-Based Graphics with Global Illumination and High Dynamic Range Photography. In *Proc. ACM SIGGRAPH 98* (1998), pp. 189–198.

[7] DUTRÉ, P., BALA, K., AND BEKAERT, P. *Advanced Global Illumination*, 2nd ed. A K Peters, 2006.

[8] FRIEDEL, I., AND KELLER, A. Fast Generation of Randomized Low-Discrepancy Point Sets. In *Monte Carlo and Quasi-Monte Carlo Methods* (2000), pp. 257–273.

[9] GHOSH, A., AND HEIDRICH, W. Correlated Visibility Sampling for Direct Illumination. *The Visual Computer, 22*, 9 (2006), 693–701.

[10] GORSKI, K. M., HIVON, E., BANDAY, A. J., WANDELT, B. D., HANSEN, F. K., REINECKE, M., AND BARTELMANN, M. HEALPix – A Framework

for High Resolution Discretization, and Fast Analysis of Data Distributed on the Sphere. *The Astrophysical Journal, 622* (2005), 759–771.

[11] JENSEN, H. W., ARVO, J., DUTRÉ, P., KELLER, A., OWEN, A., PHARR, M., AND SHIRLEY, P. Monte Carlo Ray Tracing. In *ACM SIGGRAPH 2003 Course Notes* (2003).

[12] KAJIYA, J. T. The Rendering Equation. *Computer Graphics (Proc. ACM SIGGRAPH 86), 20*, 4 (1986), 143–150.

[13] KELLER, A. *Quasi-Monte Carlo Methods for Photorealisitic Image Synthesis*. PhD thesis, University of Kaiserslautern, 1998.

[14] KOLLIG, T., AND KELLER, A. Efficient Illumination by High Dynamic Range Images. In *Eurographics Symposium on Rendering* (2003), pp. 45–50.

[15] LAWRENCE, J., RUSINKIEWICZ, S., AND RAMAMOORTHI, R. Efficient BRDF Importance Sampling using a Factored Representation. *ACM Trans. Graphics, 23*, 3 (2004), 496–505.

[16] MATUSIK, W., PFISTER, H., BRAND, M., AND MCMILLAN, L. A Data-Driven Reflectance Model. *ACM Trans. Graphics, 22*, 3 (2003), 759–769.

[17] NIEDERREITER, H. *Random Number Generation and Quasi-Monte Carlo Methods*. Society for Industrial and Applied Mathematics, 1992.

[18] OSTROMOUKHOV, V. Sampling with Polyominoes. *ACM Trans. Graphics, 26*, 3 (2007), 78.

[19] OSTROMOUKHOV, V., DONOHUE, C., AND JODOIN, P.-M. Fast Hierarchical Importance Sampling with Blue Noise Properties. *ACM Trans. Graphics, 23*, 3 (2004), 488–495.

[20] OWEN, A. B. Randomly permuted (t,m,s)-nets and (t,s)-sequences. *Lecture Notes in Statistics 107* (1995), 299–317.

[21] PHARR, M., AND HUMPHREYS, G. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann, 2004.

[22] TALBOT, J., CLINE, D., AND EGBERT, P. Importance Resampling for Global Illumination. In *Eurographics Symposium on Rendering* (2005), pp. 139–146.

[23] VEACH, E., AND GUIBAS, L. J. Optimally Combining Sampling Techniques for Monte Carlo Rendering. In *Proc. ACM SIGGRAPH 95* (1995), pp. 419–428.

[24] WANG, R., ZHOU, K., SNYDER, J., LIU, X., BAO, H., PENG, Q., AND GUO, B. Variational Sphere Set Approximation for Solid Objects. *The Visual Computer, 22*, 9 (2006), 612–621.

[25] WARD, G. J. Measuring and Modeling Anisotropic Reflection. *Computer Graphics (Proc. ACM SIGGRAPH 92), 26*, 2 (1992), 265–272.

Paper V

Fast Equal-Area Mapping of the (Hemi)Sphere using SIMD

Petrik Clarberg

Lund University

ABSTRACT

We present a fast vectorized implementation of a transform that maps points in the unit square to the surface of the sphere, while preserving fractional area. The mapping uses the octahedral map combined with an equal-area parameterization, and has many desirable features such as low distortion, straightforward interpolation, and fast inverse and forward transforms. Our SIMD implementation completely avoids branching, and uses polynomial approximations for the trigonometric operations, along with other tricks. This results in up to 9 times speedup over a traditional scalar implementation. Source code is available online.

Journal of Graphics Tools, 13(3):53–68, 2008.

1 Introduction

Spherical and hemispherical functions are abundant in computer graphics. Examples include environment maps, BRDFs, visibility data, surface maps for spherically parameterized objects, and so on. To handle such data, we need a mapping from the (hemi)spherical domain to the plane. The best choice of mapping depends on the application. For example, the *cube map* is convenient because of its simplicity and hardware support, but it is not area-preserving (pixels near the corners represent a smaller solid angle).

In many cases, it is desirable to use an *equal-area* mapping, i.e., a mapping which preserves fractional area. In applications integrating functions over the (hemi)sphere, area-preservation significantly simplifies the implementation as we do not have to take the solid angle of each pixel into account. The prime example is the evaluation of the rendering equation [4], which involves an integration over the hemisphere. Another desirable property is *low distortion*, i.e., the aspect ratio of pixels on the sphere should be close to one. Otherwise, square pixels can be mapped to long, thin segments on the sphere, which causes aliasing and reduces the useful resolution. It is also desirable to have as few discontinuities as possible in order to simplify interpolation.

In this paper, we describe a fast implementation of a mapping with all these properties: equal-area, low distortion, and support for straightforward interpolation across edges. The mapping uses the octahedral map [5] combined with an area-preserving parameterization [6]. The described mapping has been successfully used for importance sampling purposes [1].

The current trend is microprocessors with many cores and wide data paths. By exploiting data parallelism using SIMD (*single instruction, multiple data*) vectorization, the performance of many applications can be greatly improved. We provide a SIMD implementation (using Intel SSE) of the described mapping that is up to 9$\times$ faster than a straightforward scalar implementation, and roughly 4$\times$ faster than an optimized scalar version. Most of this paper deals with the technical details of our implementation, such as avoiding branching, polynomial approximations of the trigonometric operations, etc. We believe this is of interest to a wide audience, as knowledge about how to write SIMD code is becoming increasingly important to fully utilize the enormous performance available in modern CPUs.

2 Equal-Area Mapping

2.1 Hemisphere

For mapping the hemisphere, we use the *concentric map* [6], which maps concentric squares to concentric circles on the hemisphere, while preserving fractional area. See Figure 1. For the first sector, i.e., where $\phi \in [-\frac{\pi}{4}, \frac{\pi}{4}]$, a point

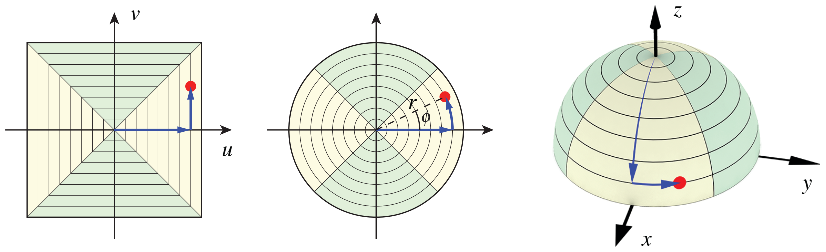


Figure 1: *The concentric map [6] transforms concentric squares in the plane to concentric circles on the hemisphere. The mapping preserves fractional area and has a relatively low distortion.*

(s,t) in the unit square $\mathscr{P} = [0,1]^2$ is transformed to a point on the hemisphere $\mathscr{H} = \{(x,y,z) \mid x^2+y^2+z^2=1, z \geq 0\}$ as follows:

$$(s,t) \rightarrow \begin{array}{l} u = 2s-1 \\ v = 2t-1 \end{array} \rightarrow \begin{array}{l} r = u \\ \phi = \frac{\pi}{4}\frac{v}{u} \end{array} \rightarrow \begin{array}{l} x = \cos\phi \cdot r\sqrt{2-r^2} \\ y = \sin\phi \cdot r\sqrt{2-r^2} \\ z = 1-r^2 \end{array} . \tag{1}$$

Similar transforms apply to the other sectors. The z-coordinate in the last step is equal to $z = 1-r^2 = \cos\theta$, where θ is the angle from the z-axis. Hence, $\sin\theta = \sqrt{1-\cos^2\theta} = r\sqrt{2-r^2}$, which explains the equations for x and y.

As noted by Shirley and Chiu, this simple mapping has a number of desirable properties. Most importantly, it preserves fractional area, which means a uniform point distribution in the square will map to a uniform distribution on the hemisphere. Second, the mapping preserves adjacency, i.e., nearby points in the square map to nearby points on the hemisphere. Last, the distortion is relatively well-behaved, which is important in order to reduce aliasing when sampling functions over the hemisphere.

2.2 Sphere

To get an equal-area mapping of the sphere, we combine the concentric map with the *octahedral map* [5], which is a clever way to "fold" a square over the sphere. The square is divided into eight triangles, where the four innermost triangles are mapped to the northern hemisphere, while the outer four are folded down to cover the southern hemisphere. Thus, each triangle maps to a quadrant in one of the two hemispheres, as illustrated in Figure 2. We rotate the concentric map by 45° and insert it into the inner quad of the octahedral map. To the best of our knowledge, this combination of the two mappings was first used in [1].

The transform from the unit square $\mathscr{P}$ to the sphere $\mathscr{S} = \{(x,y,z) \mid x^2+y^2+z^2=1\}$ is easy to derive. As before, the first step is to transform (s,t) to a point $(u,v) \in [-1,1]^2$. We start by considering the innermost triangle in the first quadrant, as

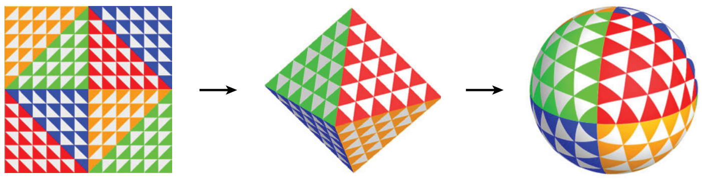


Figure 2: *The octahedral map [5] is obtained by folding a quad into an octahedron, which is projected onto the sphere using an arbitrary parameterization. Image courtesy of Emil Praun and Hugues Hoppe.*

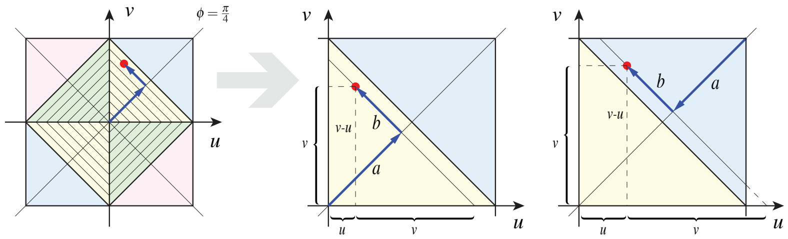


Figure 3: *The concentric map applied to the first quadrant of the octahedral map. The inner region (yellow) maps to the northern hemisphere, while the outer (blue) maps to the southern. The lengths a and b are found using simple trigonometry.*

shown in Figure 3. The lengths of a and b are $a = (u+v)/\sqrt{2}$ and $b = (v-u)/\sqrt{2}$, and the transform to the unit disk is given by:

$$\begin{aligned} r &= \sqrt{2}a = u+v, \\ \phi &= \frac{\pi}{4}\frac{b}{a} + \frac{\pi}{4} = \frac{\pi}{4}\left(\frac{v-u}{r}+1\right), \quad \text{with } 0 \le \phi \le \frac{\pi}{2}, \end{aligned} \tag{2}$$

where ϕ is measured from the positive u-axis (hence the addition of $\frac{\pi}{4}$). The outermost triangle maps to the southern hemisphere, and its parameterization is obtained by mirroring the innermost triangle about the diagonal. Here, $b = (v-u)/\sqrt{2}$ as before, but a differs slightly and is given as $a = (2-u-v)/\sqrt{2}$. The transform to the unit disk is:

$$\begin{aligned} r &= \sqrt{2}a = 2-u-v, \\ \phi &= \frac{\pi}{4}\frac{b}{a} + \frac{\pi}{4} = \frac{\pi}{4}\left(\frac{v-u}{r}+1\right), \quad \text{with } 0 \le \phi \le \frac{\pi}{2}. \end{aligned} \tag{3}$$

Note that the computation of ϕ is the same in both these cases, only r differs. The mapping from the disk to the sphere is the same as the last step of Equation 1, except that the z-component is negated, i.e., $z = -(1-r^2)$, if we are in the outer triangle. The final mapping is shown in Figure 4, and a proof of area-preservation is given in Appendix A.

So far, we have ignored the remaining quadrants. However, following a similar reasoning as above, the transforms are easily derived. A straightforward implementation, similar to Shirley and Chiu's version for the hemisphere, results in three levels of `if`-statements as there are 8 different cases (four quadrants, each divided into two triangles). The main execution cost lies in this branching, together with the trigonometric operations.

3 SIMD Implementation

We will now describe how each part of the algorithm can be efficiently written using the x86 streaming SIMD extensions (SSE). SSE code can be written directly using inline assembly language instructions, or using compiler intrinsics (which we use). Carefully hand-optimized assembly code can be faster, but is tedious and error-prone to write. Intrinsics, which is a C/C++ mapping of the assembly instructions, enable features like compiler optimizations and automatic register allocation, which make them easier to use. Intrinsics are also likely to be more forward compatible (e.g., if more registers are added, the code can, after recompilation, automatically benefit). With SSE, four floating-point values are processed in parallel, but future generation CPUs will likely have wider data paths. As we do not use any horizontal operations, it is trivial to adapt our implementation to such architectures.

3.1 The Square to Sphere Transform

3.1.1 Avoiding Branching

In SIMD code, branching is handled using conditional instructions which set a bit mask based on the outcome of some comparison. By executing both branches and selecting the correct result based on the mask using logical operations (`and`, `or`, `not`), the equivalent of a "parallel" `if`-statement is created.

In our case, there are 8 different ways to compute (r,ϕ), which makes this approach inefficient. Hence, we take an alternative route and map the problem to the first quadrant by taking the absolute values of u and v. A similar approach is used to find r without using any conditional instructions (we will come back to this in a moment). We use the following:

$$\phi' = \frac{\pi}{4}\left(\frac{|v|-|u|}{r}+1\right), \qquad \text{where } \phi' \in [0,\frac{\pi}{2}]. \tag{4}$$

In the last step of Equation 1, we need to compute the sine and cosine of ϕ, but we only have ϕ' to work with. Using standard trigonometric rules, we find the following expressions for $\sin\phi$ and $\cos\phi$ in each of the four quadrants:

quadrant	ϕ	$\sin\phi$	$\cos\phi$
1	ϕ'	$\sin\phi'$	$\cos\phi'$
2	$\pi-\phi'$	$\sin\phi'$	$-\cos\phi'$
3	$\phi'-\pi$	$-\sin\phi'$	$-\cos\phi'$
4	$-\phi'$	$-\sin\phi'$	$\cos\phi'$

Based on this, we realize that:

$$\begin{aligned}\sin\phi &= \operatorname{sign}(v)\cdot\sin\phi' \\ \cos\phi &= \operatorname{sign}(u)\cdot\cos\phi'\end{aligned} \quad \text{where } \operatorname{sign}(x) = \begin{cases} +1 & \text{if } x \geq 0, \\ -1 & \text{if } x < 0.\end{cases} \tag{5}$$

Fortunately, both the absolute-operator needed in Equation 4, and the sign-operator used in Equation 5, can be efficiently implemented using simple logic instructions. In the IEEE-754 standard for floating-point numbers, the most significant bit (MSB) represents the sign, where 0 is positive and 1 means negative. Hence, taking the absolute value is simply a matter of and'ing by the bit mask $01\ldots1_\text{b} = \text{0x7F}\ldots\text{F}$, and changing the sign can be done by xor'ing with the sign-bit: $10\ldots0_\text{b} = \text{0x80}\ldots0$. More formally:

$$\begin{aligned} |u| &= u \,\&\, \text{0x7FFFFFFF}, \\ \operatorname{sign}(u)\cdot x &= (u \,\&\, \text{0x80000000}) \oplus x, \end{aligned} \tag{6}$$

where $\oplus$ represents xor, and & means and. In practice, we use the andnps instruction (and a value with the logical inverse of another) to avoid loading two different constants from memory as $\text{0x80}\ldots0$ is the inverse of $\text{0x7F}\ldots\text{F}$.

We apply a similar reasoning to find r (Equations 2 and 3) and the z-coordinate (Equation 1) without branching. First, we compute the signed distance, d, along the diagonal in the first quadrant, so that $d=1$ at the origin, $d=-1$ at the upper-right corner, and $d=0$ halfway in-between. By taking the absolute value, we can thus compute r as $1-|d|$. Also note that a positive distance (inner triangle) represents points on the northern hemisphere, while negative values map to the southern hemisphere. Hence, the sign of d can be used to set the sign of z. We arrive at the following:

$$d = 1-(|u|+|v|), \qquad \text{and} \qquad \begin{aligned} r &= 1-|d|, \\ z &= \operatorname{sign}(d)\cdot(1-r^2). \end{aligned} \tag{7}$$

3.1.2 Avoiding Division-by-Zero

In the center and the four corners of the square, r is exactly zero and Equation 4 results in a division-by-zero. To get a robust solution, we set $\phi' = 0$ if $r = 0$. This is a valid behavior as these points map to the south and north poles, respectively, where the value of ϕ' does not matter. Using intrinsics, the test can be compactly written as follows:

```
mask = _mm_cmpneq_ps(r, ZERO);    // compare r to zero
phi = _mm_and_ps(phi, mask);      // clear phi to 0..0 if r=0
```

The `cmpneqps` (compare not equal) instruction compares r with a predefined zero-constant, and based on the result, ϕ' is either cleared to zero, or kept unchanged. This is valid since the bit sequence $0\ldots0$ also represents a floating-point zero in the IEEE-754 standard.

3.1.3 Approximating the Trigonometric Operations

Using simple logical operations, we map the problem to the first quadrant, and from there, sign extension moves the result back to the correct quadrant. The only remaining difficulty is the trigonometric operations needed to compute x and y in Equation 1. As `sin` and `cos` are not part of the SSE instruction set, polynomial approximations have to be used. Table-based approaches are not recommended as multiple values are computed in parallel, which means multiple memory accesses at different locations and bad cache performance. The most straightforward approach is to truncate the Taylor series (Equation 8) for sine and cosine to an arbitrary number of terms.

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots \quad \text{and} \quad \cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots \tag{8}$$

The Taylor series are obtained by expanding the power series around the point $x=0$. However, the absolute error grows the further from x we get. In our case, the input is in the range $\phi' \in [0, \frac{\pi}{2}]$. If we truncate the expression for $\sin x$ after, e.g., the 7th degree term, we get a maximum absolute error of $1.57 \cdot 10^{-4}$, which is quite large. The power series can be expanded around a point somewhere in the middle of the interval, but the basic problem remains – the error increases quickly as we move away from the chosen point.

A better approach is to use the *minimax* polynomial approximation, which is the polynomial that minimizes the maximum approximation error. This has a number of interesting properties. The Chebyshev equioscillation theorem (see, e.g., [3]) states that for a minimax polynomial of degree n over an interval $a \leq x \leq b$, there exist at least $n+2$ points in the domain where the error between the polynomial and the approximated function oscillate in sign and are of equal magnitude. Thus, the minimax polynomial gives us a well-controlled error over the entire range. Remez algorithm [2] can be used to find the polynomial, but as the details are quite technical, we use the implementation provided by the `numapprox` package in Maple$^{\text{TM}}$. As an example, the 7th order minimax approximation of $\sin x$ for $x \in [0, \frac{\pi}{2}]$ is:

$$\begin{aligned} \sin x \;\approx\; & -1.947 \cdot 10^{-8} + 1.00000155x - 0.000020227x^2 - 0.16657x^3 \\ & -0.00023977x^4 + 0.0086393x^5 - 0.00020575x^6 - 0.00013731x^7, \end{aligned} \tag{9}$$

with a maximum error of only $2.00 \cdot 10^{-8}$, as compared to $1.57 \cdot 10^{-4}$ for the Taylor approximation of the same order. However, note that the evaluation is more expensive, as every term has a nonzero coefficient (7 vs. 3 `mul`'s). To reduce the complexity, we set coefficients for terms of even power to zero, as these are relatively small anyway. This is done by rewriting the problem as:

$$\sin x \approx xp(x^2) \quad \overset{y=x^2}{\Longleftrightarrow} \quad \frac{\sin\sqrt{y}}{\sqrt{y}} \approx p(y). \tag{10}$$

After a change of variables, $y=x^2$, we can find a 3rd order polynomial $p(y)$, which after insertion on the left side gives a 7th order approximation of $\sin x$ with all even coefficients set to zero. Note that the range used for the minimax optimization must be updated to $y \in [0, \frac{\pi^2}{4}]$ as $x \in [0, \frac{\pi}{2}]$. The maximum error is now $1.18 \cdot 10^{-6}$ at a cost of 4 `mul`'s, which is a good compromise.

In our case, we need to compute both $\sin\phi'$ and $\cos\phi'$. The first thing that comes to mind is to approximate only the sine, and then apply the rule $\cos x = \sqrt{1-\sin^2 x}$ to find the cosine. However, the approximation of $\sin x$, which we call $f(x)$, results in an error $\varepsilon_s(x) = \sin x - f(x)$. By analyzing $\varepsilon_s(x)$ over the range $0 \le x \le \frac{\pi}{2}$, we find that the largest error is $\varepsilon_s = 1.18 \cdot 10^{-6}$ at $x = \frac{\pi}{2}$. The error in the cosine approximaton at this point would be:

$$|\varepsilon_c| = |\cos x - \sqrt{1-(\sin x - \varepsilon_s)^2}| = \sqrt{\varepsilon_s(2-\varepsilon_s)} \approx \sqrt{2\varepsilon_s} = 0.0015, \tag{11}$$

which is too large. Hence, we have to perform two separate polynomial approximations (sine and cosine) in order to reach an acceptable precision.

Last, we note that the computation of ϕ' in Equation 4 includes a multiplication by $\frac{\pi}{4}$. By rescaling the coefficients of the minimax polynomials so that we approximate $\sin(\frac{\pi}{4}x)$ rather than $\sin x$, and similarly for the cosine, this extra multiplication can be avoided (without changing the approximation error as we only rescale the input). Finally, we arrive at:

$$\begin{aligned} \sin(\frac{\pi}{4}x) &\approx 0.785398x - 0.0807407x^3 + 0.00248440x^5 - 0.0000341486x^7, \\ \cos(\frac{\pi}{4}x) &\approx 0.999993 - 0.308371x^2 + 0.0157863x^4 - 0.000298371x^6, \end{aligned} \tag{12}$$

where $x \in [0,2]$, and the coefficients have been rounded to 6 significant digits (see the source code for full precision).

3.2 The Sphere to Square Transform

The inverse transform, i.e., the mapping of vectors on the sphere to points in the unit square, is useful in a number of applications. The main steps are:

- Map the problem to the first quadrant by taking the absolute of (x,y).
- Compute ϕ' from (x,y) using `atan`, and compute r from z.
- Convert (r,ϕ') to (u,v) in the first quadrant.
- Flip the sign bits of (u,v) to move the point to the correct quadrant.

– Map points from $[-1,1]^2$ to the unit square $[0,1]^2$.

As many parts of the implementation resemble those described in Section 3.1, we will only briefly go over the details.

3.2.1 Approximating the Arctan-function

Our input is a normalized 3D vector (x,y,z). To compute ϕ', which is the rotation in the first quadrant of the xy-plane, we start by computing the absolute values $|x|$ and $|y|$. The rationale for this is the same as before, i.e., to move the problem to the first quadrant. Then, the rotation is found using: $\phi' = \text{atan}\,\frac{|y|}{|x|}$. Since there is no SSE version of `atan`, we have to use a polynomial approximation here as well. However, it proved hard to find an approximation that yields enough precision for all inputs, as $\frac{|y|}{|x|} \to +\infty$ as $|x| \to 0$. Therefore, we apply the rule:

$$\text{atan}\,\alpha = \begin{cases} \text{atan}\,\alpha & \text{if } \alpha < 1, \\ \frac{\pi}{2} - \text{atan}\,\frac{1}{\alpha} & \text{if } \alpha \geq 1, \end{cases} \tag{13}$$

to reduce the input range to $[0,1]$, i.e., we let $|x|$ and $|y|$ switch places if $|x| < |y|$. Using the SSE instructions `minps` (minimum of two values) and `maxps` (maximum of two values), α can be efficiently computed as:

$$\alpha = \frac{\min(|x|,|y|)}{\max(|x|,|y|)}, \qquad 0 \leq \alpha \leq 1. \tag{14}$$

With this reduced range, it is easier to find a good minimax approximation. We strive for about the same precision as in the approximations of sine and cosine. For the atan-function, *rational* minimax approximations are known to give low errors. In our case, a 3rd/2nd order approximation would be sufficient (maximum error of $7.28 \cdot 10^{-6}$). However, the necessary division is rather slow and have a long latency. Thus, we opted for a 6th order polynomial approximation, which avoids the division and uses the same number of coefficients. As we will see later, it is useful to include a multiplication by $\frac{2}{\pi}$ so that the approximation returns an angle in $[0, \frac{1}{2}]$ rather than $[0, \frac{\pi}{4}]$. Our final approximation is (note $\alpha \in [0,1]$):

$$\begin{aligned} \frac{2}{\pi}\,\text{atan}\,\alpha \;\approx\;& 4.06531 \cdot 10^{-6} + 0.636227\alpha + 0.00615523\alpha^2 - 0.247326\alpha^3 + \\ & 0.0881627\alpha^4 + 0.0419157\alpha^5 - 0.0251427\alpha^6, \end{aligned} \tag{15}$$

and the maximum error is $4.07 \cdot 10^{-6}$. Last, we need to evaluate Equation 13 (scaled by $\frac{2}{\pi}$). This can be done using a compare instruction followed by four logic/arithmetic instructions, as follows:

```
__m128 mask = _mm_cmplt_ps(x, y);   // mask = x<y ? 11..1 : 0
__m128 c = _mm_and_ps(mask, ONE);   // c = x<y ? 1.0 : 0.0
mask = _mm_and_ps(mask, SIGNBIT);   // mask = x<y ? 10..0 : 0
phi = _mm_xor_ps(phi, mask);        // phi = x<y ? -phi : phi
phi = _mm_add_ps(c, phi);           // phi = x<y ? 1-phi : phi
```

3.2.2 Finding the Radius

The computation of r differs depending on whether we are in the northern or the southern hemisphere ($z \geq 0$ or $z < 0$). By rearranging the terms in the equations for z given in Section 2.2, we find that:

$$r = \begin{cases} \sqrt{1-z} & \text{if } z \geq 0 \\ \sqrt{1+z} & \text{if } z < 0 \end{cases} \quad \iff \quad r = \sqrt{1-|z|}. \tag{16}$$

Again, by taking the absolute value, we avoid branching.

3.2.3 Mapping from Disc to Square

The computation of the point (u,v) in the square, corresponding to the point (r,ϕ') in the disc, is relatively straightforward. We assume, for now, that the point lies in the "inner" triangle (northern hemisphere). By inverting the expressions for r and ϕ (Equation 2), we can compute (u,v) as:

$$\begin{cases} r = u+v \\ \phi' = \frac{\pi}{4}\left(\frac{v-u}{r}+1\right) = \ldots = \frac{\pi}{2}\frac{v}{r} \end{cases} \quad \iff \quad \begin{cases} v = r \cdot \frac{2}{\pi}\phi' \\ u = r - v \end{cases} \tag{17}$$

Then, if we are in the southern hemisphere ($z < 0$), the point (u,v) is reflected about the diagonal in the square as follows:

$$u' = 1 - v \qquad \text{and} \qquad v' = 1 - u. \tag{18}$$

This is implemented using a compare instruction followed by logic/arithmetic instructions similar to what we did in Section 3.2.1. The next step is to sign-extend (u,v) based on the original signs of x and y, i.e., we take the point from the first quadrant to its correct position. Finally, we transform the point to the unit square, $[-1,1]^2 \to [0,1]^2$, which completes the transform.

3.3 Precision

The approximations of the trigonometric operations were chosen to provide sufficient precision for all but the most demanding applications. To measure the approximation errors, we transformed a large number (10^9) of random points in the square to the sphere, using both the "exact" scalar version of the algorithm (using built-in trigonometric instructions) and our SSE-optimized version. The error was measured as the Euclidean distance in 3D between the resulting points on the sphere. As the error is very small, this measure is approximately the same as the arc length in radians on the unit sphere.

For the inverse transform, a large number of points over the sphere were mapped to points in the square. To measure the error, these 2D points were transformed back to the sphere using the exact algorithm, and the Euclidean distance was measured as before. The maximum and average errors are given in Table 1. The average error is in the order of $3 \cdot 10^{-6}$ in both directions. As a comparison, the diagonal of a single pixel in a 4096×4096 image has a shortest length of $7.7 \cdot 10^{-4}$ when mapped to the sphere.[1] The average error is thus only about 0.3% of the edge length of a pixel in a 4k map. The maximum error is well-behaved in the square to sphere transform. In its inverse, there are a few bad cases where the error goes up to about $1/3$ of a pixel (again assuming a 4k map), even though the average error is low. If a higher precision is required, it is easy to increase the order of the arctan approximation.

transform	maximum error	average error
square→sphere	$7.49 \cdot 10^{-6}$	$3.37 \cdot 10^{-6}$
sphere→square	$2.43 \cdot 10^{-4}$	$3.19 \cdot 10^{-6}$

Table 1: *The maximum and average approximation errors of the forward and inverse transforms. The errors were measured as the Euclidean distance between the points on the unit sphere representing the exact and approximated directions.*

4 Boundary Symmetry

4.1 Tiling

Each edge of the octahedral map is folded about its midpoint so that its two endpoints meet. This *boundary symmetry* [5] is useful as it means the map can be tiled by mirroring the mapping about both its axes for every other occurrence[2], as shown in Figure 4 (c). Hence, lookups for coordinates outside the $[0,1)^2$ range are trivial, and interpolation across the edges of the map is well-defined and without singularities.

For a map with $N \times N$ pixels, where $N = 2^k$ is a power-of-two, the coordinate transform from a point (s,t) to integer pixel coordinates (x,y), can be efficiently done using logic operations. We scale the input by N and truncate:

$$x = \lfloor s \cdot N \rfloor \qquad \text{and} \qquad y = \lfloor t \cdot N \rfloor, \tag{19}$$

where $\lfloor . \rfloor$ is the floor-operator The bits $0 \ldots k-1$ of the binary representation of x and y hold the pixel coordinates within the square, while bits $k, k+1, \ldots$ indicate which repetition of the tiling we are in. Mirroring needs to be done when either x *or* y is at an odd number of repetitions. We can thus `xor` x and y, and look at the kth bit to decide whether to mirror or not:

$$m = (x \oplus y) \,\&\, N, \qquad \text{if} \begin{cases} m = 0 & \rightarrow \quad \text{do not mirror} \\ m \neq 0 & \rightarrow \quad \text{mirror} \end{cases} \tag{20}$$

[1]With a map of $N \times N$ pixels, there are $2N$ pixels crossing the equator, while the circumference of the unit sphere is 2π. Hence, the shortest diagonal is of length π/N.

[2]This is a variant of the "mirrored repeat" tiling used by OpenGL.

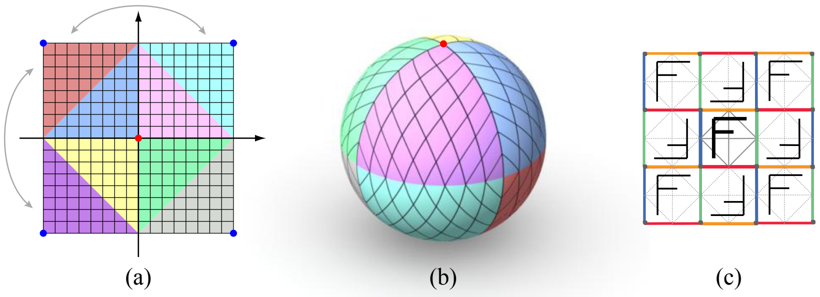


Figure 4: *Our mapping transforms square pixels (a) into curvilinear quads on the sphere (b), while preserving fractional area. The boundary symmetry [5] of the octahedral map is shown in (c). The rightmost image is courtesy of Emil Praun and Hugues Hoppe.*

The final positions are computed as $(x,y) \mod N$, followed by $x = (N-1) - x$ (and similar for y) if $m \neq 0$. The mod-operator translates to an `and` by $N-1$, which is the bit mask with 1's at the positions $0 \ldots k-1$, and 0's elsewhere.

4.2 Bilinear Interpolation

For bilinear interpolation, the four pixels nearest to the point (s,t) must be accessed, and their respective interpolation weights computed. In a map of $N \times N$ pixels, we define each pixel's center to be at $(x+0.5, y+0.5)/N$, where (x,y) are the integer coordinates of the pixel. To find the coordinates of the top-left pixel, we scale (s,t) by N and offset by 0.5, as follows:

$$x = \lfloor s \cdot N - 0.5 \rfloor \qquad \text{and} \qquad y = \lfloor t \cdot N - 0.5 \rfloor. \tag{21}$$

The fractional distance (α_x, α_y) from the top-left pixel's center is given by: $\alpha_x = (s \cdot N - 0.5) - x$ (similar for α_y). Based on this, the coordinates and the weights for the bilinear interpolation are given by:

pixel	x-coordinate	y-coordinate	weight
0	x	y	$(1-\alpha_x) \cdot (1-\alpha_y)$
1	$x+1$	y	$\alpha_x \cdot (1-\alpha_y)$
2	x	$y+1$	$(1-\alpha_x) \cdot \alpha_y$
3	$x+1$	$y+1$	$\alpha_x \cdot \alpha_y$

As the coordinates may lie outside the $\{0, \ldots, N-1\}$ range, we perform the wrapping described in Section 4.1 on each of the four pixels. For this purpose, we have written a code snippet which computes all four pixel positions, and their respective weights, in parallel using SIMD instructions. We store x and y as 16-bit integers and pack all eight combinations into a single 128-bit XMM register. Using SSE2 integer instructions, we perform the wrapping on all eight values in parallel. Finally, the coordinates are unpacked into 32-bit integers and the pixel addresses computed using bit shifts.

The lack of a floor-operator in the SSE/SSE2 instruction set presents a minor difficulty. One option is to use `cvttps2dq` (convert float to int with truncation), but this gives unexpected results for negative inputs as it truncates towards zero. Instead, we rewrite the floor-operator using rounding:

$$\lfloor x \rfloor = \mathrm{round}(x - 0.5). \tag{22}$$

The `cvtps2dq` (convert float to int) instruction performs correct rounding (assuming the rounding control bits in the MXCSR register have been correctly setup). Thus, Equation 21 can be rewritten:

$$x = \mathrm{round}(s \cdot N) - 1 \qquad \text{and} \qquad y = \mathrm{round}(t \cdot N) - 1. \tag{23}$$

In total, we use 27 SSE/SSE2 instructions to compute all pixel indices and weights. As we use 16-bit integers, the map size is limited to 64k $\times$ 64k pixels.

5 Performance

We have implemented three different versions of the forward and inverse transforms: 1) a straightforward scalar version with branching and trigonometric operations, 2) an optimized scalar version using the tricks described here, and 3) a vectorized version using SSE instructions to transform four points/vectors in parallel. To evaluate the performance, we have run the algorithms on sets of N random 2D points in the unit square, and N random 3D vectors on the sphere respectively, using a large number of iterations.

As the computations are performed repeatedly on the same data, we largely avoid cache effects and measure pure computational performance. For the larger datasets, the memory bandwidth limits the performance slightly. The total memory use for each test is $20N$ bytes, and all timings are reported as *clock cycles per single transform*. Thus, transforming a set of N points/vectors takes approximately Nt clock cycles, where t is the reported timing. Note that N has to be a multiple of four, as we use 4-wide SIMD code. All tests were executed on an Intel 45nm quad-core "Penryn" CPU running at 3.2GHz using one core, and the code was compiled using gcc 4.0.1 on Mac OS X 10.5.2.

The performance of the square to sphere transform is summarized in Table 2. The vectorized version is a factor 3.5$\times$–4.2$\times$ faster than the optimized scalar code, and 6.5$\times$–8.6$\times$ faster than the standard version. The numbers for the inverse transform (sphere to square) are listed in Table 3. Here, the speedup is a factor 4.6$\times$–5.0$\times$ compared to the optimized scalar code, and 5.0$\times$–6.4$\times$ compared to the standard version. Theoretically, SSE-optimized code should be up to a factor 4$\times$ faster than a corresponding scalar implementation, as it uses 4-wide instructions. However, the differences in the instruction sets make it possible to exceed this limit. For example, branching is avoided by using compare-instructions together with logical operations.

The achieved speedup can make a significant difference. As an example, consider ray tracing with 256 rays/pixel, with sampling directions computed in the unit square and mapped to the sphere using our transform. With a fast ray tracer capable of 5 million rays/second on a single core, the total rendering time for a 1 megapixel image would be about 61.0 seconds, using the standard scalar transform. With the SIMD version, the rendering time goes down to 52.7 seconds – a 13.5% improvement.

N	scalar standard	scalar optimized	SSE optimized
256	121.9	77.9	18.8
4k	156.2	77.9	18.7
64k	160.7	78.0	18.7
1M	162.4	80.4	22.8
16M	162.5	80.5	22.8

Table 2: *Execution times of our three different implementations of the* square to sphere *transform, using datasets of N 2D points as input. The timings are reported as number of clock cycles per transformed point.*

N	scalar standard	scalar optimized	SSE optimized
256	114.7	106.9	23.0
4k	145.1	115.8	23.2
64k	147.8	115.7	23.2
1M	149.1	117.7	25.6
16M	149.0	117.5	25.6

Table 3: *Execution times for the inverse transform, i.e.,* sphere to square. *The timings are reported as number of clock cycles per transformed vector.*

Acknowledgements

The author was supported by the Swedish Foundation for Strategic Research and by Intel Corporation. Thanks also to Tomas Akenine-Möller and Jacob Munkberg for proof-reading and many valuable suggestions, and to Jacob for help with Figure 4.

A Proof of Area Preservation

Here, we present a formal proof that the described mapping, $P\colon (u,v) \rightarrow (x,y,z)$, from the square to the sphere, indeed preserves fractional area. The magnitude of the vector product of the partial derivatives of P with respect to u and v gives the area-distortion, dA, of the transform:

$$dA = \left\| \frac{\partial P}{\partial u} \times \frac{\partial P}{\partial v} \right\| = \left\| \begin{array}{c} y_u z_v - z_u y_v \\ z_u x_v - x_u z_v \\ x_u y_v - y_u x_v \end{array} \right\|, \tag{24}$$

where we have used the shorthand notation x_u for $\partial x / \partial u$, and so on. We consider the inner triangle of the first quadrant in the square $(u,v) \in [-1,1]$ (see Figure 3), and expand the expressions for the partial derivatives of (x,y,z) with respect to u and v. Due to symmetry, the same proof applies to the other parts of the map. Using the chain rule, and noting that $r_u = r_v = 1$ (as $r = u+v$), we get:

$$\begin{array}{rclcl} x_u &=& x_r r_u + x_\phi \phi_u &=& x_r + x_\phi \phi_u, \\ x_v &=& x_r r_v + x_\phi \phi_v &=& x_r + x_\phi \phi_v, \end{array} \tag{25}$$

with similar expressions for the y and z components. Combining Equation 24 and 25, and simplifying, we obtain:

$$dA = \left\| \begin{array}{c} (\phi_v - \phi_u)(y_r z_\phi - z_r y_\phi) \\ (\phi_v - \phi_u)(z_r x_\phi - x_r z_\phi) \\ (\phi_v - \phi_u)(x_r y_\phi - y_r x_\phi) \end{array} \right\|. \tag{26}$$

Further, the partial derivatives of ϕ (Equation 2) can be written:

$$\phi_u = -\frac{\pi}{2} \frac{v}{(u+v)^2} \qquad \text{and} \qquad \phi_v = \frac{\pi}{2} \frac{u}{(u+v)^2}, \tag{27}$$

which gives:

$$\phi_v - \phi_u = \frac{\pi}{2} \frac{1}{u+v} = \frac{\pi}{2r}. \tag{28}$$

Now, we are ready to write down the expressions for the partial derivatives of (x,y,z) with respect to r and ϕ:

$$\begin{array}{rclcrcl} x_r &=& \frac{2(1-r^2)}{\sqrt{2-r^2}} \cos\phi & & x_\phi &=& -r\sqrt{2-r^2} \sin\phi \\ y_r &=& \frac{2(1-r^2)}{\sqrt{2-r^2}} \sin\phi & & y_\phi &=& r\sqrt{2-r^2} \cos\phi \\ z_r &=& -2r & & z_\phi &=& 0 \end{array} \tag{29}$$

Applying Equation 28 and 29, Equation 26 reduces to:

$$dA \;=\; \pi \left\| \begin{array}{c} y_\phi \\ -x_\phi \\ 1-r^2 \end{array} \right\| \;=\; \pi \sqrt{y_\phi^2 + (-x_\phi)^2 + (1-r^2)^2} \;=\; \ldots \;=\; \pi. \tag{30}$$

Hence, the fractional area grows by a factor π when we go from the square $(u,v) \in [-1,1]$ to the sphere $\mathscr{S}$. This is consistent with our expectations as the area of the square is 4, and the surface area of the unit sphere is 4π.

Bibliography

[1] CLARBERG, P., AND AKENINE-MÖLLER, T. Practical Product Importance Sampling for Direct Illumination. In *Proceedings of Eurographics* (2008).

[2] FRASER, W. A Survey of Methods of Computing Minimax and Near-Minimax Polynomial Approximations for Functions of a Single Independent Variable. *Journal of the ACM 12*, 3 (1965), 295–314.

[3] HART, J. F. *Computer Approximations*. Krieger Pub. Co., 1978.

[4] KAJIYA, J. T. The Rendering Equation. *Computer Graphics (Proceedings of ACM SIGGRAPH), 20*, 4 (1986), 143–150.

[5] PRAUN, E., AND HOPPE, H. Spherical Parametrization and Remeshing. *ACM Transactions on Graphics, 22*, 3 (2003), 340–349.

[6] SHIRLEY, P., AND CHIU, K. A Low Distortion Map between Disk and Square. *Journal of Graphics Tools, 2*, 3 (1997), 45–52.

Paper VI

An Optimizing Compiler for Automatic Shader Bounding

Petrik Clarberg[*] Robert Toth[*] Jon Hasselgren[*]
Tomas Akenine-Möller[*†]

[*]Intel Corporation [†]Lund University

Abstract

Programmable shading provides artistic control over materials and geometry, but the black box nature of shaders makes some rendering optimizations difficult to apply. In many cases, it is desirable to compute bounds of shaders in order to speed up rendering. A bounding shader can be automatically derived from the original shader by a compiler using interval analysis, but creating optimized interval arithmetic code is non-trivial. A key insight in this paper is that shaders contain metadata that can be automatically extracted by the compiler using data flow analysis. We present a number of domain-specific optimizations that make the generated code faster, while computing the same bounds as before. This enables a wider use and opens up possibilities for more efficient rendering. Our results show that on average 42–44% of the shader instructions can be eliminated for a common use case: single-sided bounding shaders used in lightcuts and importance sampling.

Computer Graphics Forum (Proceedings of EGSR 2010), 29(4):1259–1268, 2010.

1 Introduction

The advent of highly realistic computer-generated graphics in feature films and games has largely been made possible by the separation of rendering algorithms and visual content. Programmable shading provides means for artists to create wonderful environments, without having to deal with much of the technicalities of the renderer. From the rendering system's point of view, a shader is a black box, which can only be point-sampled. This presents a problem, as higher level information about a shader is often required to make use of more efficient rendering algorithms. For example, in global illumination where the light transport integrals are very complex, it is critical to be able to compute *bounds* of a shader in order to use algorithms such as lightcuts [29, 28] and importance sampling. In a rasterization pipeline, shader bounds may be used to avoid computations that do not contribute to the image [10].

A shader bounding function for an arbitrary shader can be carefully handcrafted, but this is tedious and error-prone for all but the simplest shaders. Alternatively, a compiler can be used to automatically derive a bounding function using interval analysis [13, 10, 26]. The compiler transforms the shader instructions to operate on *intervals* rather than single values. The end result is a *bounding shader*, which given bounds on the shader inputs computes bounds on its outputs, i.e., a conservative range of possible outputs. Some examples are shown in Figure 1.

There are two factors directly affecting the efficiency of the rendering system: the execution time of the bounding shader, and the tightness of the computed bounds. In this paper, we focus on generating faster bounding shaders, without changing the computed bounds. Naïve transformation from scalar to interval shader code is relatively straightforward for a compiler. Each arithmetic instruction is replaced by an instruction sequence that performs the same operation on intervals. How-

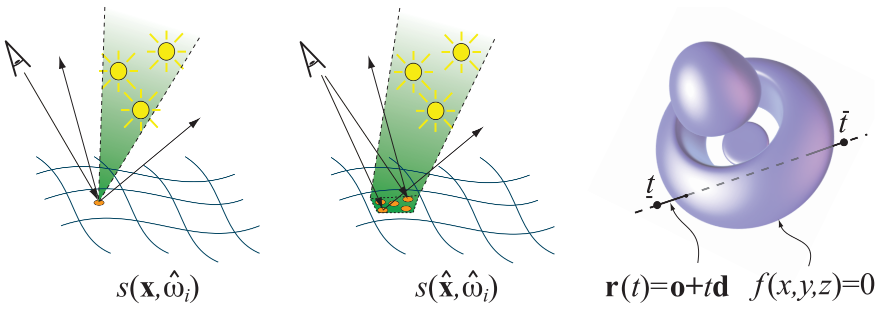

$\bar{t}$
$\underline{t}$
$s(\mathbf{x},\hat{\omega}_i)$
$s(\hat{\mathbf{x}},\hat{\omega}_i)$
$\mathbf{r}(t)=\mathbf{o}+t\mathbf{d}$
$f(x,y,z)=0$

Figure 1: *The bounding shader s computes bounds for the shaded result at a specific shading point,* $\mathbf{x}$*, or cluster of points,* $\hat{\mathbf{x}}$*, given bounds on the light directions,* $\hat{\omega}_i$*. This is critical functionality in lightcuts and importance sampling. The right image shows a ray tracing application, where a bounding shader is used to find the closest intersection.*

ever, by exploiting domain-specific knowledge and data flow analysis, we show that it is possible to achieve much better results. Although we focus on interval arithmetic (IA) [17], where an interval is simply represented as a minimum and a maximum value, higher-order methods such as affine arithmetic [5], or Taylor model arithmetic [3] can also be used.

One of our key contributions is a method we call *static bounds analysis*, in which bounds are propagated through the shader at compile time in order to determine the type and possible range of each variable. This information allows us to generate optimized interval arithmetic code. Similar results cannot be achieved using standard compiler techniques. We also propose an optional extension called *valid range analysis*, which exploits the fact that some instructions put strict limits on their inputs. Another important contribution is the use of *dynamic bounds assumptions*, which creates a fast path for the common case, but falls back on a more general bounding shader if the assumptions fail at runtime.

2 Related Work

Originally developed in the 1950s to compute bounds on rounding errors in numerical computations, interval analysis [17] is now used in a wide range of scientific and engineering disciplines.

Hardware/Software Support for Interval Analysis Specialized hardware architectures with support for interval analysis have been proposed, see, e.g., [23], but the extra area/power is hard to motivate for non-scientific workloads. Software implementations provide more flexibility and are currently the only alternative for graphics applications. There are, however, few compilers with support for interval analysis on existing hardware [24, 1], and most users are left to using publicly available libraries [27] or GPU implementations [4]. It should be noted that none of the existing implementations apply any interval-specific compiler optimizations.

Interval Analysis in Computer Graphics A full overview is beyond the scope of this paper, so we limit the discussion to a few selected applications. Snyder [25] use interval analysis to robustly solve a wide range of problems related to parametric surfaces. Mitchell [16] proposed to use interval arithmetic for robust ray tracing of implicit surfaces, which initiated a lot of work in this direction. See Hijazi et al.'s survey [14] for an overview. Collision detection is another successful application of interval analysis, with methods for implicit surfaces [7], rigid bodies [21], and articulated models [33]. Some applications, especially with long interval computation chains, benefit from using higher-order methods, such as affine arithmetic [5] and Taylor models [3].

Programmable Shading Programmable shading has been a keystone in offline rendering for more than two decades. The RenderMan shading language [9] was one of the first languages for production-quality rendering, and it introduced many of the concepts used by today's shading languages. Programmable shading is also critical for bringing flexibility to ray tracing systems [19]. GPUs capable of executing shaders first appeared in 2001, and recent graphics APIs define many types of shaders, each performing a specific task in the graphics pipeline.

Interval Analysis of Programmable Shaders A variety of applications have used interval analysis to compute bounds of shaders. Greene and Kass [8] bound shaders that can be expressed in data flow form (i.e., shaders without intricate control flow) to achieve error-bounded antialiasing. They use a compiler to automatically generate interval arithmetic code [15]. Standard compiler optimizations (e.g., common-subexpression elimination) and mathematical simplifications are applied, but no interval-specific optimizations. Heidrich et al. [13] compute bounds on procedural RenderMan shaders using affine arithmetic for sampling purposes. They note that affine arithmetic should not be used for expressions involving only non-affine values, and that it is important to use optimized approximations, e.g., square instead of general multiplication, where possible. A similar framework has been used for ray tracing of procedural displacement shaders [12].

In real-time graphics, interval analysis has recently been used to compute conservative bounds for fragment programs over a tile of pixels in order to discard, or *cull*, shading computations [10]. The authors propose a hardware architecture capable of interval arithmetic, thereby avoiding the problem of generating optimized scalar code from an interval-based shader. Later work has extended this concept to compute bounds on vertex programs and cull base primitives prior to tessellation in a DirectX 11-style pipeline using Taylor model arithmetic [11].

Velázquez-Armendáriz et al. [26] present a basic compiler for automatic bounding shading generation, and showcase its utility on a wide range of photo-realistic rendering examples. We extend their work with a number of domain-specific optimizations for generating faster code.

3 Interval Analysis Primer

The goal of interval analysis is to compute conservative bounds for arbitrary computations. This is done by redefining all operations to operate on intervals rather than individual values. We limit ourselves to extended real numbers (i.e., including $\pm\infty$), and define an interval as:

$$\hat{a} = [\underline{a}, \overline{a}] = \{x \in \mathbb{R} \,|\, \underline{a} \leq x \leq \overline{a}\}. \tag{1}$$

3.1 Arithmetic Operations

The basic arithmetic operations are easily extended to operate on intervals. Addition and subtraction become:

$$\hat{a}+\hat{b} = \left[\underline{a}+\underline{b}, \overline{a}+\overline{b}\right] \quad \text{and} \quad \hat{a}-\hat{b} = \left[\underline{a}-\overline{b}, \overline{a}-\underline{b}\right]. \tag{2}$$

Multiplication is slightly more complicated due to the cases where one or both of the intervals overlap zero:

$$\hat{a} \cdot \hat{b} = \left[\min(\underline{a}\underline{b}, \underline{a}\overline{b}, \overline{a}\underline{b}, \overline{a}\overline{b}), \max(\underline{a}\underline{b}, \underline{a}\overline{b}, \overline{a}\underline{b}, \overline{a}\overline{b})\right]. \tag{3}$$

Some operations, such as division, are more complex:

$$\begin{aligned} [\underline{a},\overline{a}] / [\underline{b},\overline{b}] &= [\underline{a},\overline{a}] \cdot \left(1/[\underline{b},\overline{b}]\right), \quad \text{where} \\ 1/[\underline{b},\overline{b}] &= \begin{cases} [1/\overline{b}, 1/\underline{b}] & \text{if} \quad 0 \notin [\underline{b},\overline{b}], \\ [-\infty, \infty] & \text{if} \quad 0 \in [\underline{b},\overline{b}]. \end{cases} \end{aligned} \tag{4}$$

In the last example, useful information is lost if the denominator interval contains zero. This problem is not limited to division, but applies to any function that is piecewise continuous, e.g., tan. By working with *multi-intervals*,

$$\hat{a} = \bigcup_i [\underline{a}_i, \overline{a}_i], \tag{5}$$

we can split $[\underline{b},\overline{b}]$ into $[\underline{b},0] \cup [0,\overline{b}]$ and get:

$$1/[\underline{b},\overline{b}] = [-\infty, 1/\underline{b}] \cup [1/\overline{b}, \infty] \quad \text{if} \quad 0 \in [\underline{b},\overline{b}]. \tag{6}$$

Such multi-intervals may be further split, or merged if overlapping, depending on the operations performed. We propose using multi-intervals for the compile-time analysis, but single intervals at runtime for efficiency reasons.

3.2 General Functions

A function, $f : \mathbb{R}^n \to \mathbb{R}$, may be extended to interval form to compute bounds for the result based on bounds of the arguments. We define the *interval extension*, $\hat{f}$, of f as:

$$\hat{f}(\hat{\mathbf{x}}) \supseteq \{f(\mathbf{x}) \,|\, \mathbf{x} \in \hat{\mathbf{x}}\}. \tag{7}$$

Using interval arithmetic, we treat $\hat{\mathbf{x}} = (\hat{x}_1, \ldots, \hat{x}_n)$ as individual intervals and compute intervals for each intermediate result independently. Tighter bounds may be achieved by keeping information on how the intermediate results depend on $\hat{\mathbf{x}}$. These dependencies may be modeled as linear functions (affine arithmetic), or at a higher cost, as general polynomials (Taylor model arithmetic).

3.3 Rounding Considerations

In practice, an application using interval arithmetic is forced to work with floating-point numbers of finite precision. Most general purpose implementations are designed to guarantee interval enclosure of real operations, i.e., produce mathematically conservative results. This requires *outward* rounding, where the computations of the upper/lower intervals are rounded up/down respectively.

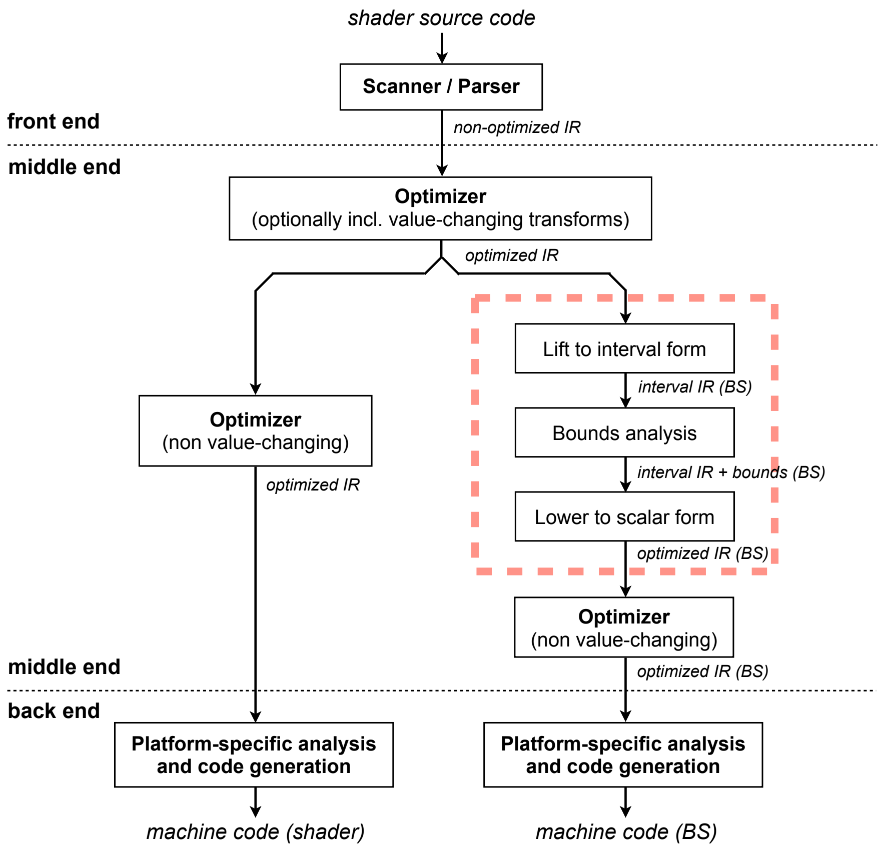


Figure 2: *The middle end of the compiler performs initial optimizations on the intermediate representation (IR) before lifting it to interval form (right) to get a bounding shader (BS). After bounds analysis and lowering to optimized scalar form, further non-value changing optimizations may be applied. The front and back ends are the same as in a traditional shader compiler (left).*

In computer graphics, most applications ignore round-off errors and view the result of a shader evaluation as ground truth. Thus, it is sufficient to compute bounds that are conservative only up to machine precision. That is, a shader should never return a result outside the bounds computed by the corresponding bounding shader, but we need no guarantee that the bounds are mathematically conservative. This is ensured if there is a path through the bounding shader that executes the same sequence of floating-point operations, using the same rounding mode as in the original shader. Normally, this is always the case, as each interval operation performs the equivalent floating-point operation on all relevant combinations of extreme values, and then picks the outer bounds using min/max operations.

3.4 Application to Shader Languages

In addition to arithmetic operations, shader languages (e.g., RenderMan, GLSL, and HLSL) support functionality such as conditionals, loops, and texture lookups, which must be handled. Many of these features have been extensively studied. Interval-based texture lookups can be done using mipmap hierarchies of minimum and maximum values [18]. The bounds on the texture coordinates are used to compute an appropriate mipmap level, from which a number of min/max samples are drawn. Cube map lookups can be handled similarly. Previous work has also explored how to handle noise functions and derivatives [13].

Conditionals present a problem, as whenever overlapping intervals are compared, the condition is not unambiguously true or false. One solution is to use step functions and rewrite conditional expressions as arithmetic [13]. A more general method, which we use, is to evaluate both execution paths and merge the results. This is most easily done by transforming the intermediate representation to *static single assignment* (SSA) form [6], and treating the Φ functions that are inserted at joints in a non-standard way (they are normally replaced by copies) as selector functions [26] .

4 An Optimizing Interval Compiler

In this section, we will introduce a compiler infrastructure and a number of optimizations targeted at generating faster bounding shaders. There are two main steps involved. First, we *lift* the original shader to interval form, i.e., replace scalar instructions by their interval arithmetic equivalents. This lifting step is straightforward, and can be done trivially by a compiler with an intermediate representation (IR) supporting interval instructions.

The second step is to *lower* the shader on interval form to efficient scalar code. The standard approach is to replace each interval instruction by a general sequence of scalar instructions performing the desired interval computation. However, better results can be obtained if we have prior knowledge about what range of values each instruction operates on. Much of this paper deals with automatic extraction of such information through *bounds analysis*.

Figure 2 shows the proposed design of an optimizing compiler for automatic generation of bounding shaders (BS). We will focus on the algorithms enclosed in dashed red, and first look at some examples to highlight the types of optimized interval to scalar conversions that are possible. Then, we will discuss the lifting step and introduce several novel algorithms for compile-time bounds analysis.

4.1 Lowering to Optimized Scalar Form

Most interval operations can be implemented in a variety of ways depending on the generality of the computation. For example, multiplying two positive intervals is much easier than the case where the intervals may overlap zero. Our goal is to select the most compact implementation possible, exploiting bounds information determined at compile time.

We define a table of implementation alternatives for each instruction, along with the necessary conditions. There is always a fully general implementatation, which works under *any* condition. This job is tedious, but fortunately has to be done only once. Note that the listed conditions are applied at compile time, so the implementation must be valid for all possible runtime intervals matching the condition. Next, we will look at a few illustrative examples.

Multiplication The table below shows different possibilities for evaluating the multiplication operator with *scalar-scalar*, *scalar-interval*, and *interval-interval* operands. In this case, knowing the signs of one or both of the operands enables significantly more efficient code (the cases where $\hat{a}$ and $\hat{b}$ are reordered have been left out for brevity):

expr	**condition**	**implementation**	**#inst**
$x \cdot y$	any	$x \cdot y$	1
$x \cdot \hat{a}$	any	$[\min(x\underline{a}, x\overline{a}), \max(x\underline{a}, x\overline{a})]$	4
	$x \geq 0$	$[x\underline{a}, x\overline{a}]$	2
	$x \leq 0$	$[x\overline{a}, x\underline{a}]$	2
$\hat{a} \cdot \hat{b}$	any	$[\min(\underline{a}\underline{b}, \underline{a}\overline{b}, \overline{a}\underline{b}, \overline{a}\overline{b}),$ $\max(\underline{a}\underline{b}, \underline{a}\overline{b}, \overline{a}\underline{b}, \overline{a}\overline{b})]$	10
	$\hat{a} \geq 0$	$[\min(\underline{a}\underline{b}, \overline{a}\underline{b}), \max(\underline{a}\overline{b}, \overline{a}\overline{b})]$	6
	$\hat{a} \leq 0$	$[\min(\underline{a}\overline{b}, \overline{a}\overline{b}), \max(\underline{a}\underline{b}, \overline{a}\underline{b})]$	6
	$\hat{a}, \hat{b} \geq 0$	$[\underline{a}\underline{b}, \overline{a}\overline{b}]$	2
	$\hat{a} \leq 0 \leq \hat{b}$	$[\underline{a}\overline{b}, \overline{a}\underline{b}]$	2
	$\hat{a}, \hat{b} \leq 0$	$[\overline{a}\overline{b}, \underline{a}\underline{b}]$	2

Square If the operands of a multiplication come from the same source, we can use a more efficient *square* operator:

expr	**condition**	**implementation**	**#inst**
x^2	any	$x \cdot x$	1
$\hat{a}^2$	any	$[\max(\underline{a}, -\overline{a}, 0)^2, \max(-\underline{a}, \overline{a})^2]$	7
	$\hat{a} \geq 0$	$[\underline{a}\underline{a}, \overline{a}\overline{a}]$	2
	$\hat{a} \leq 0$	$[\overline{a}\overline{a}, \underline{a}\underline{a}]$	2

Absolute Value Some operations can be completely removed under special circumstances. An absolute value, for example, requires no evaluation for positive operands:

expr	condition	implementation	#inst
$\|x\|$	any	$\|x\|$	1
	$x \geq 0$	x	0
$\|\hat{a}\|$	any	$[\max(\underline{a}, -\overline{a}, 0), \max(-\underline{a}, \overline{a})]$	5
	$\hat{a} \geq 0$	$[\underline{a}, \overline{a}]$	0
	$\hat{a} \leq 0$	$[-\overline{a}, -\underline{a}]$	2

Square Root Other instructions are valid only on a limited range, e.g., a square root requires a positive argument. If we at compile time can guarantee the result will be *Not-a-Number* (NaN), we generate a compiler warning (shown in red) as this condition likely indicates a programming error:

expr	condition	implementation	#inst
$\sqrt{x}$	any	$\sqrt{x}$	1
	$x < 0$	NaN	0
$\sqrt{\hat{a}}$	any	$[\sqrt{\underline{a}}, \sqrt{\overline{a}}]$	2
	$\hat{a} < 0$	[NaN, NaN]	0

Additional Notes It is often the sign of operands that differentiate between implementation alternatives, but there are exceptions. The interval pow$(\hat{a}, \hat{b})$ function, for example, is monotonically increasing if $\hat{a} \geq 1$, and decreasing if $\hat{a} \leq 1$. The general case requires 4 pow and 6 min/max instructions, but it can be reduced to only 2 pow in some cases.

In these examples, we have counted each basic instruction (add, mul, neg etc) as a single instruction for simplicity. Assuming the bounding shader is executed on a throughput-oriented architecture, this should be a rough approximation of its execution cost. Naturally, the cost depends on the hardware and the exact implementations chosen.

4.2 Lifting to Interval Form

In the lifting step, we analyze the dependency graph to determine which instructions are operating on interval data. Those instructions are replaced by their equivalents operating on intervals rather than scalar values, e.g., an add instruction is replaced by an interval-add and so on. We assume the user or application specifies which inputs are given as intervals. This can be accomplished by annotating selected inputs with an `interval` keyword [24], or by designing the compiler API to allow sufficient control. Next, we will look at a simple example. Consider the function:

```
float f(interval float a, interval float b) {
    float c = abs(a);
    return sqrt(c * b);
}
```

After lifting this to interval form and converting to three-address code on SSA form [6], we get:

expression	**#inst**
$\hat{a}_0 \leftarrow \hat{a}$	
$\hat{b}_0 \leftarrow \hat{b}$	
$\hat{c}_0 \leftarrow \lvert\hat{a}_0\rvert$	5
$\hat{t}_1 \leftarrow \hat{c}_0 \cdot \hat{b}_0$	10
$\hat{t}_2 \leftarrow \sqrt{\hat{t}_1}$	2
`return` $\hat{t}_2$	
sum:	17

The cost of each interval operation is measured in terms of scalar instructions, assuming the most general implementation of each instruction, as defined in Section 4.1. This is what a compiler not using our optimizations would generate.

4.3 Static Bounds Analysis

As we have seen in Section 4.1, there is a lot to gain from using optimized implementations of the interval operations. We propose to use data flow analysis to determine conservative bounds on each shader variable at compile time. First, initial bounds are assigned to each shader input. The bounds are then propagated through the shader using a generalized form of interval arithmetic. The method is related to constant propagation [31], but instead of compile-time constants, we track bounds for the runtime range of each variable. These are then used to pick the most efficient implementation of each instruction when lowering the code to scalar form. Next, we will go over the details.

Initial Bounds The data type of a variable is the first source of loose conservative bounds, simply due to the range of representable numbers the variable can hold. Most shading languages support at least a subset of the following basic types:

data type	**possible range**
half, float, double	$\{x \in \mathbb{R} \mid -\infty \leq x \leq \infty, \mathrm{NaN}\}$
integer (n bits)	$\{x \in \mathbb{Z} \mid -2^{n-1} \leq x \leq 2^{n-1}-1\}$
unsigned integer (n bits)	$\{x \in \mathbb{Z} \mid 0 \leq x \leq 2^{n}-1\}$
boolean	$\{\texttt{true}, \texttt{false}\}$
signed normalized	$\{x \in \mathbb{R} \mid -1 \leq x \leq 1\}$
unsigned normalized	$\{x \in \mathbb{R} \mid 0 \leq x \leq 1\}$

For aggregate types, such as vectors and matrices, we assign bounds to each element individually based on its basic type.

Shader languages usually also support a number of pre-defined *state* variables with basic information about the point being shaded. In HLSL, these are accessed using shader input semantics, e.g., `SV_Position` (D3D10), and in GLSL there are built-in variables, such as `gl_FragCoord`. Some state variables have strict limits set by the rendering system. For example, `SV_Position` in the pixel shader specifies the sample position in screen space coordinates (x,y), and the depth z in normalized device coordinates. Hence we know that $z \in [0,1]$ and $x,y \in [0,N]$, where N is the size of the largest supported render target.

To be fully general, we represent a variable's compile-time bounds as a union of zero or more discrete values and zero or more disjoint intervals. Given a variable, x, we define its bounds, B_x, as:

$$B_x = \bigcup_i B_x^i, \qquad \text{where } B_x^i = \begin{cases} x_i, & \text{or} \\ [\underline{x}_i, \overline{x}_i]. \end{cases} \tag{8}$$

Additionally, we have found it useful to store flags indicating whether a variable can be NaN, and whether both positive and/or negative zeros are possible. The latter is particularly useful to optimize division so that $1/[0,x] = [1/x,\infty]$ rather than $[-\infty,\infty]$ if the denominator is known to exclude -0.

Bounds Propagation In the compile-time propagation of bounds information, we use a generalized form of interval arithmetic, which handles bounds on the form in Equation 8. For binary operations, $z \leftarrow x \text{ op } y$, we evaluate all combinations of discrete values and/or intervals in x and y:

$$B_z = \bigcup_{i,j} (B_x^i \text{ op } B_y^j). \tag{9}$$

Any overlapping bounds are merged to create a new disjoint set of bounds representing the possible values of z. Unary operations, $y \leftarrow \text{ op } x$, are handled by applying the operator to each B_x^i. Type casts retain bounds as far as possible and may be a source of additional bounds information. For instance, a bool cast to float in GLSL/HLSL will be a variable in $\{0,1\}$ rather than $[-\infty,\infty]$.

In practice, a variable rarely has more than a single interval bound or a few discrete values, but there are cases that give rise to more complex bounds information. For example, branches can introduce different assignments to a variable, and divisions may introduce multi-intervals (Equation 6). Functions that introduce discontinuities, e.g., the step function, is another example. For instance, $\hat{c} \leftarrow \hat{a} \cdot \text{step}(x,\hat{b})$ will be in the range $\{0\} \cup [\underline{a},\overline{a}]$ if $x \in \hat{b}$. Note that if $0 \in \hat{a}$, we would merge the result and only propagate $[\underline{a},\overline{a}]$.

Many shader instructions grow the bounds, and as we start with often very loose compile-time bounds, it appears there would be little to gain from static bounds analysis. Fortunately, there is a long list of shader instructions with strict limits on their output:

instruction	**max output range**
$\text{sign}(x)$, $\text{step}(c,x)$	$\{0,1\}$
$\text{fract}(x)$, $\text{smoothstep}(c_1,c_2,x)$	$[0,1]$
$\text{clamp}(x,c_1,c_2)$	$[c_1,c_2]$
$\text{abs}(x)$, $\exp(x)$, $\text{pow}(x,y)$, $\text{sqrt}(x)$, $\text{rsqrt}(x)$, $\text{length}(\mathbf{x})$, $\text{acosh}(x)$	$[0,\infty]$
$\cosh(x)$	$[1,\infty]$
$\sin(x)$, $\cos(x)$, $\tanh(x)$, $\text{noise}(\mathbf{x})$	$[-1,1]$
$\arcsin(x)$, $\arctan(x)$	$[-\frac{\pi}{2},\frac{\pi}{2}]$
$\arccos(x)$	$[0,\pi]$
$\text{atan }2(y,x)$	$[-\pi,\pi]$
$\text{mod}(x,y)$	$[\min(y,0),\max(y,0)]$

This shows that even with *unbounded* shader inputs, we are likely to introduce useful bounds information during bounds propagation. Other examples of when an unbounded input can result in reduced bounds is division, e.g., $1/[1,\infty] \in [0,1]$, and squares, $[-\infty,\infty]^2 = [0,\infty]$.

Example The algorithm is best illustrated by an example. Applying the method to the program in Section 4.2 gives:

expression	**bounds**	**#inst**
$\hat{a}_0 \leftarrow \hat{a}$	$\hat{a}_0 \in [-\infty,\infty]$	
$\hat{b}_0 \leftarrow \hat{b}$	$\hat{b}_0 \in [-\infty,\infty]$	
$\hat{c}_0 \leftarrow \lvert\hat{a}_0\rvert$	$\hat{c}_0 \in [0,\infty]$	5
$\hat{t}_1 \leftarrow \hat{c}_0 \cdot \hat{b}_0$	$\hat{t}_1 \in [-\infty,\infty]$	6
$\hat{t}_2 \leftarrow \sqrt{\hat{t}_1}$	$\hat{t}_2 \in [0,\infty]$	2
`return` $\hat{t}_2$		
	sum:	13

In this case, the inputs $\hat{a}$ and $\hat{b}$ can lie anywhere in $[-\infty,\infty]$, as we have no additional metadata. At the $|\cdot|$ instruction, $\hat{c}_0$ is limited to positive numbers, which means the multiplication $\hat{c}_0 \cdot \hat{b}_0$ can be done using an optimized *positive* $\cdot$ *unknown* interval multiplication (6 vs. 10 instructions).

4.4 Valid Range Analysis

Some operations put restrictions on the range of valid arguments. For example, the argument to a square root must be positive, otherwise NaN is returned. Other examples are:

instruction	**valid input range**
$\log(\hat{a})$, $\text{sqrt}(\hat{a})$, $\text{rsqrt}(\hat{a})$, $\text{pow}(\hat{a},x)$	$\hat{a} \in [0,\infty]$
$\arcsin(\hat{a})$, $\arccos(\hat{a})$, $\text{atanh}(\hat{a})$	$\hat{a} \in [-1,1]$
$\text{acosh}(\hat{a})$	$\hat{a} \in [1,\infty]$

We can generate more efficient code if we at compile time make the *assumption* that all such instructions will receive valid input at runtime. This puts bounds on the valid range of variables, which can be propagated *backwards* through the code to put stricter bounds on the possible runtime range of variables. Applied to the same example as before:

expression	**bounds**	**valid range**	**#inst**
$\hat{a}_0 \leftarrow \hat{a}$	$\hat{a}_0 \in [-\infty,\infty]$	$\uparrow$ $\hat{a} \in [-\infty,\infty]$	
$\hat{b}_0 \leftarrow \hat{b}$	$\hat{b}_0 \in [-\infty,\infty]$	$\hat{b} \in [0,\infty]$	
$\hat{c}_0 \leftarrow \lvert\hat{a}_0\rvert$	$\hat{c}_0 \in [0,\infty]$	$\hat{a}_0 \in [-\infty,\infty]$	5
$\hat{t}_1 \leftarrow \hat{c}_0 \cdot \hat{b}_0$	$\hat{t}_1 \in [-\infty,\infty]$	$\hat{b}_0 \in [0,\infty]$	2
$\hat{t}_2 \leftarrow \sqrt{\hat{t}_1}$	$\hat{t}_2 \in [0,\infty]$	$\hat{t}_1 \in [0,\infty]$	2
`return` $\hat{t}_2$			
		sum:	9

Here the values in the *bounds* column have been determined using static bounds analysis in a first pass. We note that the square root puts a limit on its argument, $\hat{t}_1 \geq 0$, for the result to be valid. Tracking this backwards, we find that $\hat{b}_0$ must be positive for the result of $\hat{t}_1 \leftarrow \hat{c}_0 \cdot \hat{b}_0$ to be positive (as $\hat{c}_0$ is known to be positive). Hence, the interval multiplication can be replaced by an efficient 2 instruction version for *positive* $\cdot$ *positive* interval (Section 4.1).

It is important to point out that since we *assume* each instruction will produce valid results, the generated bounding shader will be undefined if executed for bounds violating this assumption. There is no way to detect this solely based on its output, so valid range analysis must only be used when the application knows it is safe for a given input. This can be ensured by first executing a bounding shader with this optimization disabled (i.e., using NaN checks). Also note that several iterations of static bounds analysis and valid range analysis may be necessary for the best results. After an initial pass, the refined bounds may be propagated further using static bounds analysis. This may in turn introduce possibilities for further improvements in a backwards pass. We iterate until convergence, which in our experience occurs within a few iterations, although pathological cases may exist.

4.5 Dynamic Bounds Assumptions

In general, if an input parameter's bounds are limited, the generated code will be more efficient. Several versions of a bounding shader can be generated with varying extents of input *bounds assumptions*. At runtime, we dynamically select the most restrictive (fastest) version that is valid for the current input. For instance, a restrictive version of the bounding shader may assume a texture access returns a value in $[0,1]$, and more general versions $[0,\infty]$ (e.g., for HDR textures) and $[-\infty,\infty]$. The appropriate version is then determined at runtime when a specific texture is bound to the shader. An alternative approach is to reactively compile specialized bounding shaders when an assumption is met. This is a more viable approach if the input parameter space is large.

4.6 Complex Control Flow

With interval arithmetic, whenever the control flow graph diverges, we must potentially execute both paths and at joints merge the results. This is easy for if-then-else branches (see Section 3.4), but more complex situations may lead to a tree of possible execution paths, where the result is the union of all possible outcomes. Previous work has thus been limited to statically unrollable loops [26].

We handle dynamic loops with interval conditions by, at runtime, accumulating the outcome of each iteration where the loop potentially terminates. Consider, for example, a loop on the form `do` $\hat{x} = \ldots$ `while`$(\hat{x} < c)$. When the condition is unambiguously true, i.e., $\overline{x} < c$, we loop as expected. Otherwise, we accumulate the result, $\hat{x}_{\cup} = [\min(\underline{x}, \underline{x}_{\cup}), \max(\overline{x}, \overline{x}_{\cup})]$, where $\hat{x}_{\cup}$ is initially empty, and continue looping until the branch condition is unambiguously false, i.e., $\underline{x} \geq c$. At that point, $\hat{x}_{\cup}$ holds conservative bounds for the union of all outcomes. At compile-time, the same technique is hard to apply due to often very wide compile-time bounds, and we resort to assuming that variables coming from back edges in the graph are unbounded. In some cases, it may help to iterate through loops a few times to refine the compile-time bounds.

Recursive functions, $\hat{y} = f(\hat{x})$, can be handled similarly in the bounds analysis by first assuming $\hat{x}$ and $\hat{y}$ are unbounded, and by replacing all recursive calls to f by the return value $\hat{y}$. This will give possibly refined compile-time bounds on $\hat{y}$, and the analysis can be iterated. Finally, the refined bounds are used to generate an optimized version of f, which includes real recursive calls. In practice, as noted previously [10], it is hard to guarantee runtime termination of interval code with complex control flow. One way is to manually add explicit termination criteria, e.g., maximum loop iterations or recursion depth, but how to do this automatically is an unsolved problem.

4.7 NaN and Infinity Issues

To guarantee correct results, a rigorous approach to detect and handle floating-point exceptions in the bounding shader must be used. The IEEE 754 standard specifies that all operations that produce a floating-point output must propagate NaNs, *except* for minimum and maximum operations (e.g., `minps` in SSE returns the source operand if one operand is NaN). This presents a problem, as we can construct bounding functions that give erroneous bounds [20]. Another way NaNs can be suppressed is through conditionals involving operands with NaN values.

We have identified three main approaches in the context of bounding shaders (sorted from strict to relaxed):

1. *Propagate NaNs.* If we make sure that NaNs are always propagated, we will get an indication that an invalid operation has occured. This requires explicit NaN detection for each min/max operation (at an overhead of 3 SSE instructions). We also have to ensure NaN values in branches are propagated.

2. *Suppress NaNs.* Assume the original shader never returns NaNs for any input generated by the rendering system – this is not unreasonable, as there is little point trying to bound a shader resulting in NaNs. Under this assumption, we may clamp the inputs to any instruction in the bounding shader that could generate NaNs, as we know the inputs will never fall outside the valid range in the original shader. It also requires us to redefine multiplication, so that $0 \cdot \pm\infty = 0$.

3. *Do not care about NaNs.* This may be the most reasonable approach in certain situations, as some applications can tolerate errors or do not depend on the bounds being strictly conservative for correctness.

In all these cases, if a bounding shader returns a NaN, it should be interpreted as *"bounds could not be computed"*. This may trigger further subdivision of the input bounds or a fallback on a more general technique.

4.8 Influence on Standard Compiler Optimizations

Some standard compiler optimizations are relatively more important for bounding shaders than for conventional shaders. For example, common-subexpression elimination is highly useful, as similar computations often are performed for the lower and upper bounds (see, e.g., Equation 3). In many cases only one of the shader outputs needs to bounded, or only the lower or upper bound of an interval computed. An example is depth culling [10], where only $z_{\min}$ is used. Hence, executable backward static slicing [32] (cf., dead code elimination) is very important.

As outlined in Section 3.3, we have to be careful not to introduce different rounding errors compared to the original shader in order to guarantee conservative results (up to machine precision). Therefore, we do *not* allow value-changing compiler transformations (i.e., optimizations that may change the floating-point precision) to be applied *after* the shader has been lifted to interval form. However, before the lifting step, such optimizations are allowed with respect to the bounding code, as shown in Figure 2. It is theoretically possible to perform value-changing optimizations also after separating the intermediate representations, but these would have to be performed in tandem on both IRs. We have left this possibility for future work.

5 Implementation

We have implemented a compiler prototype in C++, which uses metaprogramming to build a program graph; our implementation is based around a class `Var`, which implements all necessary shader functionality in overloaded operators and friend functions. Rather than computing a result, each function inserts a corresponding node in the program graph. Each assignment allocates a new temporary, which directly gives us an intermediate representation in SSA form. Macros are used to insert appropriate constructs for loops and conditionals. The graph nodes implement Equation 9, which is used to propagate bounds information through the graph. The type of each variable, i.e., scalar or interval, is also determined in this step. Last, each node outputs an optimized sequence of scalar instructions. To compile a shader, all variables are replaced by `Var` (or a wrapper to handle vectors); all inputs are assigned a type and initial bounds (default $\pm\infty$), and the shader is executed to generate the bounding shader.

Currently, two back ends have been implemented: one for SSE, which uses auto-vectorization to generate 4-wide data parallel bounding shaders, and one for HLSL. In our prototype, the back ends output C++/HLSL source code, which is finally passed through standard optimizing compilers. This setup has enabled us to quickly prototype and debug various optimizations, as it builds on existing compiler infrastructures to provide input parsing and optimized code generation. A production-quality bounding shader compiler is under development, but that is a much larger project.

6 Results

Here, we present results for a variety of bounding shaders. The effect of our optimizations are measured in terms of instruction count and execution speed for the generated kernels. All SSE/SSE2 code was compiled with Microsoft Visual Studio 2008 using `/O2 /fp:precise` (to ensure floating-point consistency), and executed on an Intel Core 2 Extreme QX9650. We will first look at BRDF bounding shaders for lightcuts and importance sampling purposes. Finally, we will show some GPU results.

6.1 Lightcuts

It has recently been demonstrated [26] that interval arithmetic is a viable solution to generate the bounding functions required by lightcuts [29, 28]. Let $s(\mathbf{x}, \omega_i) = f_r(\mathbf{x}, \omega_o, \omega_i) \cos\omega_i$ be a cosine-weighted BRDF, where $\mathbf{x}$ denotes a single shading point including all associated attributes. The upper bound at a single gather point for a cluster of lights, is given by $\bar{s}(\mathbf{x}, \hat{\omega}_i)$, i.e., only the light direction, ω_i, is an interval. Correspondingly, the upper bound for multiple gather points, e.g., to support depth-of-field and motion blur, is $\bar{s}(\hat{\mathbf{x}}, \hat{\omega}_i)$. Here all inputs are intervals.

Table 1 shows instruction counts and execution times using Intel SSE for the two types of bounding shaders, compared to the original point-sampled shaders, $s(\mathbf{x}, \omega_i)$, for a range of physically-based BRDFs. *Fractal* iteratively evaluates a fractal and interpolates between multiple Cook-Torrance lobes, and contains dynamic loops with both interval and scalar loop conditions. The other shaders do not contain complex control flow. With previous techniques, the average instruction counts are $3.1\times$ (single) and $4.8\times$ (multiple gather points) those of the original shaders. With our optimizations, 41.8–43.9% fewer instructions are generated,

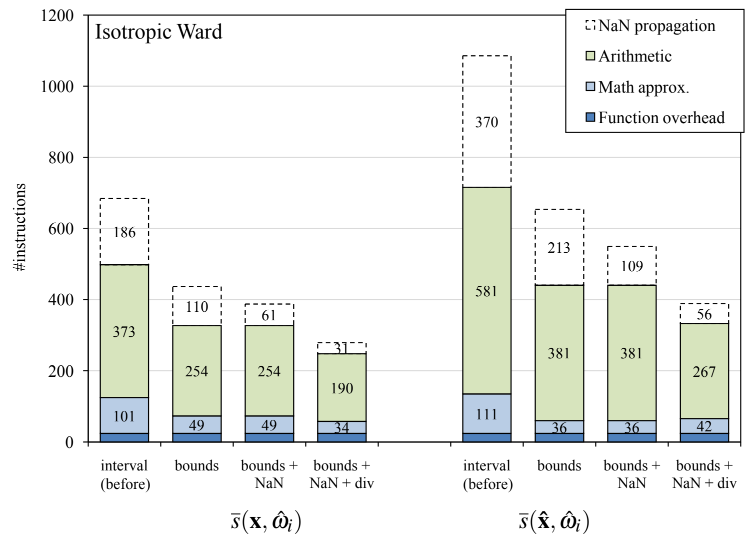
Shader	$s(\mathbf{x}, \omega_i)$	$\bar{s}(\mathbf{x}, \hat{\omega}_i)$		$\bar{s}(\hat{\mathbf{x}}, \hat{\omega}_i)$	
	#instr	before	after	before	after
Diffuse	20	2.8×	1.6×	5.8×	3.2×
Phong	149	2.1×	1.2×	4.5×	2.7×
IsoWard	151	3.3×	1.6×	4.7×	2.2×
Ward	174	3.4×	1.8×	5.3×	2.7×
Ashikhmin	486	2.8×	1.9×	3.8×	2.3×
Fractal	447	4.1×	2.6×	4.5×	3.0×
Average		3.1×	**1.8**×	4.8×	**2.7**×

Shader	$s(\mathbf{x}, \omega_i)$	$\bar{s}(\mathbf{x}, \hat{\omega}_i)$		$\bar{s}(\hat{\mathbf{x}}, \hat{\omega}_i)$	
	#cycles	before	after	before	after
Diffuse	20	32	26	58	36
Phong	182	311	205	559	330
IsoWard	192	445	273	510	317
Ward	218	495	325	623	420
Ashikhmin	514	1289	813	1535	898
Fractal	1297	4546	3155	4323	3036
Average		2.3×	**1.6**×	3.0×	**1.9**×

Table 1: *Instruction count (top) and execution time (bottom) compared to the original shaders using SSE for the two types of single-sided bounding shaders needed for lightcuts, before and after our optimizations are applied. The number of instructions is reduced by on average 41.8% and 43.9%, respectively. This saves 32.8–36.7% in clock cycles. All numbers exclude the overhead of NaN handling.*

which translates to a 32.8–36.7% saving in execution time. Note that the optimized bounding shaders compute the *exact* same bounds as before. It is also interesting to note that bounding shaders are in general faster than would be expected based on instruction count. This is likely due to more opportunities for register renaming and latency hiding.

The effect of different optimizations are shown in Figure 3 for two representative shaders: an isotropic version ($\alpha_x = \alpha_y$) of the Ward BRDF [30], and the complex anisotropic Ashikhmin model [2]. To get these results, we specified loose compile-time bounds on each shader's inputs; all values were assumed to be finite, and all material parameters such as diffuse and specular color, and shininess, were assumed to be non-negative. These values are fetched from the respective material channels (e.g., textures) in the shader setup, but the cost of this has not been included. Note that the use of textures instead of runtime shader constants, has no impact on our optimizations, as we only work with compile-time bounds.



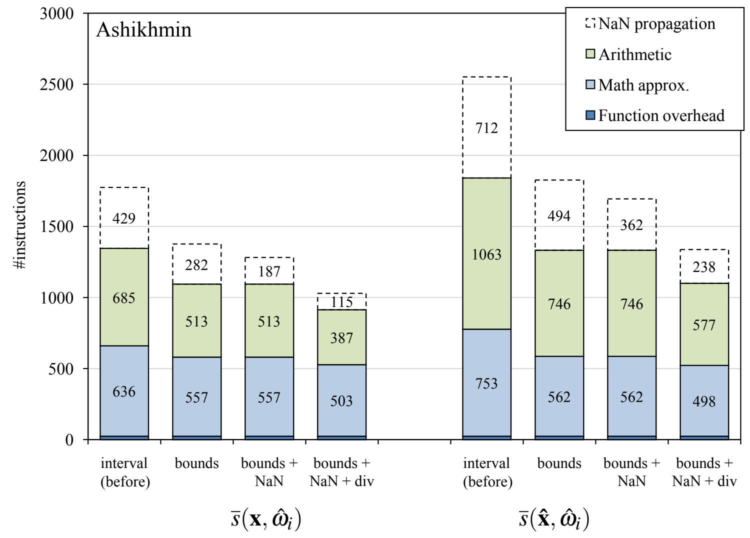


Figure 3: *The effect of our various compiler optimizations on two representative single-sided BRDF bounding shaders using SSE. The baseline is single-sided bounding shaders compiled using standard compiler optimizations (*interval*). Bounds analysis (*bounds*) significantly reduces the code size by tracking compile-time bounds and choosing optimized implementations of the interval operations. The overhead of robust NaN propagation is marked with a dashed box. This decreases when compile-time analysis of NaNs is enabled (*NaN*). Our optimizations targeted at optimizing divisions (*div*), shrink the compile-time bounds by tracking the signs of zeros (see Section 4.3), which further reduces the total instruction count.*

6.2 Importance Sampling

Importance sampling of arbitrary programmable shaders has traditionally been a difficult problem, and specialized methods have been developed for various reflectance models. By hierarchically evaluating $\bar{s}(\mathbf{x}, \hat{\omega}_i)$ over the hemisphere, a piecewise constant upper bound is created, which can be used as an importance function [26]. Note that $\bar{s}(\mathbf{x}, \hat{\omega}_i)$ is the same shader as in lightcuts (see Table 1 and Figure 3 for results).

A natural extension would be to use both bounds of s, i.e., lower and upper, in order to create a more accurate importance function. A step in this direction was taken by Rousselle et al. [22], although they used precomputed max and average trees rather than a min/max hierarchy. The function $\hat{s}(\mathbf{x}, \hat{\omega}_i)$ can be used to build an importance function for a single point, and $\hat{s}(\hat{\mathbf{x}}, \hat{\omega}_i)$ for a cluster of shading points. When both bounds are computed, it is harder for the compiler to remove entire computation chains, and we are mainly limited to detect cases where faster implementations of specific operations can be used, e.g., positive instead of general multiplication. Despite this, our optimizations reduce the instruction count by around 25% (see Table 2).

It is interesting to compare the instruction counts for single vs double-sided intervals in Tables 1 and 2. With standard compiler optimizations, going from double-sided to code specialized for single-sided intervals, only saves on average 5%. With our optimizations, the savings are ∼27%. For *Fractal*, the difference is smaller due to control flow making it harder to eliminate long computations chains, but there is still plenty of optimization potential locally.

Shader	$s(\mathbf{x}, \omega_i)$	$\hat{s}(\mathbf{x}, \hat{\omega}_i)$		$\hat{s}(\hat{\mathbf{x}}, \hat{\omega}_i)$	
	#instr	before	after	before	after
Diffuse	20	3.4×	2.8×	6.7×	4.9×
Phong	149	2.2×	2.1×	4.8×	4.0×
IsoWard	151	3.4×	2.4×	4.9×	3.2×
Ward	174	3.5×	2.5×	5.4×	3.7×
Ashikhmin	486	2.8×	2.3×	3.8×	3.0×
Fractal	447	4.2×	2.8×	4.6×	3.1×
Average		3.2×	**2.5**×	5.0×	**3.7**×

Table 2: *Instruction count compared to the original shaders using SSE for double-sided bounding shaders, before and after optimization. A prime application is importance sampling. The number of instructions is reduced by on average 23.8% and 27.1%, respectively. All numbers exclude the overhead of NaN propagation.*

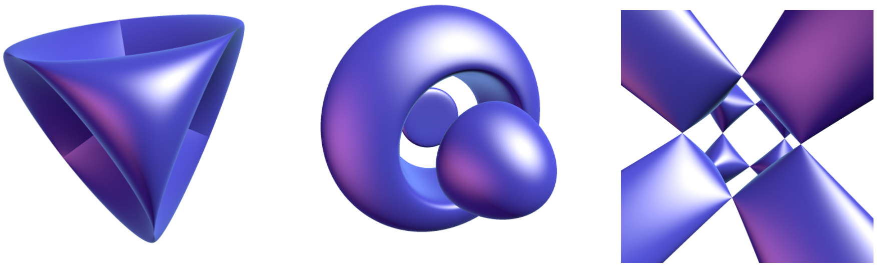


Figure 4: *Examples of implicit surfaces evaluated using interval arithmetic to find the first intersection along each ray.*

6.3 Implicit Surfaces

To measure the performance of optimized bounding code on the GPU, we have implemented a simple ray tracer for implicit surfaces, $f(x,y,z)=0$, that runs in the pixel shader. The algorithm hierarchically evaluates the surface equation using interval arithmetic to compute its *minimum*, in order to locate the first intersection along each ray using an existing algorithm [16]. Figure 4 shows renderings of three well-known implicit surfaces. The table below summarizes the number of generated shader instructions for the bounding kernels, and their execution times on an NVIDIA GTX285 GPU. The instructions are counted per-component in the case of vector operations. As can be seen, the reduction in instruction count is 20–35%, and in clock cycles 15–31%.

#instructions	**Steiner**	**Mitchell**	**Kummer**
original	13	16	20
interval (before)	74	51	64
interval (after)	48 (-35.1%)	41 (-19.6%)	44 (-31.3%)

#cycles	**Steiner**	**Mitchell**	**Kummer**
original	11.5	14.9	17.5
interval (before)	62.7	44.8	55.9
interval (after)	43.6 (-30.5%)	38.3 (-14.6%)	41.4 (-25.8%)

7 Conclusions and Future Work

We believe techniques for automatically extracting shader bounds will be increasingly important to close the gap between artistic control and fast rendering. In this paper, we have showed that bounding shaders based on interval arithmetic can be significantly optimized by performing bounds analysis and case selection at compile time. To the best of our knowledge, this is the first time data flow analysis has been used to optimize interval code generation. Although we have focused on computer graphics applications, many of the techniques would be directly applicable in many other fields. It is important to remember that our focus is entirely on compiler optimizations, and that the generated code computes the exact same bounds as before, only faster.

All of our results assume bounds are tracked individually, with no knowledge about the relationship between, e.g., the components of vectors. With higher-level information, e.g., specifying that vectors are normalized, it is possible to further optimize the code. The effect of this can be simulated by inserting clamps after dot products of normalized vectors, which saves an additional 13.3%–28.4% for the Ashikhmin shaders. It would be interesting to explore these kinds of optimizations and generalizations to higher-order arithmetics.

Acknowledgements

We acknowledge support from the Swedish Foundation for Strategic Research. In addition, Tomas Akenine-Möller is a *Royal Swedish Academy of Sciences Research Fellow* supported by a grant from the Knut and Alice Wallenberg Foundation. We are grateful to all reviewers for their valuable feedback.

Bibliography

[1] AKKAŞ, A., SCHULTE, M. J., AND STINE, J. E. Intrinsic Compiler Support for Interval Arithmetic. *Numerical Algorithms 37*, 1–4 (2004), 13–20.

[2] ASHIKHMIN, M., AND SHIRLEY, P. An Anisotropic Phong BRDF Model. *Journal of Graphics Tools, 5*, 2 (2000), 25–32.

[3] BERZ, M., AND HOFFSTÄTTER, G. Computation and Application of Taylor Polynomials with Interval Remainder Bounds. *Reliable Computing, 4* (1998), 83–97.

[4] COLLANGE, S., FLÓRES, J., AND DEFOUR, D. A GPU interval library based on Boost interval. In *RNC9, Real Numbers and Computers* (2008), pp. 61–72.

[5] COMBA, J. L. D., AND STOLFI, J. Affine Arithmetic and its applications to Computer Graphics. In *Proceedings of VI SIBGRAPI 1993* (1993), pp. 9–18.

[6] CYTRON, R., FERRANTE, J., ROSEN, B. K., WEGMAN, M. N., AND ZADECK, F. K. Efficiently Computing Static Single Assignment Form and the Control Dependence Graph. *ACM Transactions on Programming Languages and Systems, 13*, 4 (1991), 451–490.

[7] DUFF, T. Interval Arithmetic and Recursive Subdivision for Implicit Functions and Constructive Solid Geometry. In *Computer Graphics (Proceedings of SIGGRAPH 92)* (1992), vol. 26, pp. 131–138.

[8] GREENE, N., AND KASS, M. Error-Bounded Antialiased Rendering of Complex Environments. In *Proceedings of SIGGRAPH 1994* (1994), ACM, pp. 59–66.

[9] HANRAHAN, P., AND LAWSON, J. A Language for Shading and Lighting Calculations. In *Computer Graphics (Proceedings of SIGGRAPH 90)* (1990), vol. 24, pp. 289–298.

[10] HASSELGREN, J., AND AKENINE-MÖLLER, T. PCU: The Programmable Culling Unit. *ACM Transactions on Graphics, 26*, 3 (2007), 92:1–10.

[11] HASSELGREN, J., MUNKBERG, J., AND AKENINE-MÖLLER, T. Automatic Pre-Tessellation Culling. *ACM Transactions on Graphics, 28*, 2 (2009), 19:1–10.

[12] HEIDRICH, W., AND SEIDEL, H.-P. Ray-Tracing Procedural Displacement Shaders. In *Graphics Interface* (1998), pp. 8–16.

[13] HEIDRICH, W., SLUSALLEK, P., AND SEIDEL, H.-P. Sampling Procedural Shaders using Affine Arithmetic. In *Proceedings of SIGGRAPH 1998* (1998), vol. 17, ACM, pp. 158–176.

[14] HIJAZI, Y., HAGEN, H., HANSEN, C. D., AND JOY, K. I. Why Interval Arithmetic is so Useful. In *Visualization of Large and Unstructured Data Sets* (2007), pp. 148–163.

[15] KASS, M. CONDOR: Constraint-Based Dataflow. In *Computer Graphics (Proceedings of SIGGRAPH 92)* (1992), vol. 26, pp. 321–330.

[16] MITCHELL, D. P. Robust Ray Intersection with Interval Arithmetic. In *Proceedings on Graphics interface '90* (1990), pp. 68–74.

[17] MOORE, R. E. *Interval Analysis*. Prentice-Hall, 1966.

[18] MOULE, K., AND MCCOOL, M. D. Efficient Bounded Adaptive Tessellation of Displacement Maps. In *Graphics Interface* (2002), pp. 171–180.

[19] PARKER, S. G., BOULOS, S., BIGLER, J., AND ROBISON, A. RTSL: a Ray Tracing Shading Language. In *Proceedings of IEEE Symposium on Interactive Ray Tracing* (2007), pp. 149–160.

[20] POPOVA, E. D. Interval operations involving NaNs. *Reliable Computing, 2*, 2 (1996), 161–165.

[21] REDON, S., KHEDDAR, A., AND COQUILLART, S. Fast Continuous Collision Detection between Rigid Bodies. *Computer Graphics Forum (Proceedings of Eurographics), 21*, 3 (2002), 279–288.

[22] ROUSSELLE, F., CLARBERG, P., LEBLANC, L., OSTROMOUKHOV, V., AND POULIN, P. Efficient Product Sampling using Hierarchical Thresholding. *The Visual Computer (Proceedings of CGI 2008), 24*, 7-9 (2008), 465–474.

[23] SCHULTE, M. J., AND SWARTZLANDER, JR., E. E. A Family of Variable-Precision Interval Arithmetic Processors. *IEEE Transactions on Computers, 49*, 5 (2000), 387–397.

[24] SCHULTE, M. J., ZELOV, V., AKKAS, A., AND BURLEY, J. C. The Interval-Enhanced GNU Fortran Compiler. *Reliable Computing 5*, 3 (1999), 311–322.

[25] SNYDER, J. M. Interval Analysis for Computer Graphics. In *Computer Graphics (Proceedings of SIGGRAPH 92)* (1992), vol. 26, pp. 121–130.

[26] VELÁZQUEZ-ARMENDÁRIZ, E., ZHAO, S., HAŠAN, M., WALTER, B., AND BALA, K. Automatic Bounding of Programmable Shaders for Efficient Global Illumination. *ACM Transactions on Graphics, 28*, 5 (2009), 142:1–9.

[27] ŽILINSKAS, J. Comparison of Packages for Interval Arithmetic. *Informatica 16*, 1 (2005), 145–154.

[28] WALTER, B., ARBREE, A., BALA, K., AND GREENBERG, D. P. Multidimensional Lightcuts. *ACM Transactions on Graphics, 25*, 3 (2006), 1081–1088.

[29] WALTER, B., FERNANDEZ, S., ARBREE, A., BALA, K., DONIKIAN, M., AND GREENBERG, D. P. Lightcuts: A Scalable Approach to Illumination. *ACM Transactions on Graphics, 24*, 3 (2005), 1098–1107.

[30] WARD, G. J. Measuring and Modeling Anisotropic Reflection. *Computer Graphics (Proceedings of ACM SIGGRAPH), 26*, 2 (1992), 265–272.

[31] WEGMAN, M. N., AND ZADECK, F. K. Constant Propagation with Conditional Branches. *ACM Transactions on Programming Languages and Systems, 13*, 2 (1991), 181–210.

[32] XU, B., QIAN, J., ZHANG, X., WU, Z., AND CHEN, L. A Brief Survey of Program Slicing. *SIGSOFT Software Engineering Notes, 30*, 2 (2005), 1–36.

[33] ZHANG, X., REDON, S., LEE, M., AND KIM, Y. J. Continuous Collision Detection for Articulated Models using Taylor Models and Temporal Culling. *ACM Transactions on Graphics, 26*, 3 (2007), 15:1–10.

Hierarchical Stochastic Motion Blur Rasterization

Jacob Munkberg* Petrik Clarberg* Jon Hasselgren* Robert Toth*
Masamichi Sugihara* Tomas Akenine-Möller*†

*Intel Corporation †Lund University

ABSTRACT

We present a hierarchical traversal algorithm for stochastic rasterization of motion blur, which efficiently reduces the number of inside tests needed to resolve spatio-temporal visibility. Our method is based on novel tile against moving primitive tests that also provide temporal bounds for the overlap. The algorithm works entirely in homogeneous coordinates, supports MSAA, facilitates efficient hierarchical spatio-temporal occlusion culling, and handles typical game workloads with widely varying triangle sizes. Furthermore, we use high-quality sampling patterns based on digital nets, and present a novel reordering that allows efficient procedural generation with good anti-aliasing properties. Finally, we evaluate a set of hierarchical motion blur rasterization algorithms in terms of both depth buffer bandwidth, shading efficiency, and arithmetic complexity.

High Performance Graphics, pp. 107–118, 2011.

1 Introduction

At the heart of every rendering engine there is some form of *visibility* computations. A more advanced algorithm allows effects such as motion blur and depth of field to be rendered by a more elaborate camera model. Depth of field helps to direct the viewer's attention, and motion blur reduces temporal aliasing, so that lower frame rates can be used. Both these effects are also highly desired in the field of real-time graphics.

While an incredible amount of research and engineering effort has been spent on perfecting and fine-tuning the algorithms and the corresponding hardware units for rasterizing static triangles [11, 29, 28, 23, 22], the same is far from true for rasterization of motion-blurred geometry. However, there has been increased research activity in this field [8, 2, 10, 24, 6], but much remains to be done before the relative efficiency of rasterizing triangles with blur effects is close to that of static triangle rasterization. To be able to add correct motion blur to current and future games, one of our goals is to support efficient rendering of motion blur with mixed sizes of the triangles, i.e., both large triangles and smaller triangles, generated by, e.g., tessellation.

To that end, we present what we believe is the first *hierarchical* rasterization algorithm for motion-blurred triangles. We strive for an algorithm that extends current real-time GPU pipelines, while retaining many of its important features, such as per-tile occlusion culling, mixed triangle sizes, shading after visibility, and multi-sampling anti-aliasing (MSAA).

Our contributions are:

- A hierarchical algorithm for motion blurred triangle rasterization, including a low-cost tile vs moving triangle overlap test that returns a conservative time interval of overlap.
- Modification of an existing hardware-friendly sampling pattern for use in motion blur rasterization with high-quality anti-aliasing. We present an efficient algorithm for computing the samples within a time interval on the fly.
- Detailed performance evaluation of several different motion blur rasterization algorithms in terms of arithmetic intensity, memory bandwidth usage, and shading efficiency.

We hope that our new algorithms will advance the field of motion blur rasterization so that in the near future, fixed-function rasterization units will have support for such effects.

2 Related Work

Efficient rendering of motion blur has been a long-standing problem in computer graphics. Existing solutions often rely on approximate post-processing based methods or stochastic ray tracing [9]. We will not go into detail on these methods, and instead focus on rasterization-based methods for correct motion blur that can be integrated into future hardware GPU pipelines.

A brute-force technique is to draw the scene at N different times and average the result using accumulation buffering [20, 16]. The resulting strobing artifacts can be replaced by noise by using stochastic rasterization [8]. Here, a bounding box around the blurred triangle is traversed, and all samples are tested against the primitive displaced according to the samples' times. This becomes inefficient when the bounding box is large compared to the primitive. The screen space area of the traversed region can be reduced using either an oriented bounding box (OBB) in 2D homogeneous space [2], or the convex hull in screen space [24]. Existing (two-dimensional) hierarchical rasterization methods can be leveraged to efficiently traverse these bounds, but all temporal samples still have to be tested. This becomes expensive with large motion. In contrast, our algorithm derives temporal bounds per tile to cull samples.

Fatahalian et al. [10] improve the situation for stochastic micropolygon rasterization by partitioning the time domain into multiple intervals (initially proposed by Pixar), or by using interleaved sampling [17] with a fixed number of sample times. Both methods rasterize the primitive independently for each time/interval, which generates samples in an incoherent order, i.e., sparse in screen space. For a REYES pipeline with shading at the vertex level, this is not a problem, but applied to a graphics pipeline with shading at the fragment level, it makes reusing shading over multiple samples (MSAA) difficult. Per-tile occlusion culling also becomes substantially more expensive. Furthermore, each triangle has to be setup multiple times. In contrast, our algorithm uses a coherent screen space traversal order, which facilitates MSAA and efficient occlusion culling. Figure 1 shows the spatio-temporal coverage of each algorithm.

Although a hard problem, analytical determination of visibility has been explored. Most recently, Gribel et al. [14] presented a method for analytical motion blur rasterization where the samples' temporal overlaps with a moving primitive are analytically determined and stored in linked lists per pixel. Our work is similar in that we analytically determine conservative time bounds, but we do this for entire tiles of pixels and the generated samples are stored in a traditional multi-sampled render target. The use of a tiled traversal with temporal bounds allows us to quickly reject samples.

Hierarchical occlusion culling is critical for achieving good performance in modern GPUs by early determining if a tile is entirely occluded (z_{max}-culling) [25] or entirely visible (z_{min}-culling) [3]. However, motion blur makes culling using a traditional hierarchical z-buffer [13] less efficient. By storing multiple temporal depth

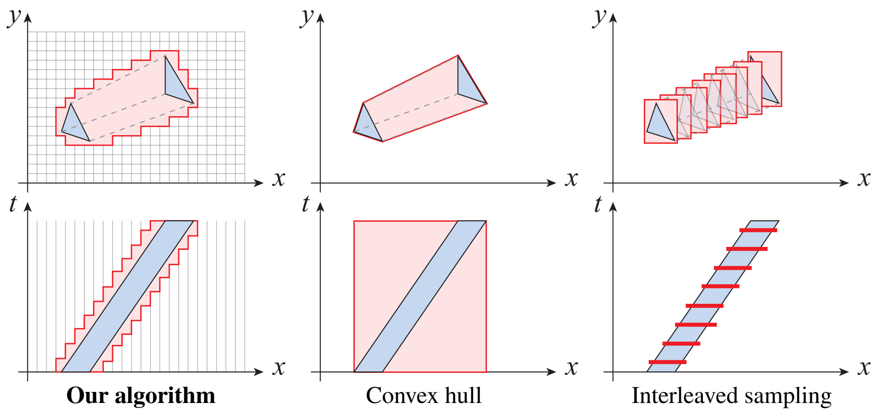


***Figure 1:** The three-dimensional sampling space, (x,y,t), traversed with different methods for stochastic motion blur rasterization. Sample-in-triangle inside tests are performed for all samples within the red regions. Our algorithm, based on novel hierarchical tile tests with temporal overlap computations, significantly reduces the amount of inside tests compared to using the convex hull in screen space [24]. With interleaved sampling [17] the samples are restricted to a fixed number of pre-defined times.*

values (*tz-slice*) [2], or a full temporal pyramid of depth values (*tz-pyramid*) [4] per node, efficiency can be improved. Our rasterization order, i.e., one tile at the time, together with conservative temporal bounds, makes the use of these occlusion culling techniques efficient and straightforward.

3 Overview

Our hierarchical motion blur traversal algorithm works entirely in two-dimensional homogeneous space (2DH) to robustly handle moving triangles crossing the $z = 0$ plane. The first step is conservative backface culling [26] and *temporal* view frustum culling. Each triangle vertex moves along a line in 2DH, and by finding the intersection of these three lines with each frustum plane, we obtain the time interval, $[t_s, t_e]$, when the moving triangle is inside the view frustum.

The traversal algorithm for a triangle can be summarized as follows, where a *tile* is a rectangular block of pixels:

```
BBOX = Compute moving triangle bounding box
for each tile in BBOX [hierarchical traversal]
    TIME = Compute time interval of overlap
    Occlusion culling of tile in TIME
    for each sample in tile in TIME
        Test sample against primitive
```

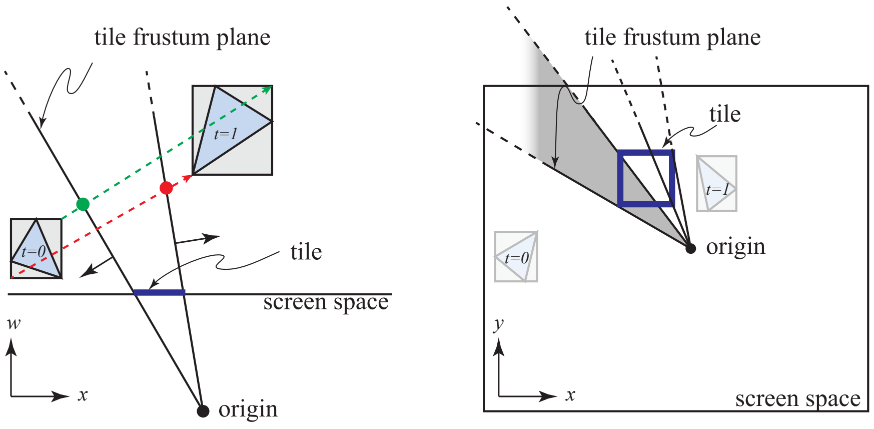


Figure 2: *A moving triangle is enclosed by an AABB in 2DH with linear per-vertex motion. The left figure shows the xw plane, with indicators when the moving AABB enters (green dot) and exits (red dot) the tile frustum. The right illustration shows the screen space view of this example.*

To compute a screen space bounding box around the moving triangle, we bound the screen space projections of the six vertices (the triangle vertices at t_s and t_e) if the moving triangle is entirely in front of the $z = 0$ plane, and revert to the conservative bounding approach presented by McGuire et al. [24] otherwise.

In Section 4, we introduce the tile vs moving triangle tests (line 3), which form the necessary basis for our hierarchical traversal algorithm. The output for a certain tile is either *trivial reject*, or a conservative time interval where overlap possibly occurs. The computation of per-tile time bounds greatly reduces the number of temporal samples that are tested for fast moving primitives, as large subsets of the spatio-temporal samples within a tile can be discarded. It also makes hierarchical occlusion culling simple and efficient. For each tile, we only test the primitive against occlusion information in the relevant time interval (line 4).

As with all stochastic methods, the statistical distribution of the sample points has a large impact on the result. Stochastic rasterization has the additional constraints that the samples must be consistent from primitive to primitive (otherwise cracks may appear), and extremely fast to generate as the sampling takes place in the inner loop of the rasterizer. We have chosen to base our samples on binary (t,m,s)-nets [27] for their extensive stratification properties. Section 5 introduces a remapping of a known pattern to provide a temporally ordered sequence that is extremely inexpensive to compute in hardware. Last, we discuss temporal filtering for high-quality shading of the generated samples in Section 6, followed by implementation details and a thorough evaluation in Sections 7 and 8, respectively.

4 Tile Tests with Temporal Bounds

It is well-known that efficient rasterization of static geometry can be obtained by hierarchical testing of a tile of pixels against a triangle [23]. This is done by overlap testing the bounding box of the triangle against the tile, and also testing each triangle edge against the tile [1]. We will extend this to moving geometry, where the bounding box becomes a moving box, and the triangle edges sweep through space.

More specifically, we derive tight bounds for the overlap between a screen space tile and a triangle with linear per-vertex motion in three dimensions. Each vertex, $\mathbf{p}_i$, moves from the position $\mathbf{q}_i$ at $t=0$, to $\mathbf{r}_i$ at $t=1$, that is: $\mathbf{p}_i(t)=(1-t)\mathbf{q}_i+t\mathbf{r}_i$. All computations are performed in 2D homogeneous coordinates, with a vertex defined as $\mathbf{p}=(p_x,p_y,p_w)$. The main idea is to find a conservative time interval, $\hat{t}_{tot}=[\underline{t}_{tot},\bar{t}_{tot}]$, in which the moving triangle overlaps the tile. Per-sample tests are then done *only* for samples whose times belong to the time interval. In the following, we first describe how a tile is tested against a moving box, and then how a tile is tested against a moving triangle edge.

4.1 Frustum Plane–Moving AABB Overlap

We create a moving AABB in 2DH by bounding the triangle at $t=0$ and $t=1$ and interpolating between the two AABBs. This is an approximation to the true swept bounding box, but it is guaranteed to be conservative at all times. Based on a tile on screen, we then setup four frustum planes that are aligned to the sides of the tile. Each frustum plane, π_i, passes through the origin and is defined by its plane equation $\mathbf{n}_i\cdot\mathbf{p}=0$, where $\mathbf{n}_i$ is the plane's normal. A point $\mathbf{p}$ is outside the plane if $\mathbf{n}_i\cdot\mathbf{p}>0$. If a point is inside all planes, then it is inside the frustum. For static geometry, it is sufficient to test the corner of an enclosing AABB that is farthest in the negative direction (n-vertex) relative to π_i [12], in order to determine if the box is entirely in the positive half-space. The sign bits of the plane's normal, $\mathbf{n}_i$, directly decides which corner is the n-vertex. We note that the same holds for linearly moving bounding boxes, as the orientations of the frustum planes remain constant. Figure 2 shows an example of a moving triangle, whose moving AABB intersects with two tile frustum planes.

The point of intersection in time between the moving n-vertex and a plane π_i is given by:

$$\mathbf{n}_i\cdot((1-t)\mathbf{q}_n+t\mathbf{r_n})=0 \quad\Longleftrightarrow\quad t=\frac{\mathbf{n}_i\cdot\mathbf{q}_n}{\mathbf{n}_i\cdot(\mathbf{q}_n-\mathbf{r}_n)}, \tag{1}$$

where $(1-t)\mathbf{q}_n+t\mathbf{r_n}$ is the moving n-vertex for π_i. Let $d_0=\mathbf{n}_i\cdot\mathbf{q}_n$ and $d_1=\mathbf{n}_i\cdot\mathbf{r}_n$.

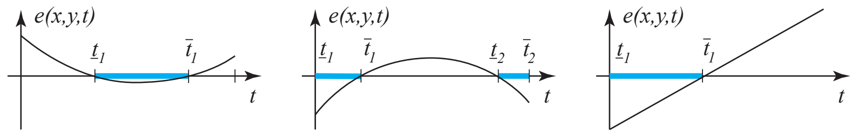


Figure 3: *Edge equations as functions of t for a specific (x,y) location. We are interested in finding the time intervals where e < 0 (highlighted in turquoise).*

The temporal overlap, $\hat{t}_i$, between the AABB and the plane π_i is given by:

$$\hat{t}_i = \begin{cases} \emptyset & \text{if } d_0, d_1 > 0 & \text{both outside} \\ [\max(0,t),1] & \text{else if } d_0 > 0 & \mathbf{q}_n \text{ outside} \\ [0,\min(1,t)] & \text{else if } d_1 > 0 & \mathbf{r}_n \text{ outside} \\ [0,1] & \text{otherwise} & \text{both inside,} \end{cases} \tag{2}$$

where t is computed using Equation 1. The temporal overlap between all the tile planes and the moving AABB is given by $\hat{t}_{box} = \bigcap_i \hat{t}_i$, where we can test for fine-grained trivial rejection after each iteration of the loop over the four frustum planes, π_i.

4.2 Moving Triangle Edge Tests

For triangles with linear vertex motion in three dimensions, each triangle edge sweeps out a bilinear patch. The corresponding *time-dependent* edge functions are quadratic in t. To determine if a screen space tile overlaps the swept triangle, we evaluate the triangle's three edge equations for the four corners of the tile and check if any corner is inside all three edges. In that case, we again determine a time interval in which the triangle conservatively overlaps the tile to reduce the number of per-sample inside tests.

The edge equation for a triangle with linear per-vertex motion can be written as follows [2]:

$$e(x,y,t) = \mathbf{n}(t) \cdot \mathbf{s} = (\mathbf{f}t^2 + \mathbf{g}t + \mathbf{h}) \cdot \mathbf{s}, \tag{3}$$

where $\mathbf{s} = (x,y,1)$ is a sample position in screen space. For a given $\mathbf{s}$, we have a maximum of two roots to $e(x,y,t) = 0$, and up to two time intervals per edge, $\hat{t}_i = [\underline{t},\bar{t}]$, where the sample is inside ($e < 0$). Some examples are given in Figure 3.

Handling near-linear edge motion in an efficient and robust way is extremely important, because often a large portion of the triangles in a scene will have close to linear motion. In these cases, directly finding the roots of $e=0$ involves a division with a very small quadratic coefficient, which may lead to numerical instability. Therefore, we have devised a robust test that performs well when the edge equations are near-linear, and that is increasingly conservative when the quadratic term grows. We bound the quadratic edge function's projection within a screen space tile using lines with constant slopes. This *linearization* of the overlap test greatly

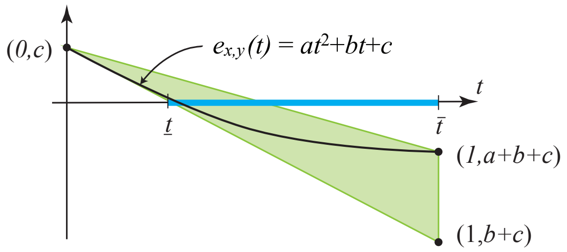


Figure 4: *The lower bound of a quadratic polynomial from Equation 4 is bounded by a linear approximation around* $t = 0$.

reduces the computations needed. The edge function (Equation 3) is linearized according to:

$$e(x,y,t) = \mathbf{n}(t)\cdot\mathbf{s} \geq \mathbf{o}\cdot\mathbf{s} + \gamma t, \quad \forall \mathbf{s} \in S, \tag{4}$$

where S is a region in screen space, e.g., the bounding box of the swept triangle. With $\mathbf{n}(t) = \mathbf{f}t^2 + \mathbf{g}t + \mathbf{h}$, we rewrite the edge function for a certain screen space position, $\mathbf{s}$, as [14]:

$$\mathbf{n}(t)\cdot\mathbf{s} = \mathbf{f}\cdot\mathbf{s}\,t^2 + \mathbf{g}\cdot\mathbf{s}\,t + \mathbf{h}\cdot\mathbf{s} = at^2 + bt + c, \tag{5}$$

where $a = \mathbf{f}\cdot\mathbf{s}$, $b = \mathbf{g}\cdot\mathbf{s}$ and $c = \mathbf{h}\cdot\mathbf{s}$. It can be shown that this curve is included in the triangle given by the points $(0,c),(1,b+c)$ and $(1,a+b+c)$ as seen in Figure 4. We search for a lower linear bound of the curve's slope, which is given by:

$$\min(b, a+b) = \min(\mathbf{g}\cdot\mathbf{s}, (\mathbf{f}+\mathbf{g})\cdot\mathbf{s}). \tag{6}$$

A conservative minimal slope, γ, for all $\mathbf{s} \in S$, is given by:

$$\gamma = \min_{\mathbf{s}\in S}(\mathbf{g}\cdot\mathbf{s}, (\mathbf{f}+\mathbf{g})\cdot\mathbf{s}). \tag{7}$$

If we set $\mathbf{o} = \mathbf{n}(0) = \mathbf{h}$, we have obtained a linearized version of the edge equation according to Equation 4. This linear representation is conservative even if the edge function has a large quadratic term. Note that γ can be computed in the triangle setup using the moving triangle's screen space AABB as S. For more accurate bounding, γ can be recomputed on a coarse tile level, using the tile extents as S.

Given the linearization, the tile vs moving edge test is considerably simplified. By looking at the signs of the xy-components of $\mathbf{o}$, we only need to test one tile corner, $\mathbf{s}$. A conservative time for the intersection of the triangle edge and the tile is given by:

$$\mathbf{o}\cdot\mathbf{s} + \gamma t = 0 \quad \Longleftrightarrow \quad t = -\frac{\mathbf{o}\cdot\mathbf{s}}{\gamma}. \tag{8}$$

Note that $-\frac{\mathbf{o}}{\gamma}$ can be precomputed, so the time of overlap for a tile only costs 2 `MADD` per edge. Depending on the sign of γ, the tile's temporal overlap, $\hat{t}_k$, with edge e_k is defined as:

$$\hat{t}_k = \begin{cases} [\max(0,t),1] & \text{if } \gamma < 0, \\ [0,\min(1,t)] & \text{otherwise,} \end{cases} \tag{9}$$

where t is computed according to Equation 8. Once all three triangle edges have been tested, the temporal overlap between the tile and the swept triangle is given by $\hat{t}_{edges} = \bigcap_k \hat{t}_k$, where we can test for fine-grained trivial rejection after each iteration of the loop over edges (e_k). The final interval is the intersection of the intervals from both the moving box test (Section 4.1) and the moving edge test, i.e., $\hat{t}_{tot} = \hat{t}_{box} \bigcap \hat{t}_{edges}$.

Given $\hat{t}_{tot}$, we first perform spatio-temporal occlusion culling, and for the surviving tiles and time intervals, we proceed with individual sample-in-triangle inside tests. The following section describes the computation of our sampling positions, (x,y,t).

5 Sampling

In a motion blur rasterizer, each pixel is associated with a number of fixed (x,y,t) samples. The samples must be the same from triangle to triangle to get correct visibility, but vary randomly from pixel to pixel to reduce temporal and spatial aliasing. Stochastic sampling introduces noise, and it is well-known that sample points with good statistical properties, e.g., large minimum distance, provide a good balance between noise and aliasing. For us, it is also desirable that the samples project to a good distribution in (x,y) for high-quality anti-aliasing of static primitives.

Our application imposes a number of further constraints. First, sampling needs to be fast and use minimal storage, as it is performed at the core of the rasterizer. Second, each pixel should have the same number of samples to simplify hardware design. Additionally, since our tile tests compute the temporal overlap, $\hat{t}_{tot}$, it is important to be able to quickly find the relevant samples for a tile, i.e., the samples should be ordered in t. These requirements severely restrict our options. For example, Poisson disk points are not guaranteed to project to a good distribution in two dimensions, and it may be hard to guarantee a fixed number of samples.

For these reasons, we have chosen to work with sampling distributions that are realizations of digital (t,m,s)-nets [27]. Although often used for quasi-Monte Carlo integration in offline rendering [19], we believe samples based on digital nets are ideal also for motion blur rasterization due to their extensive stratification properties and ease of construction. Next, we will give a brief introduction (see Niederreiter's work [27] for more details), and introduce a novel variation of a known method for generating three-dimensional samples with good properties. Our samples are ordered in t and have a good spatial distribution when projected to screen space.

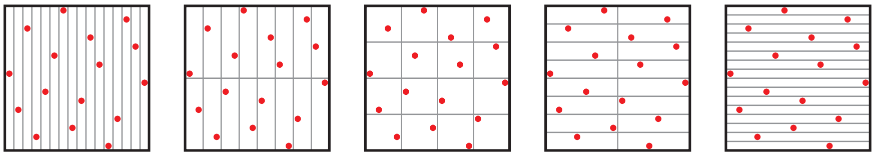


Figure 5: *Example of a* $(0,4,2)$*-net in base 2. The five figures illustrate all elementary intervals with area* $b^{t-m} = 2^{-4}$ *over the unit square, where each one has exactly* $b^t = 2^0 = 1$ *samples. We present a method for procedural construction of three-dimensional* (t,m,s)*-nets with properties targeted at motion blur rasterization.*

Definition A set of b^m s-dimensional points $\mathbf{x}_j = (x_j^{(1)}, \ldots, x_j^{(s)})$ is a (t,m,s)-net in base b if every *elementary interval* of volume b^{t-m} contains exactly b^t points, where $b \geq 2$ and $0 \leq t \leq m$ are integers. The elementary intervals are discrete subintervals of space:

$$E = \prod_{i=1}^{s} \left[\frac{a_i}{b^{l_i}}, \frac{a_i+1}{b^{l_i}} \right) \subseteq [0,1)^s, \tag{10}$$

where $0 \leq l_i$ and $0 \leq a_i < b^{l_i}$ are integers. The volume constraint gives $\sum_{i=1}^{s} l_i = m - t$. An example in two dimensions is shown in Figure 5. In our case, $s = 3$ and we work exclusively with binary numbers ($b = 2$) for efficiency reasons. Lower t value (referred to as "quality") gives better stratification, i.e., fewer points per stratum. Hence, we are interested in $(0,m,3)$-nets in base 2, which have 2^m points and exactly one point per elementary interval. This property ensures that samples near in time are spatially far apart, and vice versa, which is important to minimize noise.

A *digital* (t,m,s)-net can be defined using a set of *generator matrices* $C_1, \ldots, C_s$ over a finite field $\mathbb{F}_q$, where q is prime [27]. $\mathbb{F}_q$ consists of elements numbered $\{0, \ldots, q-1\}$, and all arithmetic operations are performed modulo q. Since we work in base 2, $q = 2$ and the C_i matrices are binary $m \times m$ matrices. The ith component of the jth point is given by:

$$x_j^{(i)} = \left(2^{-1}, \ldots, 2^{-m}\right) \left[C_i \begin{pmatrix} d_0(j) \\ \vdots \\ d_{m-1}(j) \end{pmatrix} \right] \in [0,1), \tag{11}$$

where $d_k(j)$ are the bits of j, $j \in \{0, \ldots, 2^m - 1\}$, with d_0 being the least significant bit. The matrix-vector product $C_i \left(d_0(j) \cdots d_{m-1}(j)\right)^T$ is performed in $\mathbb{F}_2$.

Our Method Three-dimensional digital nets with good 2D projections are not very well explored. Grünschloß and Keller [15] propose one method based on reordering of the Sobol' sequence, where the first two dimensions are the Larcher-Pillichshammer (LP) points [19], which have a good spatial distribution. The first

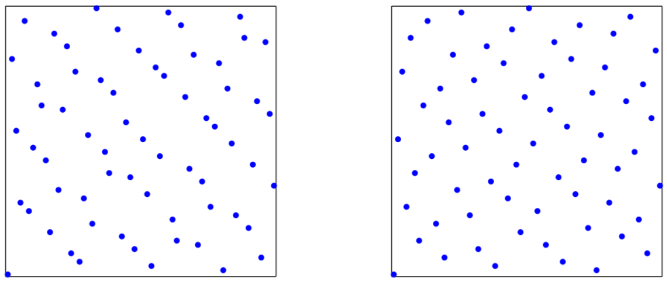


Figure 6: *The original samples [15] are ordered in the first component. The projection onto the other two dimensions are shown on the left for $m = 6$. After our permutation, the non-time dimensions project to the Larcher-Pillichshammer points (right), which give better spatial anti-aliasing.*

component is sequentially ordered, i.e., $x_j^{(1)} = \frac{j}{2^m}$. Unfortunately, we have observed that the projection onto the other two dimensions is not as well-distributed, exhibiting a structure of diagonal lines. See Figure 6 (left). This leads to inferior spatial anti-aliasing, as our algorithm uses the ordered dimension as time.

To address this, we propose a *permuted* construction that is ordered in t, while still projecting to the LP-points in the remaining two dimensions, as shown in Figure 6 (right). Our samples are given by the generator matrices (see Appendix A for details):

$$C_1' = \left(\binom{m+1-l}{m+1-k} \mod 2 \right)_{k,l=1}^{m}, \tag{12}$$

$$C_2' = \left(\binom{m-l}{k-1} \mod 2 \right)_{k,l=1}^{m} \quad \text{and} \quad C_3' = \begin{pmatrix} 0 & & 1 \\ & \iddots & \\ 1 & & 0 \end{pmatrix}.$$

These binary matrices are visualized in Figure 7. The figure compares our matrices to the original matrices of Grünschloß and Keller, denoted C_1, C_2, C_3. Note that our modified matrices compute the *same* set of points as before, but the points are generated in a different order. The order is important, as our input is a sequential index in time, and the remaining two dimensions are used as the spatial sample position. We want the projection to screen space to be as good as possible for high-quality spatial anti-aliasing. This is especially important for static and slowly moving primitives, where the user gets plenty of time to study the quality. Note that reordering the points by permuting the matrices is not the same as just assigning the dimensions differently.

Although the matrices look deterring, they make an efficient procedural computation of samples possible. Addition equals XOR in $\mathbb{F}_2$, so entire columns of the matrix-vector product in Equation 11 can be added using single XOR operations. Additionally, we omit the leftmost vector multiplication and view the result as the

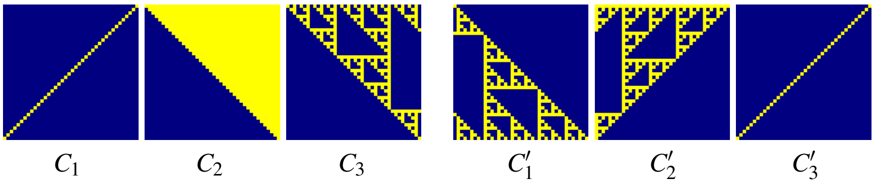


Figure 7: *The right three images show examples of our generator matrices for $m = 40$. The first two components are given by shifted and reflected Sierpiński triangles. These two matrices generate the same points as C_1 and C_2, but permuted into an order that better suits our purposes (ordered in t).*

digits of a fixed-point representation. The following C-function computes the xy-coordinates of the point with sequential index j, i.e., sample time $t = j/2^m \in [0,1)$, for any $m < 32$. All coordinates are integers in $\{0,\dots,2^m-1\}$.

```
void GetXY(uint j, const uint m, uint& x, uint& y)
{
   x = y = 0;
   uint c1 = 0x3, c2 = 0x1 << (m-1);
   for (j <<= 32-m; j != 0; j <<= 1)
   {
      if (j & 1u<<31)    // Add matrix columns (XOR)
      {
         x ^= c1 >> 1;
         y ^= c2;
      }
      c1 ^= c1 << 1;     // Update matrix columns
      c2 ^= c2 >> 1;
   }
}
```

The algorithm examines j one bit at the time, starting at its high bit $m-1$, and adds up columns of C'_1 and C'_2. The matrices are computed on the fly, using only bit shifts and XOR operations. In hardware, the above algorithm can be implemented using a very small number of gates. During rasterization, we generate samples for $t \in \hat{t}_{tot}$ at the finest hierarchical level in the traversal. For example, with 4×4 pixel tiles at 16 samples/pixel, we have 256 samples, so $m = 8$ and the samples' xy-coordinates are interpreted as 2.6 bits fixed-point numbers (i.e., the top two bit gives the pixel position, and the lower six bits the sub-pixel placement). Throughout the paper, we also quantize time to 64 discrete values (see Section 7), although this is not a necessity.

To avoid a repeating sample pattern, we apply *random digit scrambling* [19] to the generated xy-coordinates, as it gives good results at an extremely low cost. Conceptually, the sampling domain is hierarchically split in half along each spatial dimension, and the two halves randomly permuted. The same permutations are

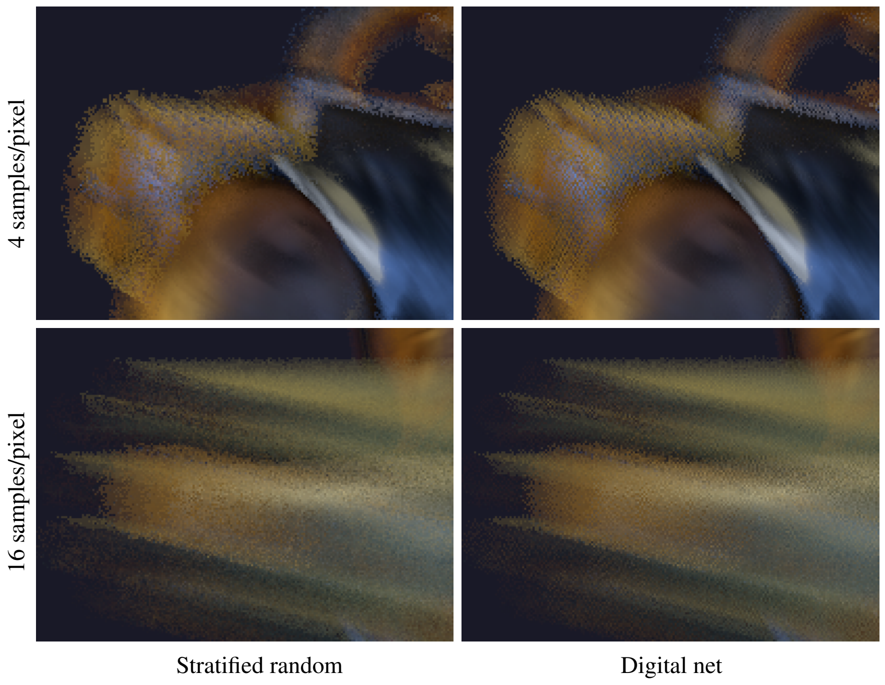


Figure 8: *Comparison between stratified random sampling (left) and sampling based on digital nets (right), at two different sampling rates. The regularity in the digital net based pattern gives a noticeably smoother appearance, while avoiding obvious aliasing.*

applied to all samples within a tile. In base 2, this operation can be performed by a bitwise XOR between the *xy*-coordinates and two independent random bit vectors. We compute the random vectors based on a hash of the tile position, which ensures consistency from frame to frame. The regular structure of the scrambled digital net gives low noise without any obvious aliasing artifacts. Note that random digit scrambling also largely preserves the properties of the projected samples (Figure 6), which is an important aspect. To increase the randomness, a sub-pixel jittering can be applied, but we have not found that to be necessary. Figure 8 shows the rendering quality compared to traditional stratified random sampling, where *xy* and *t* have been independently stratified per pixel.

Discussion An alternative to using procedurally generated samples is to store a lookup table of samples, ordered in t. This allows for more flexibility, but incurs an additional hardware cost. Inspired by Grünschloß and Keller [15], we have experimented with randomized permutation-based search for $(0,m,3)$-nets with larger minimum point distance than the above construction, but the results of this has been left for future work.

6 Shading

A core feature of our algorithm is that we visit a particular pixel at most *once* for a certain triangle. This is similar to McGuire et al.'s work [24], but different from interleaved and interval rasterization [17, 10], where a pixel may be visited many times for the same triangle. Using our traversal order, it is therefore possible to use multisampling anti-aliasing (MSAA) strategies with temporal filtering [21, 24], which is not feasible for interleaved and interval rasterization since they slice the time dimension. As will be seen in Section 8, multisampling can give a considerable reduction in shader cost, which is a big advantage when implementing the traversal algorithm in an existing GPU pipeline. It should be noted that decoupled shading solutions [30, 7] show more promise to further reduce the shading cost. However, the multisampling approach can be implemented on current hardware with no, or very small, modifications as shown by McGuire et al [24], which makes it a good first step towards a graphics hardware solution fully supporting motion blur. Although our shading system is very similar to previous work, we describe it briefly since being able to use multisampling is currently an important feature of our algorithm.

We base our temporal shader filtering on the work of Loviscach [21] and McGuire et al. [24]. The basic idea is to use anisotropic texture filtering to integrate textures over the motion footprint by modifying the derivatives. By integrating over time in the shader, it is possible to sample the shader only once per pixel, and write the result to all covered samples.

We assume that the only varying shader inputs are the barycentric coordinates $u(x,y,t), v(x,y,t)$, and disregard from explicit shader input variables representing the sample time. For texture filtering, we need to estimate the texture footprint. The integration domain is given by a fourth order rational function in t, but we choose to make the same approximations as Loviscach [21], and use his approach for perturbing the screen space texture gradient axes to account for the temporal derivative. McGuire et al. [24] present an approximation where the screen space gradients use a fixed axis in texture space. However, this method suffers from severe aliasing for some view directions, as can be seen in Figure 9.

We compute $\frac{\partial u}{\partial x}$, $\frac{\partial u}{\partial y}$ and $\frac{\partial u}{\partial t}$ by finite differences (same for the partial derivatives of v). For each quad of 2×2 pixels with at least one sample covered, we evaluate the shader at five points: each of the four pixel centers at $t = 0.5$ for the quad, in order to compute x and y derivatives using finite differences, and one additional point for one of the samples at $t = 1$ to compute a per-quad approximation of the temporal derivative. For a more fine-grained temporal derivative, one can shade the entire quad at two distinct times and compute per-pixel temporal derivatives, or shade on a per sample/pixel basis, which would shade four points to compute per-pixel $\frac{\partial u}{\partial x}$, $\frac{\partial u}{\partial y}$, and $\frac{\partial u}{\partial t}$ derivatives (and the partial derivatives of v). We use the quad approximation in all our tests. The shading approach integrates very well into existing pinhole camera rendering systems with MSAA support with relatively modest modifications to the hardware.

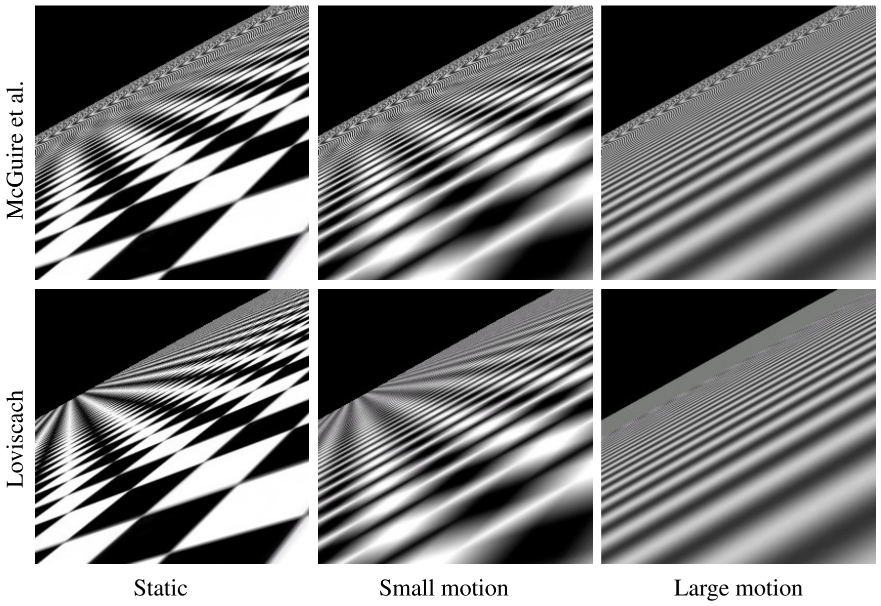


Figure 9: *Comparison between the filter kernel approximations proposed by McGuire et al. (top) and Loviscach (bottom). The results are rendered on an NVIDIA GeForce 290 GTX with* $16\times$ *anisotropic filtering. Note the severe aliasing in the top row. This is due to over-emphasizing the motion contribution, and the approximation of the screen space filter kernel as a fixed* $(1,1)$*-vector.*

It should be noted that our approach for computing derivatives may lead to shading samples that lie outside the triangle. This can cause shading artifacts, but we have not found them to be significant in our test scenes. McGuire et al. [24] used a different approach and picked the last covered time sample as the shading sample, but reprojected to $t = 0$. This has other implications such as overblurring due to too large filter kernels. Working out a strategy for correct shader filtering and derivative computations with stochastic sampling is an interesting problem that deserves a thorough evaluation. However, we leave that for future work.

With interleaved rasterization [17, 10] in a pipeline with shading after visibility, there are no adjacent screen space samples with the same sample time. Without introducing large hardware changes (e.g., a shader cache) the spatial derivatives can be computed in two ways. One option is to compute derivatives per sample by executing the shader at three spatial positions. The other option is to compute derivatives using finite differences from four nearby samples with the same time, while taking the perturbed sample positions into account. This method produces coarser derivatives, as the samples with the same time are typically separated by two or more pixels. In either case, as we shade at all sample times, the temporal derivatives are less important and can be ignored without much loss in image quality. We chose to use the first alternative for the statistics presented in this paper, as the quality more closely matches that of our derivatives. Furthermore, coarse derivatives will degrade the quality even for static scenes compared to current graphics API specifications, which we want to avoid.

7 Implementation

Hierarchical Rasterizers To evaluate our algorithm, we have implemented four rasterizers called CONVEX, OUR, INTERVAL and HIERARCHICAL INTERLEAVE in a software rasterization pipeline that can execute DX11 traces, but lacks a general shading system. In order to evaluate shading and sampling quality, we have also designed a GPU rasterizer which performs stochastic rasterization in a pixel shader, similar to McGuire et al.'s work [24].

CONVEX (based on McGuire et al. [24]) traverses all tiles within an AABB overlapping the screen space convex hull (*CH*) of the moving triangle. The original paper uses a two-step algorithm that triangulates the convex hull and rasterizes stochastically in the pixel shader (to run on current GPUs). In our software implementation, each tile is tested against the *CH* edges, and if it overlaps, all spatio-temporal samples within the tile are tested against the triangle. Both approaches perform hierarchical rasterization, but the latter may be more efficient in hardware, as a tile is visited only once.

OUR algorithm is similar, but uses the tile tests with temporal bounds introduced in Section 4 instead of the *CH* edges. The temporal bounds significantly reduce the number of samples that must be tested.

INTERVAL is based on Pixar's motion blur algorithm (described by Fatahalian et al. [10]). The main difference compared to CONVEX and OUR is the iteration order, where INTERVAL has an outer loop over sample times instead of over tiles. This is the only algorithm that cannot be easily extended to hierarchical rasterization. We still use a tiled traversal approach for the sake of hierarchical depth culling, but we have no test for trivially rejecting a non-overlapping tile.

HIERARCHICAL INTERLEAVE is a hierarchical 2DH version of Fatahalian et al.'s interleaved rasterizer [10], where the triangle is rasterized with an interleaved sampling pattern. Like INTERVAL, this algorithm has the outer loop over sample times rather than tiles. Note that for mixed triangle sizes, adding a hierarchical test to Fatahalian et al.'s [10] interleaved rasterizer improves performance a lot, which is to be expected since the target of the original paper was micropolygon rendering.

All algorithms perform backface culling [26] and view-frustum culling (including temporal bounds). In all tests, we use the same high-quality interleaved sampling patterns, described in Section 5, with 64 fixed times. The reason for this is that the edge functions can be pre-computed for 64 times in the triangle setup and reused over the triangle, which gives substantially reduced costs for our test scenes. The inside test, which uses 2DH edge equations, is therefore identical for all four

	CONVEX	OUR
Iteration order	Tiles	Tiles
Screen space bbox	BBox of *CH*	Bound tri over $[0,1]$
Tile test	*CH* edges	Swept tri
Occl. & sample test	$t \in [0,1]$	$t \in \hat{t}_{tot}$
Shading	MSAA	MSAA
Sampling pattern	Arbitrary	Ordered in t

	INTERVAL	HIER. INTERLEAVE
Iteration order	Sample times	Sample times
Screen space bbox	Bound tri over $[t_i, t_{i+k}]$	Bound tri at t_i
Tile test	None	Tri edges at t_i
Occl. & sample test	$t = [t_i, t_{i+k}]$	$t = t_i \ \forall i$
Shading	Supersampling	Supersampling
Sampling pattern	Ordered in t	Interleaved

Table 1: *Algorithm comparison. CH denotes the screen space convex hull of the moving triangle. For* INTERVAL*, we divide the N unique sample times in N/k intervals.*

algorithms. The traversal strategies are summarized in Table 1.

Using a sample pattern with a fixed set of sample times, there is an amount of motion where a fast moving triangle has no spatial overlap between adjacent times, t_i and t_{i+1}. Figure 10 illustrates this. In this case, there is little temporal coherence to exploit, i.e., each tile has only one or a few covered samples. Therefore, we propose a fallback to HIERARCHICAL INTERLEAVE traversal whenever the individual bounding boxes no longer overlap. We use a simple heuristic to decide when this occurs. The dimensions of the bounding boxes at $t=0$ and $t=1$ are $w_0 \times h_0$ and $w_1 \times h_1$, respectively, and the swept bounding box $w_s \times h_s$. If we rasterize at N unique times, HIERARCHICAL INTERLEAVE traversal is chosen whenever $\min(w_0, w_1) < w_s/N$ or $\min(h_0, h_1) < h_s/N$. We use this fallback in all our measurements of OUR, CONVEX, and INTERVAL. It is primarily activated for very small, fast moving triangles.

At $16\times$ multisampling, we use three hierarchical levels with 16×16, 8×8, and 4×4 pixel tiles for all algorithms except HIERARCHICAL INTERLEAVE (which uses 32×32, 16×16 and 8×8 pixel tiles). Depth culling is performed on the coarsest and finest levels, and trivial reject or time overlap test are performed on the two finer levels. These configurations were determined by extensive evaluation of both arithmetic cost and bandwidth usage. We use coarser tile levels for HIERARCHICAL INTERLEAVE since the tile tests are executed for each sample time t_i. For HIERARCHICAL INTERLEAVE, a single tile test at 4×4 pixel tiles culls at most 4 samples with $t = t_i$. In contrast, up to 256 samples with $t \in [0,1]$ can be culled for 4×4 pixel tiles with our tile tests.

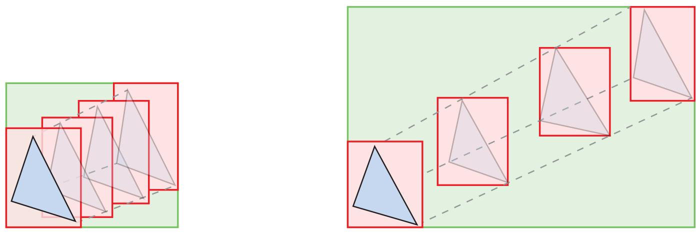


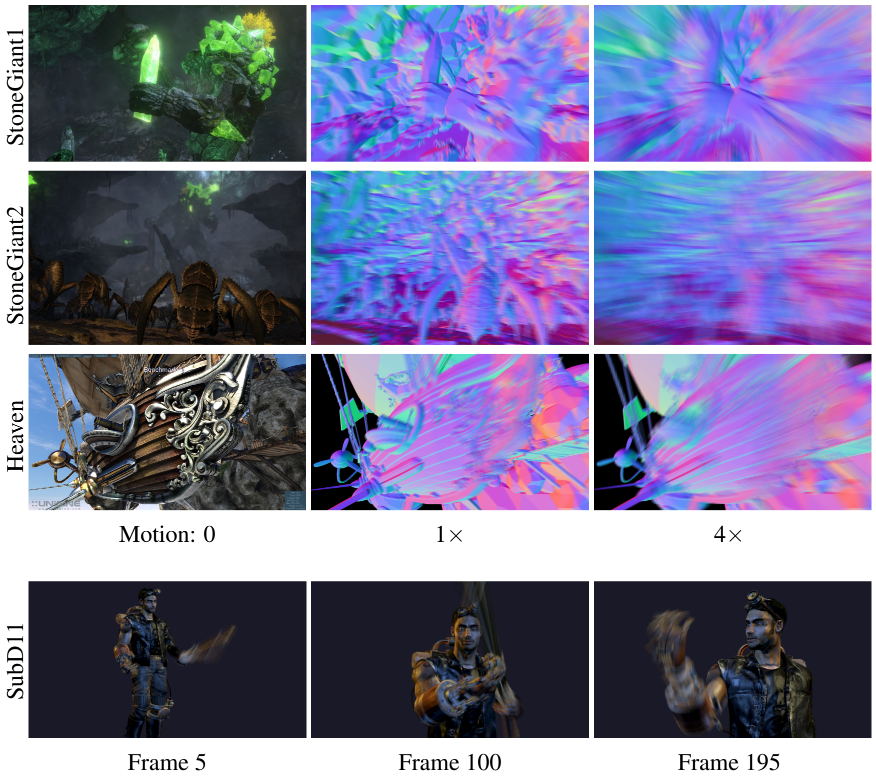
Figure 10: *Two moving triangles are rasterized at four fixed times for illustrative purposes. With little motion, the individual bounding boxes (red) overlap, which makes a screen space traversal in the swept bbox (green) preferable. At higher motion, the bounding boxes separate and* HIERARCHICAL INTERLEAVE *traversal is more efficient.*

Time-Dependent Occlusion Culling A traditional *z-max* buffer stores one conservative z-max value for each screen space tile. We use a *time-dependent* z-max buffer [2, 4], which contains multiple temporal depth values per tile for increased culling efficiency in the presence of motion blur. For better efficiency, our algorithm uses the tile's temporal overlap, $\hat{t}_{tot}$ (Section 4), to avoid performing occlusion queries for time intervals when the triangle does not overlap. Our simulator uses a fully associative cache backing the depth and z-max buffers with 64 bytes cache lines. In the following, we denote the number of different sample times represented in one cache line as s. A corresponding z-max value then represent s times over a screen space tile whose spatial extents is proportional to $1/s$ to make it fit in the cache line. A coarse z-max buffer is either constructed for each of the 64 times, and hence $s = 1$, or for a group of consecutive times (representing a smaller screen space area), where $s > 1$. In all cases, we have one z-max value for each cache line of sample depths.

The temporal coherence may be further exploited by constructing a temporal hierarchy [4]. We explored this with a two-level spatio-temporal cache-backed hierarchy, but were not able to reduce bandwidth usage. This is partly due to the cache line grouping of values – large chunks of geometry must be successfully culled in order to avoid reading z-max data – and partly due to the added cost of keeping another level of z-max data in the cache. Also note that our tests are not using any depth compression. However, a two-level temporal hierarchy decreased the arithmetic cost by around 5–10%, and we use it for all algorithms.

8 Results

Our test scenes are presented in Figure 11 and include various types of motion, triangles sizes, and geometry distributions. All results were generated using our



Scene	Triangles	Resolution	Motion
StoneGiant1	0.26M	1280 × 720	Camera translation
StoneGiant2	4.1M	1920 × 1200	Camera rotation
Heaven	2.4M	1920 × 1080	Camera translation
SubD11	0.16M	1920 × 1080	Keyframed animation

Figure 11: *Our test scenes with two frames from the StoneGiant demo (courtesy of BitSquid), one from the Heaven 2 demo (courtesy of Unigine Corp.), and the SubD11 animation from Microsoft DX SDK (June 2010). Motion blur has been added to all scenes.*

own simulation framework described in Section 7 with 16 samples per pixel. We present results for the depth buffer bandwidth, the number of shader executions, and the number of arithmetic operations required for rasterization. In most charts, we have also included a standard hierarchical rasterizer without motion blur. This is referred to as STATIC.

Depth Buffer Bandwidth Reducing memory bandwidth usage is incredibly important, and therefore, we start with a study on depth buffer bandwidth usage. Figure 12 (top) shows the depth buffer memory bandwidth usage from cache misses (including both sample depths and z-max values) when the number of sample times per cache line, s, is varied. Grouping more times into a cache line increases the penalty of larger motion, but lowers the bandwidth usage for parts of the scene moving slowly. For our algorithm, $s = 4$ is preferable for a wide range of motion. We use this number for all measurements for Our, Convex, and Interval since they have similar access patterns. Note, however, that our algorithm performs slightly fewer hierarchical occlusion queries than Convex since we use the results from the tile test to avoid testing some time intervals. More aggressive temporal grouping ($s = 16$) only pays off for very small motion, while no temporal grouping ($s = 1$) is more efficient for extreme motion. Hierarchical Interleave

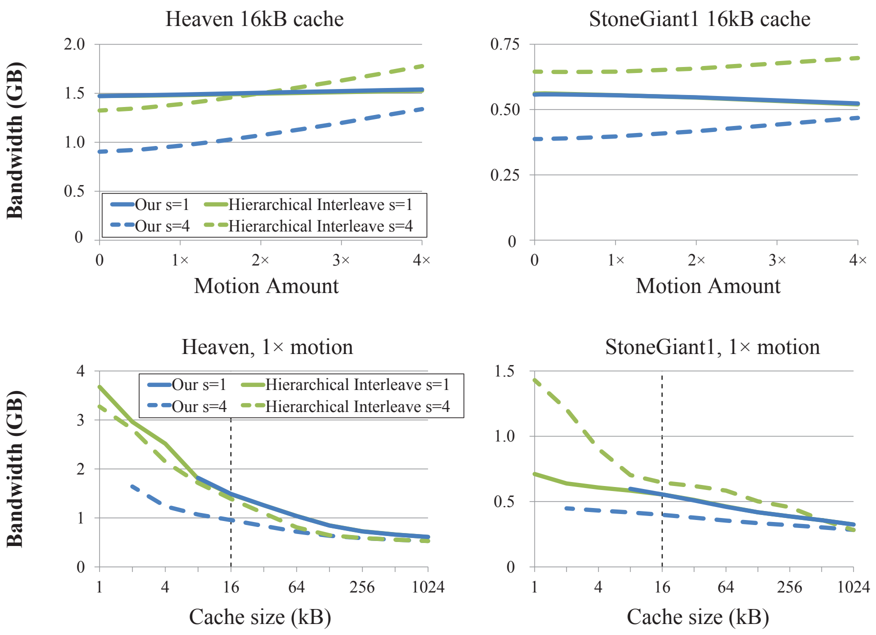


Figure 12: Bandwidth usage for z and z-max at 16 *samples per pixel. Left: varying amount of motion with a 16kB depth cache. A higher temporal z-max resolution (lower s) scales better with increasing motion, but has a higher constant cost.* $s = 4$ *is a suitable choice for* Our, *as the crossover occurs at extreme motion (outside the graph). Right: varying cache size with fixed amount of motion. The minimum cache requirement for our algorithm is to accommodate all N time layers and the z-max storage. Our algorithm with* $s = 4$ *scales well to decently small cache sizes.* Convex *is not shown as it behaves similar to our algorithm, with the only difference that more z-max queries are performed.*

scales differently, and the benefit for grouping sample times into the same cache line is low even for static scenes. This is an effect of the traversal order, where the triangle is fully traversed for one sample time before continuing with the next. For larger triangles, this may lead to eviction of the first cache line before starting traversal for the next sample time. Therefore, we use the $s = 1$ framebuffer layout for the HIERARCHICAL INTERLEAVE algorithm.

To determine a suitable cache size, we ran the bandwidth measurements with various cache sizes as presented in Figure 12 (bottom). Our algorithm needs a minimum cache size of 2kB (and in the case of $s = 1$, it needs 8kB) to avoid a 100% miss rate. However, above this minimum, it scales better than HIERARCHICAL INTERLEAVE traversal for decreasing cache sizes. In fact, HIERARCHICAL INTERLEAVE does not level out until the cache becomes very large (128kB–1MB). For the remaining comparisons, we use a 16kB cache for all algorithms. This corresponds to approximately 256 fully covered pixels worth of data, not counting z-max storage.

As shown in Figure 12 and 15–17 (top row), the depth buffer bandwidth (including both sample depths and z-max values) to external memory is relatively constant for the HIERARCHICAL INTERLEAVE traversal order with increasing motion, while it is increasing somewhat for OUR, CONVEX, and INTERVAL. Our algorithm becomes less efficient than HIERARCHICAL INTERLEAVE at some point. However, extreme motion is needed to reach the break-even point, and our algorithm consistently outperforms the competing algorithms except for the difficult StoneGiant2 scene with extreme motion ($>4\times$). In Figure 13 (top), we see that our algorithm uses less bandwidth than HIERARCHICAL INTERLEAVE for the SubD11 animation. In fact, in most frames, OUR uses only about 50% of that of HIERARCHICAL INTERLEAVE, even though the scene is animated at a low frame rate (24 fps) and contains frames with relatively large motion. Our algorithm also uses slightly less bandwidth than CONVEX since our tile tests with temporal bounds enable more efficient occlusion culling. This is particularly noticeable in the frames with largest motion. The depth buffer bandwidth of INTERVAL is similar to our algorithm but with one important difference. Since INTERVAL does not have a trivial reject test we need to perform depth culling for all tiles overlapping the triangle bounding box. This is not significant for small triangles or for scenes with only motion in the x- or y-directions. However, for SubD11 which contains large sliver triangles, this leads to many unnecessary depth culling queries and increases bandwidth significantly.

Shading Efficiency Figure 13 (bottom) shows the number of shader executions per frame for the SubD11 animation. For OUR and CONVEX, we use multisampling, while HIERARCHICAL INTERLEAVE traversal has to resort to supersampling due to the traversal order (i.e., for each time, there is only one sample per tile in the interleaved sampling pattern). For INTERVAL, we use 16 time intervals. This implies that within each time interval, there is one sample per pixel in the interleaved sampling pattern. As can be seen, the benefit of multisampling is very

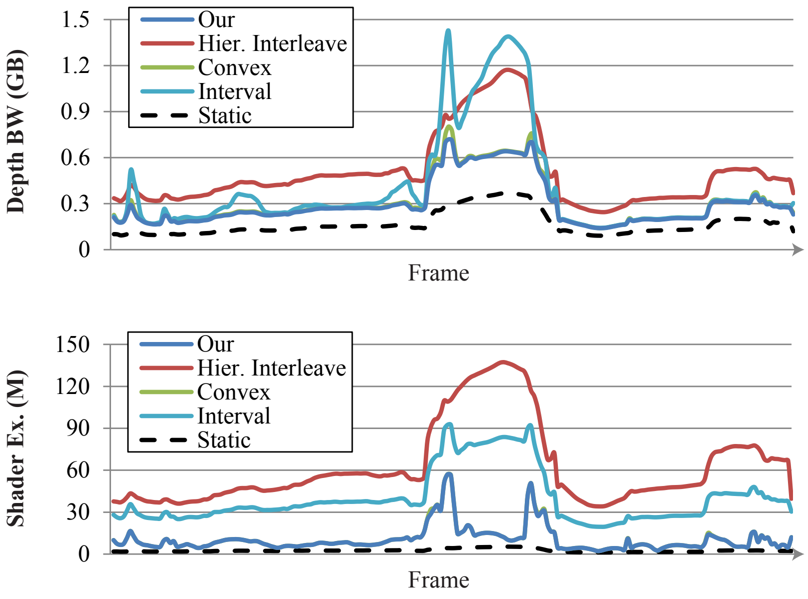


Figure 13: *Depth buffer bandwidth (top) and number of shader executions (bottom) for the SubD11 animation.*

high for SubD11. Also, INTERVAL is more efficient than HIERARCHICAL INTERLEAVE since shading is computed once per interval ($16\times$) rather than per sample time ($64\times$), and quad-fragment shading and finite differences can be used.

At extreme motion relative to the primitive size, we effectively revert to supersampling. This happens at $>4\times$ motion for the StoneGiant2 scene, which is highly tessellated (see the middle row in Figures 15–17. It is questionable whether this extreme motion will be usable for real-time rendering ($>$ 60 fps). In all cases, the shading overhead compared to STATIC is often quite high. Given these observations, we conclude that better long term solutions for shading may be cache or object-space based approaches, as discussed in Section 6. However, we believe that a multisampling approach for motion blur is likely to be adopted by hardware vendors as an intermediate step towards a pipeline with decoupled shading.

Rasterization Cost We have instrumented our code with cost estimations for the critical stages of the rasterization algorithm: triangle setup, tile test, sample test, and interpolation setup. We only account for the cost of the more complex operations such as ADD, MUL, RCP, etc., and disregard from the cost of bit-twiddling and control logic. Therefore, our results should not be seen as absolute costs for an implementation, but rather demonstrate the general trend of the algorithms and how they relate. We intentionally do not use sample test efficiency [10] as an effi-

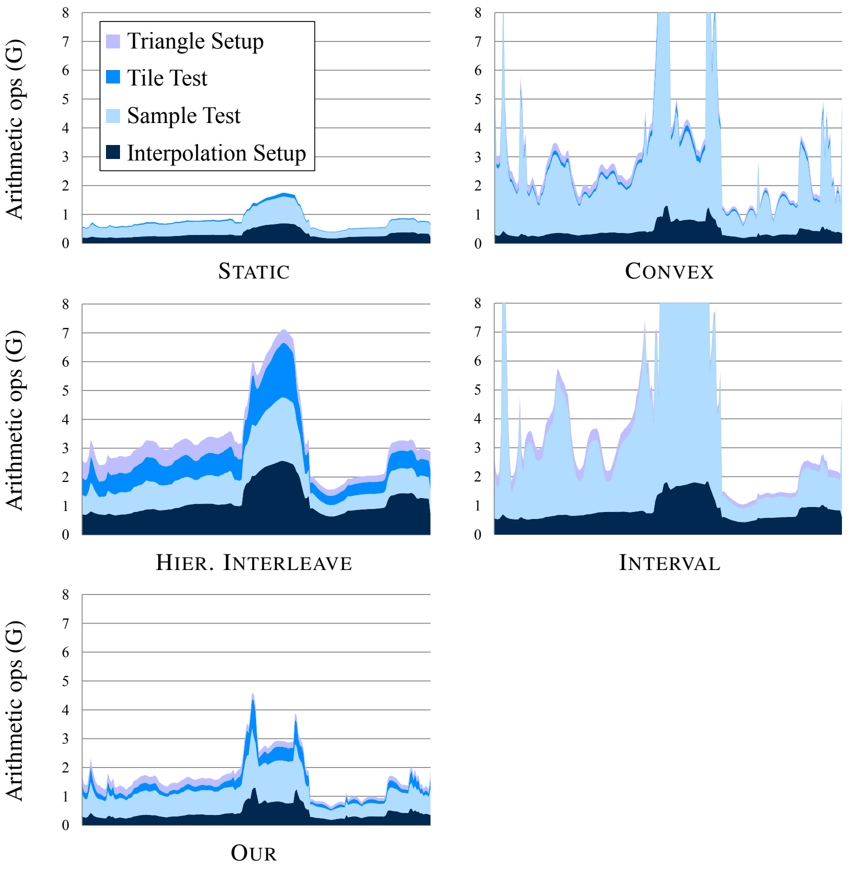


Figure 14: *Arithmetic cost breakdown for different traversal algorithms on the* SUBD11 *animation at* 16× *samples per pixel. Note that the* CONVEX *traversal has substantially higher number of sample tests, due to the lack of temporal bounds per tile. Due to the lack of hierarchical testing,* INTERVAL *and* INTERLEAVE *also suffer from excessive sample testing. Hence, we only use the improved* HIERARCHICAL INTERLEAVE *in our measurements.*

	Static BBox	Box	Edge	**Box+Edge**	
SubD11	8.5 (58)	6.2 (32)	1.8 (4.9)	1.7 (4.5)	$\times 10^9$
Heaven	160 (200)	17 (21)	17 (22)	14 (18)	$\times 10^9$

Table 2: *Efficiency of our tile tests in terms of average total arithmetic cost per frame (maximum in parenthesis). The combination of the two tests gives a cost efficient and robust tile test.*

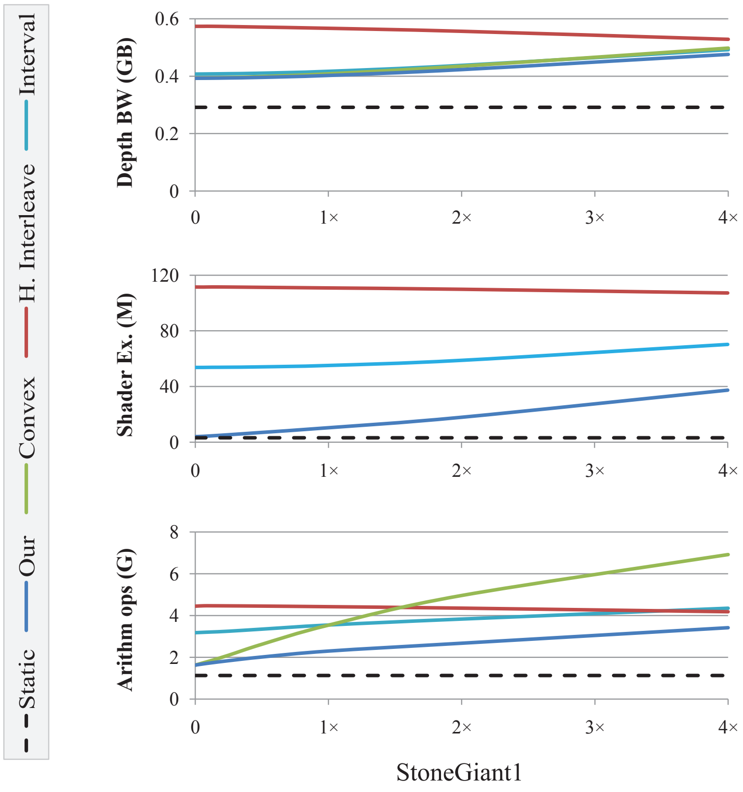


Figure 15: *The total depth buffer bandwidth, #shader executions, and arithmetic cost for the StoneGiant1 scene, as functions of the motion amount, where* 1× *denotes a modest motion amount and* 4× *represents extreme motion. See screenshots in Figure 11. Note that* CONVEX *has the same number of shader executions as* OUR

ciency measure, since it only includes a part of the rasterization cost. For example, sample test efficiency would benefit of using as small tiles as possible, but in practice, the best tradeoff is found with medium sized tiles where the sum of the tile test cost and sample test cost is minimized.

In Figure 14, we show a breakdown of the costs for the different stages of the rasterizer for the SubD11 animation. HIERARCHICAL INTERLEAVE has a significant triangle setup cost, due to that each triangle is bounded at $N = 64$ discrete times. Also, the interpolation setup is more expensive, as the shading is supersampled. Additionally, the sample test work is increased due to a larger screen space tile size. We have experimented with finer tile sizes, but that resulted in a substantial increase in the tile test cost, making the overall cost increase. INTERVAL performs

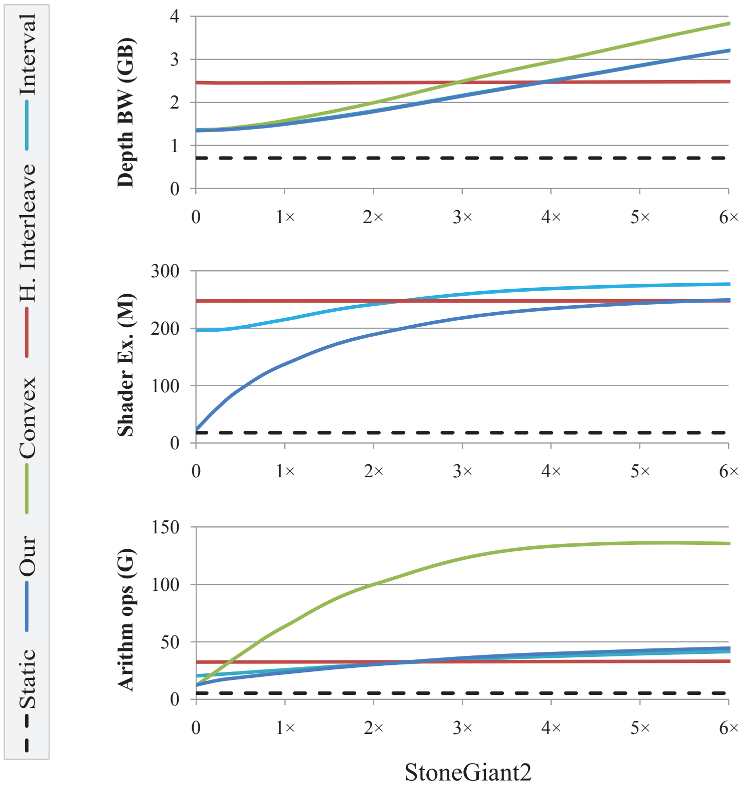


Figure 16: *The total depth buffer bandwidth, #shader executions, and arithmetic cost for the StoneGiant2 scene, as functions of the motion amount.*

quite poorly for this test scene. The reason for this is that the model contains large sliver triangles and since the INTERVAL rasterizer lacks a hierarchical tile-overlap test, it performs many unnecessary sample tests. For CONVEX, the cost of sample testing dominates, and varies widely. Our algorithm has more expensive per-tile tests, but they manage to cull a larger part of subsequent work. The total arithmetic cost using our two tile tests (moving box and moving edge) is presented in Table 2. We also measured the efficiency of our linearized edge test compared to directly solving the quadratic edge equation per tile. On the entire SubD11 animation (the scene with most complex motion), the linearized test results in 8% more inside test, but reduces the total arithmetic cost by 13% thanks to less expensive per-tile computations. On the Heaven scene, the linearized test results in 4% more inside test, but a total cost reduction of 24%.

In Figures 15–17 (bottom row), we show how the four different motion blur algorithms scale with increased motion. StoneGiant1 and Heaven both have motion

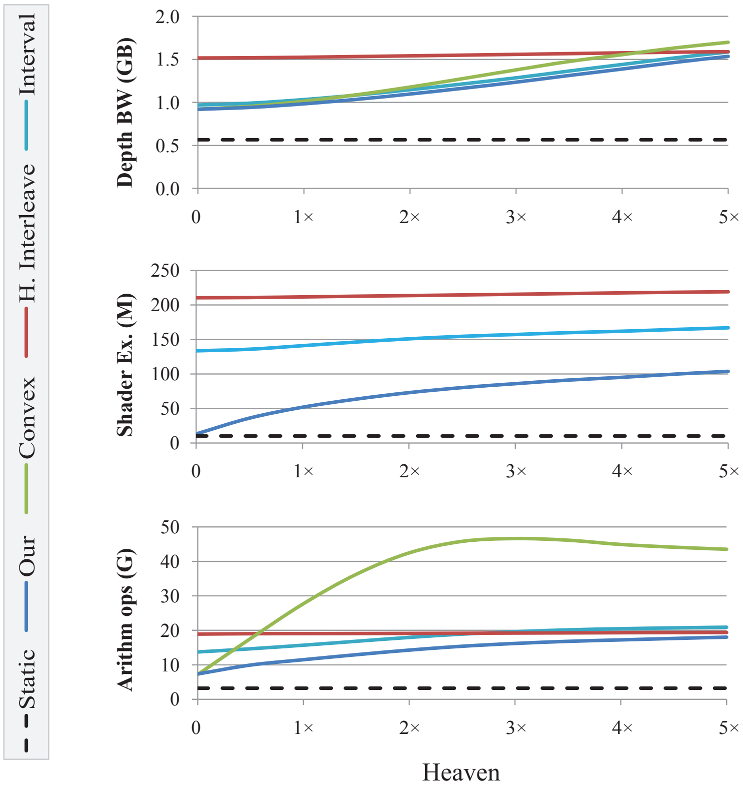


Figure 17: The total depth buffer bandwidth, #shader executions, and arithmetic cost for the Heaven scene, as functions of the motion amount.

blur from a camera translation and show similar trends despite a large difference in tessellation. For modest motion ($< 1\times$), our algorithm has about half the traversal cost of HIERARCHICAL INTERLEAVE, and has roughly the same cost at extreme motion. INTERVAL scales similar to our algorithm but has lower overall performance. CONVEX is efficient for small motion, but the arithmetic cost grows very quickly for larger motion. StoneGiant2 is a highly tessellated frame with a camera rotation, so that each triangle gets similarly long motion trails. There are about a million triangles in the two front-most spiders alone. This is a worst-case scenario for a screen space *hierarchical* traversal, as a large fraction of the scene geometry is near pixel-sized. Here, HIERARCHICAL INTERLEAVE handles extreme motion robustly at a significant cost (around 40 Gops per frame). Our approach is competitive up to about $2\times$ motion. At extreme motion levels, nearly all triangles are completely separated (see Figure 10), so there is no gain in using a screen space traversal algorithm.

9 Conclusions

This paper has described the first efficient algorithm for hierarchical rasterization of motion blur. The algorithm builds on novel tile tests that compute the temporal overlap between a screen space tile and a moving primitive. We have shown that these temporal bounds are important to reduce the volume of tested samples and enable efficient hierarchical occlusion culling. We have further devised a high quality sampling method that uses the temporal bounds to quickly generate samples with good statistical properties. Our method is based on reordering of a known three-dimensional digital (t,m,s)-net to better fit the requirements for motion blur rasterization. Finally, we have provided extensive measurements of depth buffer bandwidth usage, arithmetic intensity, and shader efficiency for MSAA on modern complex workloads, which we have not seen in other studies.

In this paper, we have taken one step towards efficient motion blurred rendering on graphics processors. Our focus has been on a rather non-intrusive change to current GPUs. One area that needs more work is efficient shading, which makes the next natural step to add a decoupled shading cache [30, 7]. This is left for future work at this point. In addition, it would be interesting to transform the algorithms into using efficient fixed-point math robustly. We hope that our work will help drive a continued interest in stochastic rasterization as a realistic method to achieve high-quality motion blur in future graphics pipelines.

Acknowledgements

Thanks to Tobias Persson from BitSquid for letting us use the StoneGiant demo, and to Denis Shergin from Unigine for letting us use images from Heaven 2.0. Tomas Akenine-Möller is a Royal Swedish Academy of Sciences Research Fellow supported by a grant from the Knut and Alice Wallenberg Foundation. In addition, we acknowledge support from the Swedish Foundation for strategic research.

A Derivation of Our Generator Matrices

We base our sampling method on the $(0,m,3)$-net proposed by Grünschloß and Keller [15]. Their first two generator matrices, giving the Larcher-Pillichshammer (LP) points, are defined as:

$$C_1 = \begin{pmatrix} 0 & & 1 \\ & \iddots & \\ 1 & & 0 \end{pmatrix} \quad \text{and} \quad C_2 = \left(\begin{cases} 1 & \text{if } k \le l, \\ 0 & \text{else} \end{cases} \right)_{k,l=1}^{m}, \tag{13}$$

and the third component is generated by the matrix:

$$C_3 = \left(\binom{l}{k} \mod 2 \right)_{k,l=1}^{m}. \tag{14}$$

To make the generated point set better suited for motion blur rasterization with our algorithm, we propose a new construction based on the matrices $C_i' = C_i D$, where $D = C_3^{-1} C_1$. D is chosen so that $C_3' = C_3 C_3^{-1} C_1 = C_1$, which generates points ordered in the third component, while keeping the first two components as the LP-points. Note that C_3 is an upper triangular matrix as $\binom{l}{k} = 0$ if $k > l$, and its determinant is thus the product of the diagonal entries, $\det(C_3) = \prod_{i=1}^{m} \binom{i}{i} = 1$, which shows it is of full rank and therefore invertible.

We start by computing the inverse C_3^{-1}. Note that C_3 is closely related to the Pascal matrix $P_n(i,j) = \binom{i-1}{j-1}, 1 \le i,j \le n$. Its inverse, P_n^{-1} is known [5] and has elements $P_n^{-1}(i,j) = (-1)^{i-j}\binom{i-1}{j-1}$. In $\mathbb{F}_2$, $-1 = 1$, and hence the term $(-1)^{i-j}$ disappears and $P_n^{-1} = P_n \pmod 2$. The elements of C_3 are the same as the lower-right submatrix of size $m \times m$ of $P_{m+1}^T \mod 2$. This result lead us to believe that C_3 is its own inverse in $\mathbb{F}_2$, i.e., $C_3^{-1} = C_3$. As a proof, we show that the elements of $C_3 C_3^{-1} = C_3^2 = I$ are:

$$C_3^2(k,l) = \sum_{i=1}^{m} \binom{i}{k}\binom{l}{i} \mod 2 = \ldots = \begin{cases} 1 & \text{if } k = l, \\ 0 & \text{if } k \ne l, \end{cases} \tag{15}$$

for $1 \le k,l \le m$. First, note that nonzero terms in the sum can only occur when both binomial coefficients are nonzero, i.e., when $k \le i \le l$, due to $\binom{l}{k} = 0$ if $k > l$. Hence, in the lower left, $k > l$, all elements are zero. For $k \le l$, the sum can be expanded using a double counting combinatorial proof [18]:

$$\sum_{i=1}^{m} \binom{i}{k}\binom{l}{i} = \sum_{i=k}^{l} \binom{i}{k}\binom{l}{i} = 2^{l-k}\binom{l}{k}. \tag{16}$$

In the upper right, $k < l$, $C_3^2(k,l) = 2^{l-k}\binom{l}{k} \mod 2 = 0$, as all elements are multiples of 2. Along the diagonal, $k = l$, the sum reduces to $2^0\binom{k}{k} = 1$, which concludes the proof. Finally, our new matrix, C_1', for computing the first component of the points is given by:

$$C_1' = C_1 D = C_1 C_3 C_1 = \left(\binom{m+1-l}{m+1-k} \mod 2 \right)_{k,l=1}^{m}, \tag{17}$$

as pre and post-multiplication by the *exchange* matrix, C_1, results in flipping C_3 vertically and horizontally. To compute, C_2', we start with $C_2 C_3$, which is given by:

$$C_2 C_3(k,l) = \ldots = \begin{cases} \sum_{i=k}^{l} \binom{l}{i} \mod 2 & \text{if } k \leq l, \\ 0 & \text{if } k > l, \end{cases} \tag{18}$$

as the rows of C_2 are zero for columns $i < k$, and $\binom{l}{i} = 0$ if $i > l$. Through experiments, we have found that $\sum_{i=k}^{l} \binom{l}{i} = \binom{l-1}{k-1} \pmod 2$, $k \leq l$. The formal proof of this can be found by deduction. Finally, C_2' is given by flipping this matrix horizontally, i.e., replacing the column index l by $m+1-l$, as follows:

$$C_2' = C_2 D = C_2 C_3 C_1 = \left(\binom{m-l}{k-1} \mod 2 \right)_{k,l=1}^{m}. \tag{19}$$

This completes the derivation of our new set of generator matrices.

Bibliography

[1] AKENINE-MÖLLER, T., AND AILA, T. Conservative and Tiled Rasterization Using a Modified Triangle Set-Up. *Journal of Graphics Tools, 10*, 3 (2005), 1–8.

[2] AKENINE-MÖLLER, T., MUNKBERG, J., AND HASSELGREN, J. Stochastic Rasterization using Time-Continuous Triangles. In *Graphics Hardware* (2007), pp. 7–16.

[3] AKENINE-MÖLLER, T., AND STRÖM, J. Graphics for the Masses: A Hardware Rasterization Architecture for Mobile Phones. *ACM Transactions on Graphics, 22*, 3 (2003), 801–808.

[4] BOULOS, S., LUONG, E., FATAHALIAN, K., MORETON, H., AND HANRAHAN, P. Space-Time Hierarchical Occlusion Culling for Micropolygon Rendering with Motion Blur. In *High-Performance Graphics* (2010), pp. 11–18.

[5] BRAWER, R., AND PIROVINO, M. The Linear Algebra of the Pascal Matrix. *Linear Algebra and its Applications, 174* (1992), 13–23.

[6] BRUNHAVER, J., FATAHALIAN, K., AND HANRAHAN, P. Hardware Implementation of Micropolygon Rasterization with Motion and Defocus Blur. In *High-Performance Graphics* (2010), pp. 1–9.

[7] BURNS, C. A., FATAHALIAN, K., AND MARK, W. R. A Lazy Object-Space Shading Architecture with Decoupled Sampling. In *High-Performance Graphics* (2010), pp. 19–28.

[8] COOK, R. L., CARPENTER, L., AND CATMULL, E. The Reyes Image Rendering Architecture. In *Computer Graphics (Proceedings of SIGGRAPH 87)* (1987), vol. 21, ACM, pp. 95–102.

[9] COOK, R. L., PORTER, T., AND CARPENTER, L. Distributed Ray Tracing. In *Computer Graphics (Proceedings of SIGGRAPH 84)* (1984), vol. 18, ACM, pp. 137–145.

[10] FATAHALIAN, K., LUONG, E., BOULOS, S., AKELEY, K., MARK, W. R., AND HANRAHAN, P. Data-Parallel Rasterization of Micropolygons with Defocus and Motion Blur. In *High-Performance Graphics* (2009), pp. 59–68.

[11] FUCHS, H., POULTON, J., EYLES, J., GREER, T., GOLDFEATHER, J., ELLSWORTH, D., MOLNAR, S., TURK, G., TEBBS, B., AND ISRAEL, L. Pixel-Planes 5: A Heterogeneous Multiprocessor Graphics System using pProcessor-Enhanced Memories. In *Computer Graphics (Proceedings of SIGGRAPH 89)* (1989), vol. 23, ACM, pp. 79–88.

[12] GREENE, N. Detecting Intersection of a Rectangular Solid and a Convex Polyhedron. In *Graphics Gems IV* (1994), Academic Press Professional, Inc., pp. 74–82.

[13] GREENE, N., KASS, M., AND MILLER, G. Hierarchical Z-Buffer Visibility. In *Proceedings of SIGGRAPH 1993* (1993), ACM, pp. 231–238.

[14] GRIBEL, C. J., DOGGETT, M., AND AKENINE-MÖLLER, T. Analytical Motion Blur Rasterization with Compression. In *High-Performance Graphics* (2010), pp. 163–172.

[15] GRÜNSCHLOSS L., AND KELLER, A. (t,m,s)-Nets and Maximized Minimum Distance, Part II. In *Monte Carlo and Quasi-Monte Carlo Methods 2008*. Springer Berlin Heidelberg, 2009, pp. 395–409.

[16] HAEBERLI, P., AND AKELEY, K. The Accumulation Buffer: Hardware Support for High-Quality Rendering. In *Computer Graphics (Proceedings of SIGGRAPH 90)* (1990), vol. 24, ACM, pp. 309–318.

[17] KELLER, A., AND HEIDRICH, W. Interleaved Sampling. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques* (2001), Springer-Verlag, pp. 269–276.

[18] KLAVŽAR, S. Counting Hypercubes in Hypercubes. *Discrete Mathematics, 306*, 22 (2006), 2964–2967.

[19] KOLLIG, T., AND KELLER, A. Efficient Multidimensional Sampling. *Computer Graphics Forum, 21*, 3 (2002).

[20] KOREIN, J., AND BADLER, N. Temporal Anti-Aliasing in Computer Generated Animation. In *Computer Graphics (Proceedings of ACM SIGGRAPH 83)* (1983), vol. 17, ACM, pp. 377–388.

[21] LOVISCACH, J. Motion Blur for Textures by Means of Anisotropic Filtering. In *Rendering Techniques 2005* (2005), pp. 105–110.

[22] MCCOOL, M. D., WALES, C., AND MOULE, K. Incremental and Hierarchical Hilbert Order Edge Equation Polygon Rasterization. In *Graphics Hardware* (2001), pp. 65–72.

[23] MCCORMACK, J., AND MCNAMARA, R. Tiled Polygon Traversal using Half-Plane Edge Functions. In *Graphics hardware* (2000), pp. 15–21.

[24] MCGUIRE, M., ENDERTON, E., SHIRLEY, P., AND LUEBKE, D. Real-Time Stochastic Rasterization on Conventional GPU Architectures. In *High Performance Graphics* (2010), pp. 173–182.

[25] MOREIN, S. ATI Radeon HyperZ Technology. In *Graphics Hardware, Hot3D Proceedings* (2000).

[26] MUNKBERG, J., AND AKENINE-MÖLLER, T. Backface Culling for Motion Blur and Depth of Field. *journal of graphics, gpu, and game tools (to appear)* (2011).

[27] NIEDERREITER, H. *Random Number Generation and Quasi-Monte Carlo Methods*. SIAM, 1992.

[28] OLANO, M., AND GREER, T. Triangle Scan Conversion using 2D Homogeneous Coordinates. In *Graphics Hardware* (1997), pp. 89–95.

[29] PINEDA, J. A Parallel Algorithm for Polygon Rasterization. In *Computer Graphics (Proceedings of SIGGRAPH 88)* (1988), vol. 22, ACM, pp. 17–20.

[30] RAGAN-KELLEY, J., LEHTINEN, J., CHEN, J., DOGGETT, M., AND DURAND, F. Decoupled Sampling for Graphics Pipelines. *ACM Transactions on Graphics (to appear), 30*, 3 (2011).