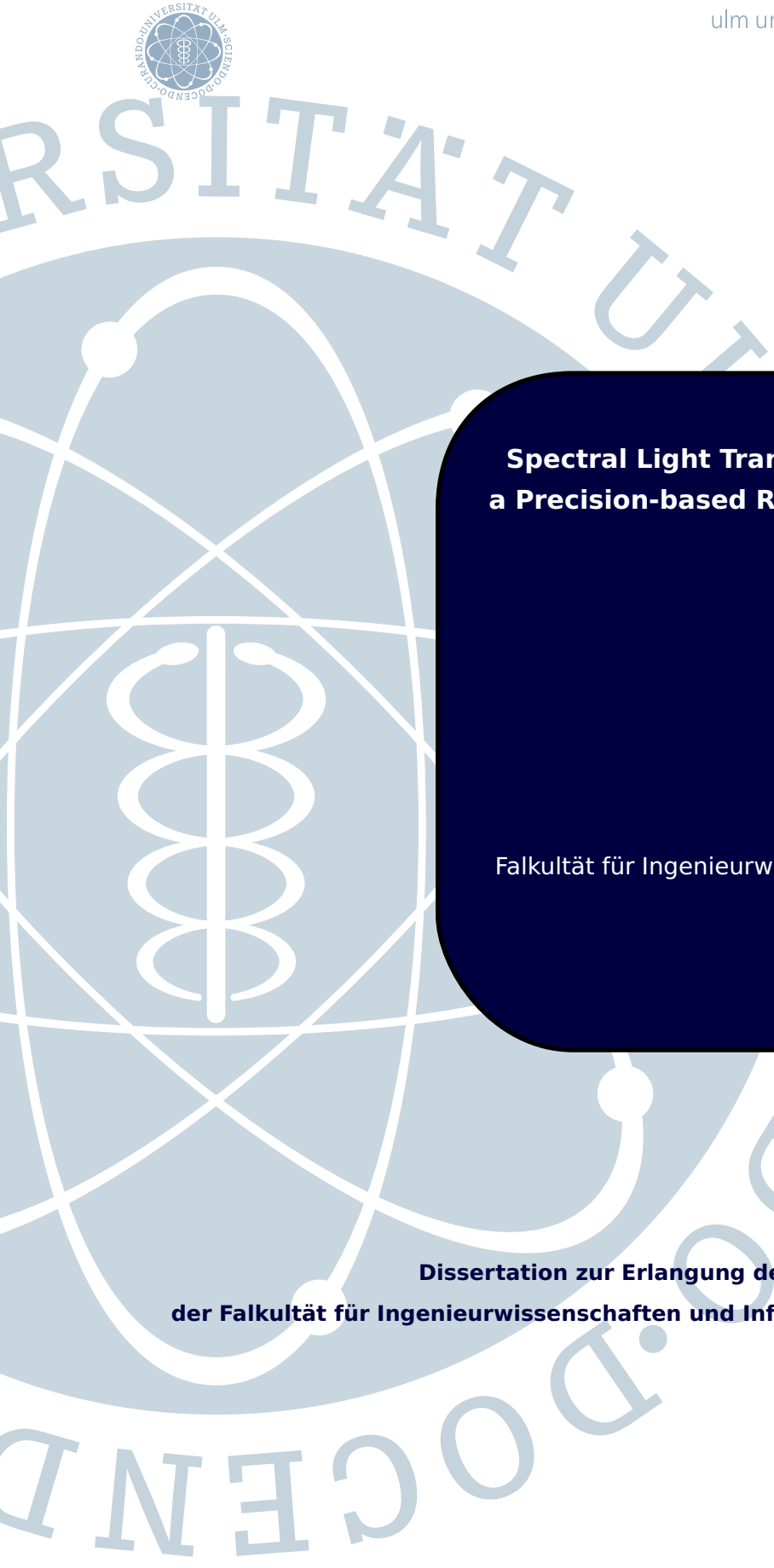


ulm university universität
ulm

Spectral Light Transport Simulation using a Precision-based Ray Tracing Architecture

vorgelegt von
Johannes Hanika
Geb. in Waiblingen

Institut für Medieninformatik
Falkultät für Ingenieurwissenschaften und Informatik
Universität Ulm

2010

**Dissertation zur Erlangung des Doktorgrades Dr. rer. nat.
der Falkultät für Ingenieurwissenschaften und Informatik der Universität Ulm**

Amtierender Dekan:	Prof. Dr. Klaus Dietmayer
Gutachter:	Dr. Alexander Keller
Gutachter:	Prof. Dr. Hendrik P. A. Lensch
Gutachter:	Prof. Dr. Jan Kautz
Tag der Promotion:	27.01.2011

Abstract

Rendering is one of the main areas of computer graphics. It is the process of creating realistic images from 3d scene descriptions and reflectance data by solving the global illumination problem e.g. by taking into account all light paths connecting the light sources and the sensors and summing up their contributions. Fields of applications include scientific visualization and simulation, fast rendering of vast datasets for visual effects in movie production, or product design. These all have their particular, high demands on a rendering system with a wide variety of input.

This thesis explores how rendering can be made more robust for these requirements. In particular, it is investigated how the calculation of intersections between light and geometry can be made numerically robust, to guarantee that no intersections are missed due to inaccuracies. Additionally, a novel way to efficiently handle very large input data is investigated. This method is based on reordering ray buffers and makes it possible to ray trace complex input data consisting of billions of micro-polygons. It is demonstrated that expensive creation of procedural geometry, out-of-core techniques, and large shading data can be used with this approach. On top of this, a color managed, bispectral light transport framework is presented, which can handle fluorescent materials. Examples of spectrally and bispectrally acquired data sets are shown. With these contributions, a rendering system can be created which can precisely simulate physically-based spectral light transport and robustly handle very complex geometry and materials.

“Ja eine Frage noch, habt ihr noch was auf Lager, wollt ihr noch jemanden grüßen oder wie? Äah, wir grüßen .. keinen. Viel spaß damit.”

Die Fantastischen Vier, Jetzt geht’s ab.

Acknowledgements

First, I would like to thank the mental images GmbH for support and funding of this research. Furthermore, this work has been partially funded by the DFG Emmy Noether fellowship (Le 1341/1-1).

Special thanks go to my supervisor Alex Keller, for precise, mathematical guidance with a clear view on every detail throughout my academic life, even after he left University. He helped make this thesis a lot clearer in presentation and more correct in content. His almost fanatical enthusiasm about rendering algorithms can really be a driving force!

I’m also glad to thank Hendrik P. A. Lensch to lead me through the second half of my thesis, who gave me the opportunity to see computer graphics once again from an all different angle and introduced me to the computational photography part of the graphics community. This gave me insights into another set of interesting problems and into the ways people work. Also his ability to keep the big picture in mind, as well as his sensible, humane guidance made work enjoyable.

I also want to express my gratitude to Jan Kautz, who agreed to take the role of the external reviewer on such short notice.

In addition, I wish to thank Holger Dammertz for countless productive discussions, and a fun working environment (including rock climbing, sailing, ...).

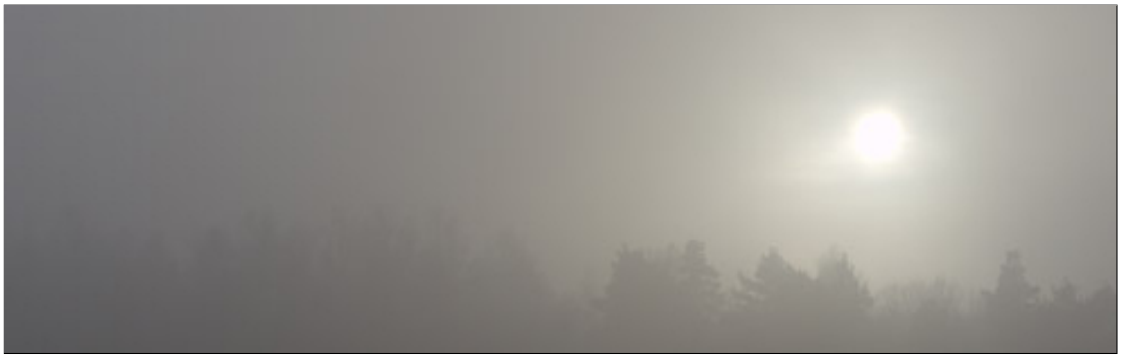
Further thanks go to Matthias Raab for a lot of help with the mathematical part of the BRDF chapter, to Matthias Hullin for measuring the bispectral BRDF which we also used to test large shader data in the Rayes part. Thanks also to Daniel Seibert, who took the reference pictures of the spectrally measured BRDF, and to X-Rite for providing the samples.

The fluorescence simulation profited much from great cooperation with Marius Peters and the measurement equipment at the Fraunhofer institute for solar energy systems in Freiburg. In this context Marion Bendig’s work on her master’s thesis was also very helpful.

I’m happy that some people took the time for proof reading this thesis, namely Holger Dammertz, Leonhard Grünschloß , Sehera Nawaz, and Matthias Raab.

Another necessary mention is the irt/hpg crowd which made the last year of conferences very enjoyable, as well as Chris Fox and Niko Bellić, who made our office a better place, along with my fellow nerds on #darktable, who provided a worthwhile distraction.

Contents



“Accuracy, heh! Efficiency, hah! The government needs not these things.”

The Dvorak Zine

1 Introduction

The goal of rendering in computer graphics is to provide intriguing, realistic images. This can go as far as predictive rendering, where light transport simulation for physics research or engineering and visualization for design can be achieved with the same algorithms. These algorithms, stripped down and tuned for speed, compute effects for movies or even for modern games in real-time.

Since the days of the first synthesized images based on light transport the demands have been rising continuously. Nowadays, a renderer has to support a large number of effects, such as depth of field, motion blur (see Figure 1.1, bottom), massive geometry (see Figure 1.2, bottom right and Figure 1.1, bottom), complex materials (see Figure 1.1 top row and middle left), global illumination (see Figures 1.2 and 1.1), and difficult paths (see the caustics in Figure 1.2). When evaluating global illumination, some even argue that considering these effects individually does not help, since the high-dimensional space of light paths is general enough to model all of these in a unified way.

Being well explored, rendering is used in a variety of fields, each with different demands, but they all require rendering to be *robust* for arbitrary input data.

- Robust for artists, architects, and product designers means to get a smooth

image in little time, which is as close as possible to the real solution. Especially faithful material properties and color rendition are important.

- Robust for physicists and engineers means to visualize the correct and precise solution.
- For the mathematician there is a definition from probability theory where test statistics can be robust against outliers, i.e. still produce meaningful results in the presence of very high variance. An example for such a robust statistic is the median.
- Robust for the movie industry includes to handle arbitrarily complex geometry, possibly at the cost of precision.

In this list, two commonly neglected features are present. The first one is spectral rendering, which provides true color rendition and can effectively model true reflection properties of complex materials. While the fundamentals are well known, it is rarely implemented in a rendering system, or comes with the disclaimer of being very slow. Also the opportunity to implement wavelength shifting effects and a color managed pipeline is not commonly taken.

If spectral rendering is chosen, real physical quantities can be calculated, not just a pleasing output for display. This is often required for the visualization and simulation needs of engineers and researchers of other fields than computer graphics.

The other aspect is precision. Quite the opposite is usually done in high performance computing. Divisions are replaced by approximate reciprocals, GPU math functions are often stripped down to fast approximations, the denormalized numbers in the floating point standard are ignored. The outliers resulting from such erroneous computations are then to be filtered out at the end.

In this thesis, we want to address all of these demands and thus make rendering algorithms more applicable to a wider field of problems from different disciplines.

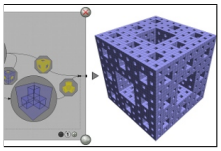
1.1. Summary of Contributions

In this thesis, we developed new techniques to make rendering more robust. In particular, these are:

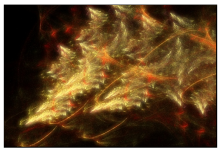
- A color managed spectral rendering framework which is able to simulate physical effects such as fluorescence correctly.
- A new analytic BRDF lobe function, which can also be used to create radial basis functions restricted to the hemisphere.
- A robust ray/triangle intersection method with easy to control, guaranteed precision in die area-saving fixed point arithmetic.

- An efficient way to handle highly complex geometry made up of micro-polygons which are created on-demand by level of detail rules, in the ray tracing setting.

Some of these topics have already been published in various publications.

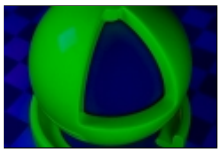


Stefan Menz, Holger Dammertz, Johannes Hanika, Hendrik Lensch, and Michael Weber. Graphical Interface Models for Procedural Mesh Growing. In *Vison, Modeling and Visualization*, pages 17–24, 2010 [MDH+10].



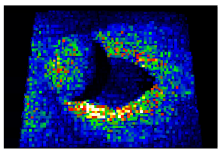
Christoph Schied, Johannes Hanika, Holger Dammertz, and Hendrik Lensch. High Performance Iterated Function Systems. In *GPU Computing Gems 2010*, pages to appear, 2010 [SHDL10].

Holger Dammertz, Daniel Sewtz, Johannes Hanika, and Hendrik Lensch. Edge-avoiding à-trous wavelet transform for fast global illumination filtering. In *Proc. High Performance Graphics 2010*, pages 67–75, 2010 [DSHL10].

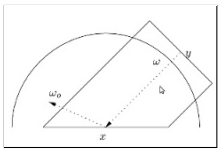


Matthias Hullin, Johannes Hanika, Boris Ajdin, Jan Kautz, Hans-Peter Seidel, and Hendrik Lensch. Acquisition and analysis of bispectral bidirectional reflectance and reradiation distribution functions. *ACM Transactions on Graphics (Proc. SIGGRAPH 2010)*, 2010 [HHA+10].

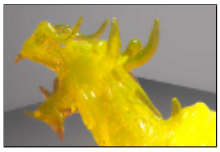
Johannes Hanika, Alexander Keller, and Hendrik Lensch. Two-level ray tracing with reordering for highly complex scenes. In *Proc. of Graphics Interface 2010*, pages 145–152, 2010 [HKL10].



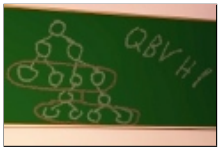
Holger Dammertz, Johannes Hanika, Alexander Keller, and Hendrik Lensch. A hierarchical automatic stopping condition for Monte Carlo global illumination. In *Proc. of the WSCG 2009*, pages 159–164, 2009 [DHKL09].



Holger Dammertz and Johannes Hanika. Plane sampling for light paths from the environment map. *Journal of graphics, gpu and game tools*, 14(2):25–31, 2009 [DH09].



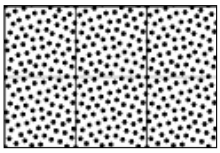
Marion Bendig, Johannes Hanika, Holger Dammertz, Jan Christoph Goldschmidt, Marius Peters, and Michael Weber. Simulation of fluorescent concentrators. In *Proc. 2008 IEEE/EG Symposium on Interactive Ray Tracing*, pages 93–98, 2008 [BHD+08].



Holger Dammertz, Johannes Hanika, and Alexander Keller. Shallow bounding volume hierarchies for fast SIMD ray tracing of incoherent rays. In *Computer Graphics Forum (Proc. 19th Eurographics Symposium on Rendering)*, pages 1225–1234, 2008 [DHK08].

Matthias Raab, Johannes Hanika, Leonhard Grünschloß, Manuel Finckh and Alexander Keller. Benchmarking Ray Tracing for Realistic Light Transport Algorithms. `http://bwfirt.sf.net/`, 2007 [RHF+07].

Johannes Hanika and Alexander Keller. Towards hardware ray tracing using fixed point arithmetic. In *Proc. 2007 IEEE/EG Symposium on Interactive Ray Tracing*, pages 119–128, 2007 [HK07].



Leonhard Grünschloß Johannes Hanika, Ronnie Schwede, and Alexander Keller. (t, m, s)-nets with maximized minimum distance. In *Proc. Monte Carlo and Quasi-Monte Carlo Methods 2006*, pages 397–412. Springer, 2008 [GHSK08].

1.2. Structure of this Thesis

This thesis proceeds as follows. First, a short overview of the required background is given in Chapter 2, along with the tools and implementation details necessary to implement spectral rendering. Chapter 3 proceeds by examining methods to reproduce the appearance of objects by measuring and simulating the spectral reflection properties, and Chapter 4 extends this to include fluorescence. In Chapter 5, precise methods to calculate intersections of rays of light with geometry are discussed. Chapter 6 takes this further by extending voxel-based intersection methods to use level of detail and to support massive amounts of geometry in an efficient way.

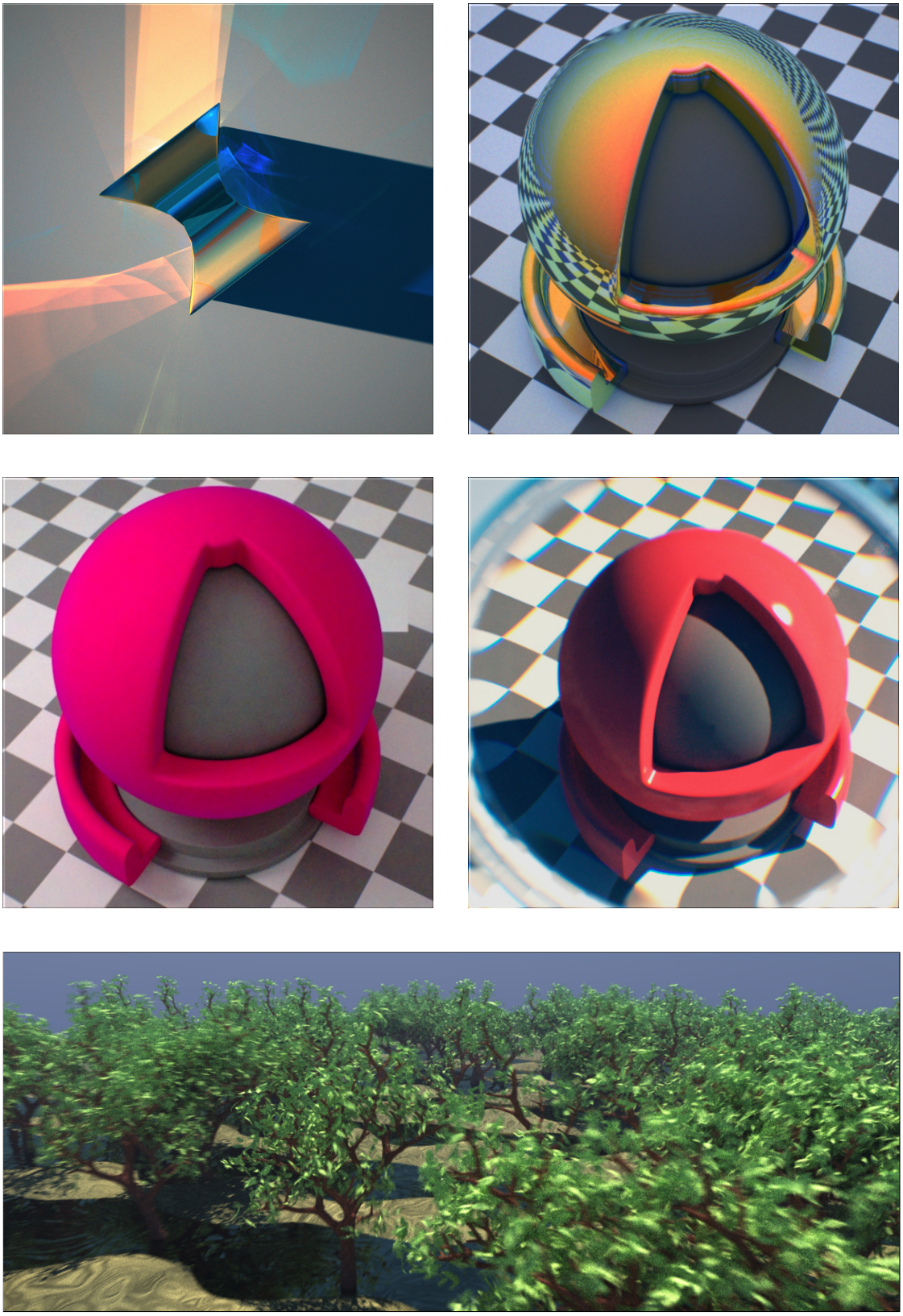


Figure 1.1.: Sample renderings showing complex materials with spectral effects such as thin film interference (top row), fluorescence (middle left), chromatic aberrations in complex lenses (middle right) and complex geometry with motion blur (bottom).

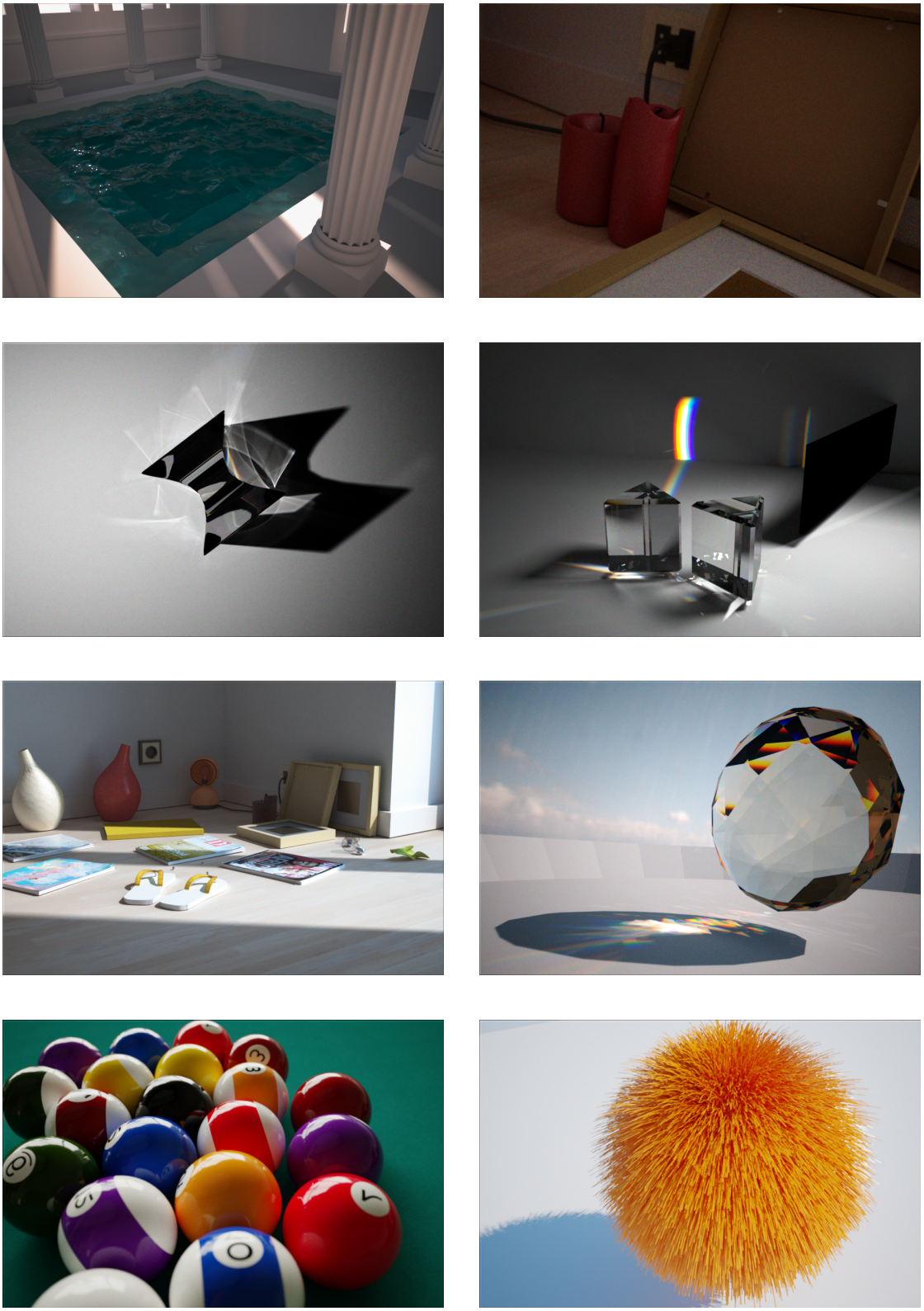


Figure 1.2.: Sample renderings showing global illumination in some common testing scenarios with difficult paths such as complex caustics, color bleeding, and multiple scattering.

“After all, all he did was string together a lot of old, well-known quotations.”

H. L. Mencken, on Shakespeare

2

Spectral Light Transport Simulation

This chapter summarizes mathematical and algorithmic tools needed throughout the course of this thesis. Some mathematical basics, however, are assumed to be familiar to the reader (such as knowledge of the complex numbers, basic algebra in a Hilbert space, or common tools such as principal component analysis). A good introduction to the graphics related algorithms can be found in Shirley’s fundamental book [Shi02].

The foundation of most rendering algorithms is *geometric optics*, specifically vacuum transport with extensions to homogeneous media will be of interest to us. This means that we assume that the structure sizes of the surfaces interacting with light are much larger than the wavelength, so no diffraction takes place in our far field computations. That is, rather than working with Huygen’s principle of wavefront propagation, we will make use of Fermat’s principle which basically results in light travelling along straight lines in homogeneous media. We also assume instantaneous equilibrium distribution, i.e. no temporal or relativistic effects (except motion blur due to long shutter times) are observed, and light paths are not bent by gravity. We only simulate incoherent light sources and, in most cases, we can even ignore polarization.

What we want to consider, however, is the dependency on wavelength. More precisely, the principles of accurate color representation and reproduction are

given in Section 2.1, the spectral global illumination problem is described in Section 2.2, some principles of the Monte Carlo method are discussed in Section 2.3, followed by methods to explore path space in Section 2.4. Finally, Section 2.5 develops a spectral rendering system on top of this background.

2.1. Colorimetry

It is widely acknowledged that the reproduction of colors benefits from spectral treatment above the usual red, green, blue (RGB) channels. A very comprehensive treatment of many questions within the field of color science was given by Nassau [Nas83]. This section summarizes the very basics needed to convert radiometric quantities to color and the other way round. A more gentle introduction to color can be found in Peter Shirley's book [Shi02], and definitions of photometric quantities (vs. radiometric as listed here) are listed in the global illumination compendium [Dut03].

Physical Quantities. To be able to convert physical quantities into color, we need to define a few of them.

Flux is the radiant energy flowing through a surface per second $\Phi = dQ/dt$. This value is proportional to the photon count passing the surface. Since all following quantities will depend on the wavelength λ, it is convenient to define

$$\Phi = \frac{dQ}{dt} \qquad \left[W = \frac{J}{s}\right]. \tag{2.1}$$

Spectral irradiance or Spectral power distribution is the incident spectral flux per surface area $A(x)$ at position x

$$E(x,\lambda) = \frac{d^2\Phi(\lambda)}{dA(x)\cdot d\lambda} \qquad \left[\frac{W}{m^2\cdot nm}\right]. \tag{2.2}$$

Spectral radiance is the spectral flux per projected differential surface area $dA^{\perp}_{\omega}(x) = \cos\theta \cdot dA(x)$, and solid angle

$$L(x,\omega,\lambda) = \frac{d^3\Phi(\lambda)}{d\omega\cdot dA^{\perp}_{\omega}(x)\cdot d\lambda} \qquad \left[\frac{W}{m^2\cdot sr\cdot nm}\right]. \tag{2.3}$$

Color Input. As input data, directly acquired spectral data is preferred. Some common input spectra are available, such as the solar spectral irradiance AM 1.5 [Ame], and the CIE standard illuminant data for white balance lighting conditions. The absolute radiance $L(x,\omega,\lambda)$ can be measured by exposure bracketing [Deb98] for every wavelength (see Figure 2.1). Similar measurements can be

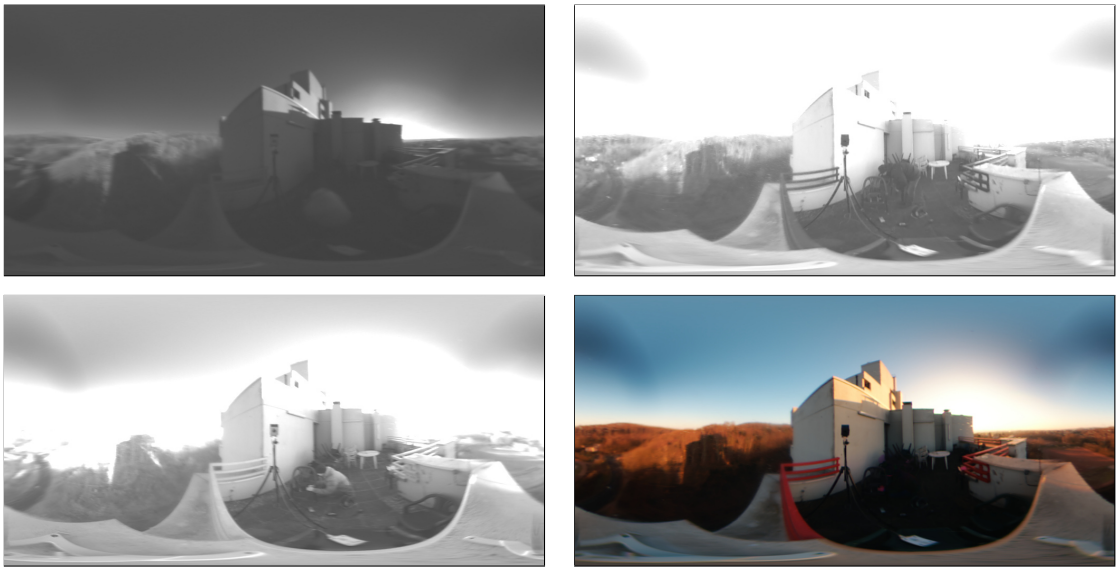


Figure 2.1.: A sky measured from 400 to 720 nm, in steps of 10 nm. Pictures of a mirroring ball where acquired through wavelength dependent filters [HHA+10]. From top left to bottom right: radiance at 400 nm, 550 nm, 720 nm, and the corresponding reconstructed sRGB image. The first three images have been tone mapped to fit the display.

done for reflectances [HHA+10]. For participating media, the spectral absorption and scattering coefficients can be acquired by measurement, as we will show in more detail in Chapter 3.

However, most of the existing input data to rendering systems is given in some variant of the RGB color space. Since expanding three dimensions to a continuous spectrum is an ill-posed problem, there exist various methods which yield slightly different results.

The simplest solutions (i.e. use a box basis with ten dimensions [Smi99] or use the first three Fourier bases [Dut03]) work well and precise enough for non-spectral input data.

Color Output. After the incoming spectral irradiance at a pixel of the sensor is known, it has to be converted to an image for display in the context of computer graphics. This works equivalently to digital cameras, where CCD or CMOS chips store the electrons which are freed by the photoelectric effect [Ein05]. One photon will free one electron, if the light frequency is above the threshold frequency of the material. The camera's sensor response will be proportional to the number of incoming photons (times quantum efficiency) which, in turn, is proportional to the flux Φ. This is, in contradiction to the classical wave description of light, independent of wavelength. So real camera sensors actually count photons and

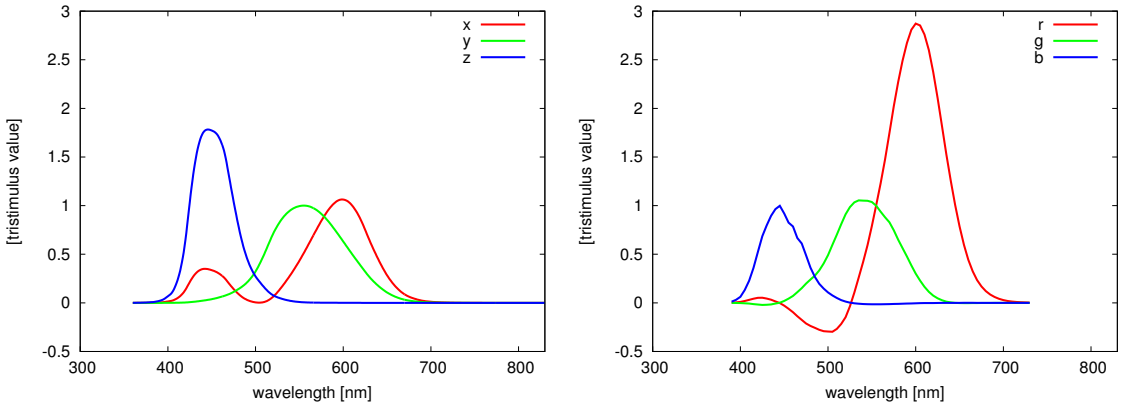


Figure 2.2.: Left: the CIE 1931 XYZ color matching functions. Right: the Stiles & Burch 1955 RGB color matching functions [SB55].

we can thus directly sum up values proportional to Φ in the accumulation buffer.

To accurately reproduce color, we accumulate in the well defined CIE XYZ color space. A thorough historical introduction to how this color space was derived can be found in [FBH98]. The accumulation buffer used for rendering stores $E(\lambda)$ as XYZ tristimulus values, since these values are obtained by a linear operator on the spectral power distribution $E(\lambda)$:

$$X = \int_\Lambda x(\lambda)E(\lambda)d\lambda \tag{2.4}$$

$$Y = \int_\Lambda y(\lambda)E(\lambda)d\lambda \tag{2.5}$$

$$Z = \int_\Lambda z(\lambda)E(\lambda)d\lambda. \tag{2.6}$$

where the tristimulus value functions x, y, and z (see Figure 2.2, left) are normalized to integrate to the same as the 1924 CIE spectral luminous efficiency function $V(\lambda)$. This way, the accumulation over the pixel area A

$$E(\lambda) = \frac{1}{\|A\|}\int_A\int_\Omega L(x,\omega,\lambda)d\omega\, dx$$

and the projection onto x via Equation (2.4) can be done in one step

$$X = \frac{1}{\|A\|}\int_A\int_\Lambda x(\lambda)E(\lambda)d\lambda dx \tag{2.7}$$

$$= \frac{1}{\|A\|}\int_A\int_\Lambda x(\lambda)\int_\Omega L(x,\omega,\lambda)\,d\omega d\lambda dx \tag{2.8}$$

$$= \frac{1}{\|A\|}\int_\Omega\int_A\int_\Lambda x(\lambda)L(x,\omega,\lambda)\,d\lambda dx d\omega \tag{2.9}$$

$$\approx \sum_k x(\lambda_k)L_k/p_k, \tag{2.10}$$

where Equation (2.10) approximates the integral by summing up Monte Carlo samples. We can therefore use a memory saving accumulation buffer with only three XYZ components per pixel instead of one with a spectral resolution. Accumulation in XYZ will produce accurate results as long as no clamping or gamut mapping is performed until the full image is accumulated. From XYZ, ICC profiles can be used to convert to the desired output color space, e.g. sRGB.

The CIE XYZ color space is a common choice of a profile connection space (PCS) in color management software (e.g. [Mar98]), because it is well standardized. Another choice would be Lab (and variants), but this is unsuitable for our needs because L is calculated from a non-linear transform, and would thus complicate the accumulation buffer. Plus, Lab is not very well suited for high dynamic range data, because the useful range of L is limited to $[0, 100]$. XYZ is also preferred over linear RGB color spaces for spectral accumulation, as the RGB color matching functions have negative values. This can lead to problems while converting to and from spectral power distributions, especially when supporting legacy RGB-based BRDFs.

Tone Mapping (see for example [Shi02]) is commonly applied to computer created images to fit the displayable or printable range. To make the final images look more realistic, we either store the results in XYZ as 16-bit digital negatives (DNG) [Inc04], or use a high-dynamic range enabled photography tool [Han09] with an XYZ input profile. This way, the data is treated like a real photograph, i.e. contrast enhancing base curves and gamma shapers from the output profile of the screen are applied transparently.

Usually, when not outputting images for a calibrated screen, the ICC output color profile will be sRGB, i.e. the CIE RGB color matching functions [SB59] (as can be seen in Figure 2.2, right) with the D65 white point and a gamma shaper with a small linear toe slope.

2.2. The Spectral Global Illumination Problem

To compute the incoming spectral radiance which is then converted to a display color system, it is necessary to solve the global illumination problem including the spectral domain. A good reference to find more information about this topic is for example the state of the art report about spectral rendering [DCWP02]. A very handy, compact collection of a lot of light transport related definitions can be found in the global illumination compendium [Dut03]. The following is a list of essential spectral formulas.

Spectral BRDF. It can be formally defined how light is reflected at the surface of objects by introducing the spectral bidirectional reflectance distribution function

(BRDF) as the outgoing radiance divided by the incoming irradiance [NRH+77]

$$f_r(\omega_i, \omega_o, \lambda) = \frac{dL(x, \omega_o, \lambda)}{dE_i(x, \lambda)} = \frac{dL(x, \omega_o, \lambda)}{L(x, \omega_i, \lambda) \cos\theta_i d\omega_i} \qquad \left[\frac{1}{sr}\right]. \tag{2.11}$$

Spectral Rendering Equation. The rendering equation [Kaj86] can be formulated to include spectral light transport:

$$L(x, \omega, \lambda) = L_e(x, \omega, \lambda) + \int_\Omega f_r(x, \omega, \omega_i, \lambda) L(y, -\omega_i, \lambda) |\langle n_x, \omega_i \rangle| d\omega_i, \tag{2.12}$$

where L_e defines the light sources, and $\langle n_x, \omega_i \rangle = \cos\theta_i$ is the cosine of the elevation angle between the incoming direction ω_i and the surface normal n_x, to reflect Lambert's law. The point $y := h(x, \omega_i)$ is the closest surface point in direction ω_i from x. If the domain Ω is defined as the whole sphere (and f_r is defined to be zero on the lower hemisphere for solid surfaces), this is a Fredholm integral equation of the second kind. If Ω is defined to be the incoming hemisphere only, it is called Volterra integral equation of the second kind, as the integration domain depends on the location x.

The same equation can be expressed as integral over the boundary $\mathcal{V}$ (the surface of all objects) with the area measure A:

$$L(x, \omega, \lambda) = L_e(x, \omega, \lambda) + \int_\mathcal{V} f_r(x, \omega, \omega_i, \lambda) L(y, -\omega_i, \lambda) \underbrace{\frac{\cos\theta_i \cos\theta}{\|y - x\|^2}}_{=:G(x,y)} V(x, y) dA. \tag{2.13}$$

The additional terms G and V are called the *geometric term* and *visibility*, respectively. The latter evaluates to one if the two points are mutually visible, and zero otherwise. The transport operator formulation is often used as a convenient shortcut to replace the lengthly integral in Equation (2.12):

$$TL := \int_\Omega f_r(x, \omega, \omega_i, \lambda) L(y, -\omega_i, \lambda) |\langle n_x, \omega_i \rangle| d\omega_i \tag{2.14}$$

$$L = \sum_{i=0}^{\infty} T^i L_e. \tag{2.15}$$

Equation (2.15) is called the Neumann series expansion.

2.3. The Monte Carlo Method

The rendering equation (2.12) cannot be solved explicitly, and when expanding the Neumann series, its domain has infinite dimensions. Especially the visibility term or the evaluation of the next intersection with the boundary in a certain direction are best evaluated using point queries. These properties make the Monte

Carlo method a good candidate for the solution, reducing the problem to randomly finding a lot of paths connecting the sources and the sinks and summing up the contributions. Before we can summarize the basic principles of the Monte Carlo method, we want to give a few notations from probability theory.

Random variable. We will mostly use *continuous random variables* and denote them by X, and x_i as the realizations (i.e. actual outcomes or values of the random variable).

Probability density function. The distribution of a random variable is described by the *probability density function* (PDF) $p(x) > 0$ which is normalized, i.e. $\int p(x)dx = 1$. To express that x_i is a realization following a particular PDF, we write $x_i \sim p(x)$, or likewise for the random variable $X \sim p(X)$.

Cumulative distribution function. To measure the probability of an event such as $x \in A \subseteq \Omega$ the *cumulative distribution function* can be used:

$$P(X \in A) = \int_A p(x)dx. \tag{2.16}$$

Expected value. We write the *expected value* of a random variable $X \sim p(X)$ defined in the domain Ω as

$$\mathrm{E}(X) = \int_\Omega x \cdot p(x)dx. \tag{2.17}$$

Variance. The *variance*, i.e. expected squared deviation from the expected value will be denoted by

$$\begin{aligned} \mathrm{Var}(X) &= \mathrm{E}\left((X - \mathrm{E}(X))^2\right) && (2.18)\\ &= \mathrm{E}(X^2) - \mathrm{E}^2(X) && (2.19)\\ &= \int_\Omega (x - \mathrm{E}(X))^2 p(x)dx. && (2.20) \end{aligned}$$

The Monte Carlo method has been used for a long time to solve hard high-dimensional problems such as neutron transport in nuclear facilities (e.g. [MRR+53]). A deep and thorough reference is Ermakow's book [Erm75], and there is also a great, more hands-on overview [Sob94] by Sobol'.

The simple principle of the Monte Carlo method is to replace deterministic quadrature by random sampling, i.e. replace the integral over a function $f(x)$ by an *estimator*:

$$\underbrace{\int f(x)dx}_{=:\theta} = \mathrm{E}(X) \approx \underbrace{\frac{1}{N}\sum_{i=1}^{N} \frac{f(x_i)}{p(x_i)}}_{=:\hat{\theta}}. \tag{2.21}$$

That is, the integral is viewed as the expected value of a random variable $X = f/p$, and this expected value is approximated by drawing N random realizations $x_i \sim p(x)$ of X and taking the mean. This estimator is unbiased, i.e. the bias $B(X) = \mathrm{E}(X) - \hat{\theta} = 0$ and has the probabilistic error bound

$$P\left(\|\theta - \hat{\theta}\| < 3 \cdot \sigma\right) \approx 0.99730, \tag{2.22}$$

where σ^2 is the variance of the estimator in Equation (2.21). It can, in turn, be estimated by

$$\hat{\sigma}^2 \quad = \quad \frac{1}{N-1} \sum_{i=1}^{N} (x_i - \hat{\theta})^2. \tag{2.23}$$

It follows that the computational complexity required to bring down the (probabilistic, with $P \approx 0.99730$) error below a threshold ε is in the order of

$$O\left(\frac{\sqrt{\mathrm{Var}(X)}}{\sqrt{N}}\right). \tag{2.24}$$

The important thing to note about Equation (2.24) is that it is independent of the dimension d of the domain the function f lives on. This is especially nice compared to the exponential cost of deterministic methods [TWW88]

$$\mathrm{comp}(\varepsilon) = O\left(\left(\frac{1}{\varepsilon}\right)^{d/r}\right), \tag{2.25}$$

to bring the error below a given threshold ε when the function f fulfills the smoothness r, i.e. $f \in C^r$. Of course this error bound holds for a different function class and has the advantage to be deterministic and sharp, that is by no chance the error lies above ε. The dependency on d in this equation is called the *curse of dimension*, and is notoriously hard to overcome. The downside of the Monte Carlo method on the other hand is the long tail of $1/\sqrt{N}$. After a quick improvement of the estimate at the beginning, a small amount of noise is very hard to get rid of. In this later stage, the variance of the random variable plays a large role.

To reduce variance there exist various methods, among them stratified sampling and quasi-Monte Carlo methods [Kel98] (which work best in low dimensions for graphics or when certain smoothness conditions are met, as in finance), *control variates* (which are best applied if the integrand can be perfectly sampled up to a constant difference), and importance sampling. Because of the widespread use in computer graphics, we explain this last technique in more detail.

Importance sampling draws realizations of a random variable x_i as proportional as possible to the integrand f. If perfect importance sampling is possible, i.e.

$f(x) \sim p(X)$, then

$$X = \frac{f(x)}{p(x)} = c = const. \tag{2.26}$$

$$\Rightarrow \mathrm{E}(X) = c \tag{2.27}$$

$$\Rightarrow \mathrm{Var}(X) = \int (x-c)^2 p(x) dx = 0. \tag{2.28}$$

So every estimator solely based on X will have zero variance and thus give the right result even after only one evaluation. Of course in this case the value of the integral is known in advance ($= c$) and Monte Carlo sampling doesn't make sense.

An example would be calculating the light leaving a bright diffuse surface (Spectralon) with normal n lit by a diffuse background with incoming light intensity $L_i = 1$ from all directions (for an explanation of the integral, see Section 2.2):

$$L = \int \underbrace{L_i \cdot \frac{1}{\pi} \cdot \langle \omega, n \rangle}_{=:f(\omega)} d\omega \tag{2.29}$$

$$p(\omega) = \frac{\cos\theta}{\pi} \qquad \text{where } \cos\theta = \langle \omega, n \rangle \tag{2.30}$$

$$X = \frac{f(\omega)}{p(\omega)} = 1. \tag{2.31}$$

The perfect sampling strategy pursued here is sampling the incoming hemisphere proportional to the cosine of the elevation angle. Since p needs to be normalized over this domain, $p = \cos\theta/\pi$, and thus $L = X = 1$ can be evaluated with just one sample.

Faster Random Numbers Truly in $[0,1)$. When implementing a rendering system in floating point precision and using the SIMD Mersenne twister pseudo random number generator [SM08], the result of the code piece given below can lead to surprises:

```
/** is meant to generate a random number in [0,1) (but doesn't) */
inline static float to_real2(uint32_t v)
{
  return v * (1.0/4294967296.0);
  /* divided by 2^32 */
}
```

because the output will be `1.0f` at times due to rounding, and thus possibly crash applications due to division by zero or similar. One solution, which is faster than the multiplication above and maintains the distribution of the points, is to just paste the most significant bits of the integer to the mantissa of the float. This works in the interval $[1,2)$ because the exponent is constant there (`0x3f800000`).

```
/** generates a random number on [0,1)-real-interval (float) */
inline static float to_real2f(uint32_t v)
{
  v = 0x3f800000 | (v>>9); // faster than double version.
  return (*(float*)&v) - 1.0f;
  /* paste 23 bits to mantissa */
}
```

2.4. Path Tracing

A general approach to view light transport is path space [Vea97]. It is the space of all paths light can travel by from the source to the sensor, and includes all high dimensional effects such as depth of field, motion blur, shadows, caustics, and all other secondary effects without any special treatment. To explore this high dimensional space ($d = \infty$ in theory, in practice $d > 100$ is rarely needed), the Monte Carlo method can be applied [Vea97, DBB06].

The simplest algorithm is path tracing [Kaj86]. Re-using subpaths can be performed in various ways, by caching virtual point light sources [Kel97] or irradiance or photons [WRC88, LW95, Jen96, HOJ08, HJ09]. Most of the common algorithms are implemented in PBRT [PH04], an open source rendering system, or one of the descendants, such as [VGmm98, Jak10].

Ray Tracing. To determine e.g. shadow boundaries, a way to evaluate visibility of two points is needed. Formally evaluating visibility is computing the term $V(x, y)$ from Equation (2.13). Ray tracing [Whi80, Gla89, Shi00] solves this for point queries and point/direction queries, and is accelerated by spatial acceleration structures such as the Quad-BVH [DHK08].

Path Sampling. To explore path space, i.e. draw samples from this domain, there exist a few basic techniques we will refer to in the following sections.

Path tracing (PT) starts rays at the camera lens and recursively bounces off the objects until a light source is hit by chance. Importance sampling is done by the BRDF of the current surface.

A slight improvement with remarkable impact on variance reduction in diffuse environments is *path tracing with next event estimation*, also know as *path tracing with direct light* (PTDL). This technique performs a direct connection to the light sources whenever a non-specular surface is found. This connection is done at every bounce, i.e. one random walk results in a lot of paths which contribute to the pixel. To make the estimator unbiased, no radiance may be accumulated when a light source is hit by chance (without a deterministic connection). Importance sampling is done by BRDF at each bounce, and by light source for the deterministic connection.

A *light tracer* (LT) is the reciprocal of a PTDL. It starts paths at the light sources and performs the same random walk in the opposite direction. At each bounce, a deterministic connection to the sensor is performed.

Bi-directional path tracing (BDPT) draws a sample from a PT/PTDL and an LT at the time, and also performs all deterministic connections of all in-between path vertices, not only to the light or the sensor [LW93, VG94]. To reduce variance, the resulting paths have to be weighted in a clever way using *multiple importance sampling* (MIS) [VG95].

path expression						method
L	D	$.^*$			E	PTDL
L	S^+	$.^*$		D	E	LT
L	$\{S\|S$	$.^*$		$S\}$	E	PT
L	S^+	D	$.^*$	S	E	PT + PMAP IS

Figure 2.3.: Path space is partitioned into three disjoint types of paths, which are each sampled by the most suitable method. The regular expression is based on Heckbert's path notation [Hec90]. The last line is a sub-class of the PT class and uses photon map importance sampling as additional technique.

Path Space Partitioning. While bi-directional path tracing creates a lot of paths ($O(n^2)$ for PT and LT with n bounces), it also needs to trace a lot of visibility rays to connect the PT and LT paths. Not all of these actually pay off, because multiple importance sampling will assign a very low weight to them. This is due to the fact that the deterministic connections don't follow any importance sampling strategy and thus find important paths by pure chance. In addition, MIS is hard to implement right because it requires to explicitly calculate the probability density functions of all applied techniques. This means that if a new technique is incorporated, much care has to be taken to get the PDF evaluation right.

A simpler and in some cases more efficient approach is to partition path space into subsets and use the most efficient *unidirectional* sampling method for each set. We use a partitioning as in Figure 2.3. The path expression follows Heckbert's path notation [Hec90] where L is a light source, D is a non-specular bounce, S is specular, and E is the eye or the sensor.

That is, most of the path space is sampled using a PTDL, when a non-specular bounce before the light source makes a deterministic light connection possible. Caustics, i.e. paths with at least one consecutive specular bounce after the light source, are computed using the LT formulation. This technique requires a deterministic connection to the sensor, and thus needs a non-specular surface before the end. The last disjoint set cannot be sampled by the PTDL or LT because deterministic connections are made impossible by the two specular bounces just before the light and the sesor, it is handled by a simple PT. A subclass of these paths $LS^+D.^*SE$, including caustics seen through a mirror, profits from an additional technique: photon map importance sampling (PMAP IS) [Jen95, LW95, Pha05]. This is convenient in two ways. First, these paths are particularly hard to generate if light source and sensor are both small. Second, the photons needed for such a map can be recorded during the LT pass. This will also make good use of an LT-created path if the deterministic connect to the sensor fails due to an additional specular surface between the caustic and the sensor. The effect of this technique can be observed in Figure 2.4.

Another advantage of this approach is that it doesn't require reciprocal materials as a condition for convergence, since both unidirectional methods will converge independently. While reciprocity is desirable for physical plausibility, this gives the rendering algorithm more freedom, which we can exploit even in physically-based simulations where reciprocity is not given (see Chapter 3).

In addition, we don't have to store the light path vertices as possible connection points. This allows for tight inner loops which write memory only to few registers and eliminates a possible source of bias, when the number of path vertices is limited due to memory constraints.

Metropolis Light Transport. An essential technique to improve the way path space is explored is Metropolis sampling [MRR$^+$53, Has70], which has first been applied to the light transport problem by Veach and Guibas [VG97]. They used

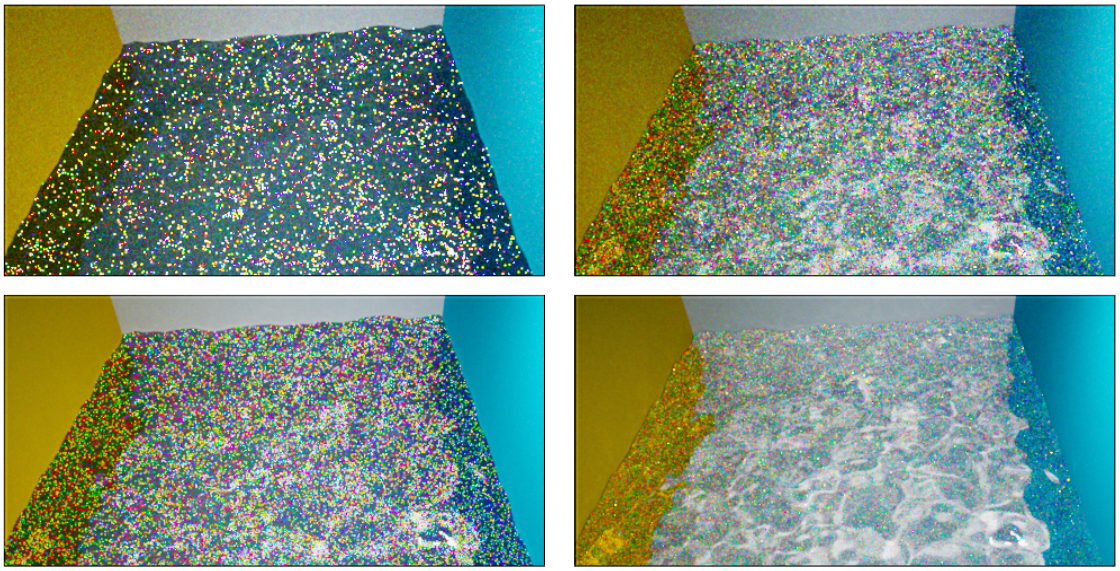


Figure 2.4.: A pool inside a box with caustics under water. The images in the top row are generated using 100 samples per pixel, the bottom row with 1000 samples. On the left partitioned path tracing is shown, on the right the same algorithm with an additional sampling technique, using the photon map for importance sampling.

custom mutation strategies to improve every effect they observed to be hard to achieve individually. Later on, Kelemen et al. [KSKAC02] simplified the algorithm by using only one single type of mutation on the random variables used as input for the path space sampler. A nice, comprehensive introduction to Metropolis light transport can be found in Cline and Egbert's technical report [CE05].

Metropolis sampling has both a theoretical and an intuitive appeal. On the theoretical side it promises perfect importance sampling, that is $p(X)$ will be proportional to $f(X)$, where X is a random variable living in the infinite dimensional path space. The intuitive version is related to how this is achieved: by a random walk on path space using small mutations. Variance is often high in very local regions of the image, as for example inside the pool in Figure 2.4. It seems counterintuitive to discard a path illuminating this area once it has been found, and start over with a completely new sample.

This is where Metropolis sampling can help out. It uses the theory of Markov chain Monte Carlo and small mutations in path space with transition probabilities $T(X_{n+1}|X_n)$ to construct a new tentative sample X_{n+1}. It is then decided randomly whether to accept this sample or to stay with the current one X_n. Intuitively, the algorithm stays in interesting regions for a while and explores the local neighborhood. Formally, it can be proven that this random walk converges to an equilibrium probability density proportional to the measurement contribution

function $f(X)$, if appropriate acceptance conditions for the tentative sample are chosen.

After equilibrium has been reached (i.e. *start up bias* is overcome), the image is formed by calculating the histogram over all states. That is, the random walk is performed, and the pixel value of the current state X_n is incremented every iteration. This histogram is then scaled by the mean image brightness b, because the probability density function has to be normalized such that $p(X) \cdot b = f(X)$.

The acceptance probability a which follows from the *detailed balance* condition

$$f(X_n)T(X_{n+1}|X_n)a(X_{n+1}|X_n) = f(X_{n+1})T(X_n|X_{n+1})a(X_n|X_{n+1}) \tag{2.32}$$

in the case of the simplified mutation strategy [KSKAC02] is

$$\begin{aligned} a(X_{n+1}|X_n) &= \min\left\{1, \frac{f(X_{n+1})T(X_n|X_{n+1})}{f(X_n)T(X_{n+1}|X_n)}\right\} && (2.33)\\ &= \min\left\{1, \frac{f(X_{n+1})}{f(X_n)}\right\}, && (2.34) \end{aligned}$$

because the transition probabilities T of the mutation on the random variables are symmetric, $T(X_n|X_{n+1}) = T(X_{n+1}|X_n)$.

To calculate the acceptance in the case of a bi-directional path tracer or a light tracer, which contribute to more than one pixel per sample, we can choose a as the ratio of the contributions $C_{i,t}(X)$ of the tentative sample to pixel i to the contributions $C_{i,c}(X)$ of the current sample

$$a(X_{n+1}|X_n) = \min\left\{1, \frac{\sum_j C_{j,t}(X)}{\sum_j C_{j,c}(X)}\right\}. \tag{2.35}$$

and accumulate the mean image brightness in each iteration by the fraction of the contributions to each pixel $C_i(X)$ and the sum for the whole sample $\sum_j C_j(X)$:

$$c_i(X) = b\frac{C_i(X)}{\sum_j C_j(X)}, \tag{2.36}$$

where $C_i(X)$ are the contributions for each pixel as determined by the standard path space sampler, and $c_i(X)$ the values to accumulate when using Metropolis sampling. This results in an unbiased estimate, as Equation (2.35) will steer the rejection sampling on the path space random walk to result in

$$p(X) = \frac{\sum_j C_j(X)}{b} \tag{2.37}$$

and thus the expected value of accumulation at a pixel i is

$$p(X) \cdot c_i(X) = \frac{\sum_j C_j(X)}{b} \cdot b\frac{C_i(X)}{\sum_j C_j(X)} = C_i(X), \tag{2.38}$$

which is the same as with the standard path space sampler. To minimize variance, this has to be combined with the standard approach of accumulating a rejected tentative sample with weight $(1 - a)$ instead of just discarding it.

One common problem of Monte Carlo methods is the so called *firefly* problem. This term refers to paths which have such a low probability to be sampled, that their value has to be astronomically high to keep the estimate unbiased. For example, to keep the right mean value in a 100×100 block of pixels, the one that finds the complicated caustic needs to be 100×100 times as bright as the correct pixel value. This pixel will probably not converge away in a long time. Metropolis improves upon this, but sometimes the path sampler is just hopelessly bad at finding complicated paths and Metropolis will slowly converge to the firefly, accumulating $1/b$ at the same pixel over again.

A commonly applied (biased) solution is to limit the number of consecutive rejects in the Markov chain. This works well in practice, as the resulting image will be the same, phenomenologically, just overly high spikes will be removed.

In Section 3.3.1, we will also make use of this kind of sampling to fit the parameter of an analytic model to measured data.

2.5. Implementation of a Spectral Rendering System

In order to receive faithful, vibrant color rendition, it is necessary to build a color managed, spectral rendering system. In this chapter, we developed the following extensions to seamlessly connect existing techniques:

The tristimulus accumulation buffer is used to do spectral accumulation of Monte Carlo paths in a memory efficient way (see Equation (2.9)), without sacrificing color accuracy. Color profiles can be applied after rendering.

Monochromatic importance sampling is necessary to avoid adding variance if the spectral domain is also sampled, and takes care that every path is importance sampled by its proper probability density (see Figure 2.5).

Path space partitioning is used to further reduce variance by avoiding low probabilities and high contributions of deterministically connected paths, and to avoid problems when BRDFs are non-reciprocal or wavelength-shifting. In these cases, deterministic connections would be inefficient. This also offers the opportunity to store photons along the light tracing paths to guide the path tracing towards the light sources more efficiently by photon map importance sampling (see Figures 2.3 and 2.4).

Metropolis light transport refines this path space sampling and has been shown to work correctly for paths which have contributions to more than one pixel (Equations (2.35) and (2.38)). Robust random numbers make sure the probabilities can be sampled precisely and the implicitly created path coherence helps

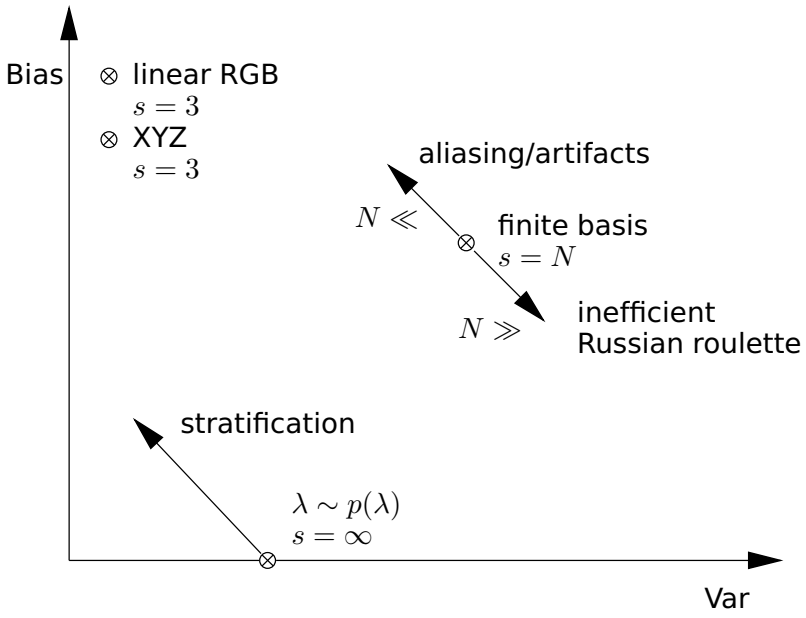


Figure 2.5.: Schematic overview of spectral transport options. Using only one random wavelength sample per path ($\lambda \sim p(\lambda)$) comes without bias, but shows more color noise as finite basis approaches. These methods, on the other hand, introduce additional variance due to inefficient Russian roulette at every bounce, which increases with the number of basis functions. If this number is reduced, bias increases.

to better exploit caching hierarchies of CPUs, at the cost of additional memory accesses to the random numbers. Consecutive paths will touch similar memory and thus employ the cache hierarchies of CPU systems better. On top of this, the shadow cache which can be used in the Quad-BVH [DHK08] will exploit this coherence to squeeze even more performance out of it for shadow rays.

In order to synthesize images using spectral transport, we combine all the above mentioned techniques and trace paths in the spectral domain. These are then projected to the tristimulus accumulation buffer. To transport spectral quantities with a path, we opted for the simplest and most general solution, and use Monte Carlo sampling to simulate the spectral domain continuously. While this increases the dimension of the sampling domain by one, and thus variance in the form of color noise, it has several advantages. Most importantly, we will find all important spots in the spectral domain with arbitrary resolution, as opposed to when using a fixed set of wavelengths (possibly due to stratification) or a finite basis. This makes sure we don't introduce any bias.

Concerning variance, the color noise will be more than can be perceived when only transporting RGB or XYZ values. But when importance sampling is done for

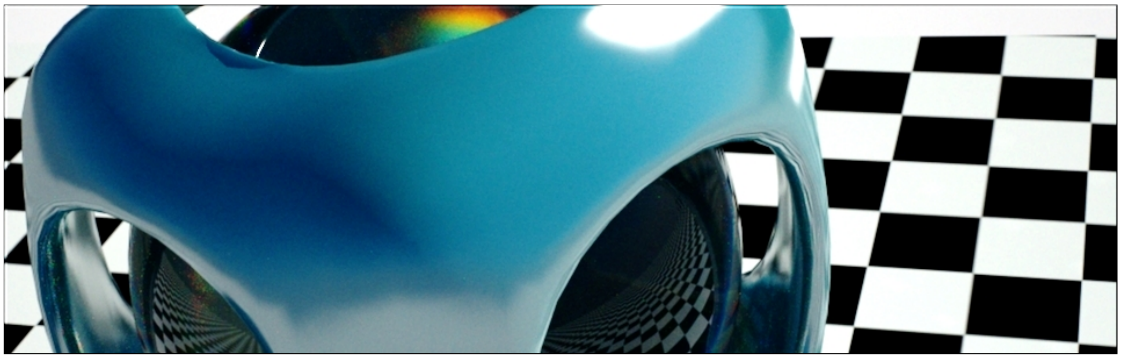
surface reflection, it can only be done perfectly for one wavelength. The other wavelengths will have to go with this sampling and correct the difference with a sample weight. This will in turn introduce additional variance, which increases when more basis functions are transported with a path at the time. Also, using RGB is prone to introduce additional bias due to the often vaguely defined color space, as well as possible gamut mapping to work around negative values. A schematic (not to scale) of these relations can be seen in Figure 2.5.

To make evaluation of the many samples required for a final image fast, various optimizations have to be applied, e.g. by making ray tracing itself faster [IWRP06, WH06, Wal07, EG07, WBB08, EG08, DK08, SFD09, DHKL09], by improving importance sampling [CJAMJ05, DH09], using stratification [GHSK08, DDK08] or by filtering the results [DSHL10]. For mono ray traversal in a realistic global illumination setting, there exists a test suite [RHF+07] which creates comparative overview statistics.

Parallelization via OpenMP, or even OpenMPI can be done on CPU architectures, if appropriate hardware is at hand [Qui03]. There has been a substantial amount of work on vectorization via CPU's SIMD units [Wal04, Ben04, BWSF06, Res07] as well as for GPU ray tracing [AL09, AK10, Lai10, PL10].

2.6. Conclusion

This chapter described the background needed to implement a physically-based spectral renderer, and some pointers how to optimize it for speed have been given. We summed up the basics of colorimetry, provided the notion of the tristimulus accumulation buffer, and complemented path space partitioning with photon map importance sampling and Metropolis light transport. Once everything is spectral, algorithms get simpler. For example there are no more for-all-colors loops, one can use simple, low-variance importance sampling, and color rendition becomes more accurate as metamerism is handled correctly. In contrast, RGB can have a fuzzy meaning, and the color matching functions can be negative. Due to the smooth XYZ color matching functions at the end of the spectral pipeline, the result does not include a lot of color noise. High-variance spots will turn up in heavily colored noise, but the variance will mostly stem from the difficult parts of the path besides the spectral domain. When used together in a rendering system, the techniques described here are able to robustly simulate spectral light transport.



"Note that the Ward and AS BRDFs are physically inspired hacks. So don't think too deeply about their motivation."

Peter Shirley, on ompf.org

3 Reflectance Models

This chapter investigates reflection properties of objects, with a focus on detailed color reproduction. A new lobe model which is defined on the projected hemisphere is investigated as a way to avoid rejecting samples during importance sampling and thus to avoid energy loss near grazing angles (Section 3.2). We will also fit analytical models to materials which have been acquired using sparse samples in the angular domain, but with a lot of samples in the spectral domain. The X-Rite MA98 is a standard device for quality control of car paint materials. It is a stripped down mini-gonioreflectometer and has been used to acquire the materials in this this chapter (Section 3.3).

3.1. Multi-Layer Material Models

Apart from geometry, the reflection properties at surfaces and in the medium make up a large part of the visual impression. This is why the bidirectional reflectance distribution function (BRDF) f_r receives some special attention in this section. In particular, when modeling a function f_r, it is necessary to fulfill a few properties to make it physically plausible.

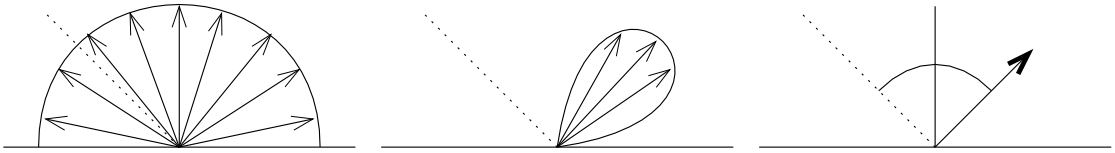


Figure 3.1.: Three types of reflection, from left to right: diffuse (e.g. Spectralon), glossy (e.g. polished metal), and specular (e.g. a mirror).

Energy Conservation. The surface should not produce energy. Thus, if no crosstalk between wavelengths is assumed, f_r is required to meet

$$\forall\omega\forall\lambda : \int_\Omega f_r(x,\omega,\omega_i,\lambda)L(-\omega_i,\lambda)|\langle n_x,\omega_i\rangle|d\omega_i < 1, \tag{3.1}$$

where the case of equality is not included so that the Neumann series (Equation (2.15)) converges, because the transport operator norm

$$\|T\| < 1 \tag{3.2}$$

$$\text{with } \|T\| = \inf\left\{c : \|TL\| \le c\|L\| \quad \forall L \in \mathcal{L}^2\right\}, \tag{3.3}$$

for all square integrable functions L, i.e. with finite energy.

Helmholtz Reciprocity is the property of surface reflection that light paths are reversible, i.e. light source and sensor can be exchanged and the same energy will be transported over a particular path. This property is often used to calibrate measurement devices. Formally, it can be written as

$$f_r(x,\omega_i,\omega_o,\lambda) = f_r(x,\omega_o,\omega_i,\lambda). \tag{3.4}$$

Types of reflection can roughly be classified into three categories: diffuse, glossy, and specular (see Figure 3.1). Since diffuse and ideal specular are the extremes of all possible glossy distributions in between, these are simple to model and sample. The shape of a glossy lobe leaves much more freedom and gives the possibility to match real materials.

There exist a number of analytic BRDF models used for rendering. Ideal specular surfaces such as dielectrics can be handled quite well by implementing Snell's law for refraction and using Fresnel's formulas to distinguish reflection and refraction, even considering polarization.

A glossy lobe based on micro-facet theory has been introduced by Cook and Torrance [CT81]. It assumes a rough surface is actually a collection of small mirroring facets and the lobe is formed by multiple reflection. Such a geometry can also been scanned using atom force microscopy and then simulated to create far-field BRDFs [Kem09].

Many BRDF models employ a halfway vector-based formulation (e.g. these more recent works [AS00, EBJ+06, GMD10, KSKK10]). These work by sampling a

halfway vector and using this to reflect the incoming ray as if the halfway vector was the normal. This can be seen as a trick to assert reciprocity of the model, and can be founded on micro-facet theory. A micro facet is picked randomly from a certain distribution, and it's normal is the halfway vector.

The 2d disk on the projected hemisphere has been used to define BRDF lobes [NNSK99] and 100% energy conserving models have been constructed [EBJ+06], at the cost of non-reciprocity. Data-driven BRDFs have been defined [MPBM03a] and there have been lots of extensions to include for example sub-surface scattering [JMLH01], rough glass, and variations on the surface [DWT+10].

3.1.1. A Multi-Layer Material for Car Paints

To express a car paint prior in form of a BRDF, we model it as a three-layer material: the primer is diffuse, on top of that follow a glossy layer of micro facets, and finally a clear coat layer. As the acquired data is very sparse in the angular domain (see Section 3.3), the choice of the glossy lobe is rather arbitrary, and no meaningful improvement in fitting error can be proved with another model, we chose an isotropic Phong lobe inspired by the anisotropic Phong model by Ashikhmin and Shirley [AS00]. To combine the layers, we use Schlick's approximation of the Fresnel terms [Sch94] and ignore the distance travelled in the medium between the layers.

Formally, the BRDF is written as a sum of three terms, diffuse, glossy, and specular:

$$f(\omega_o, \omega_i, \lambda) = f_{\mathsf{d}}(\omega_o, \omega_i, \lambda) + f_{\mathsf{g}}(\omega_o, \omega_i, \lambda) + f_{\mathsf{s}}(\omega_o, \omega_i, \lambda)$$

where ω_o is the direction towards the eye, and ω_i is the direction towards the light source. As we fit one lobe per wavelength, we omit the parameter λ for clarity from here on.

We use Schlick's rational approximation of the Fresnel term, which gives the amount of perfect reflection at angle θ from the normal with respect to a reflectivity ρ at normal incidence by the formula

$$F(\rho, \cos\theta) = \rho + (1-\rho)(1-\cos\theta)^5 \tag{3.5}$$

We denote $h = \frac{\omega_i + \omega_o}{\|\omega_i + \omega_o\|}$ as the halfway vector, and n as the surface normal. Note that naively splitting into layers using the Fresnel term can break energy conservation, at least for high albedos [SSHL97]. Given this, we define the three components of the BRDF:

Specular.

$$f_{\mathsf{s}}(\omega_i, \omega_o) = \frac{\delta(\omega_o, \omega_r(\omega_i))}{\langle \omega_i, n \rangle} F(\rho_{\mathsf{s}}, \langle \omega, h \rangle)$$

where δ is the Dirac distribution and $\omega_r(\omega_i)$ is the direction of perfect reflection of ω_i around the normal.

Glossy.

$$f_{\mathsf{g}}(\omega_i, \omega_o) = \frac{k+1}{8\pi} \frac{\langle n, h\rangle^k}{\langle h, \omega_i\rangle \max\{\langle n, \omega_i\rangle, \langle n, \omega_o\rangle\}} \cdot F(\rho_{\mathsf{g}}, \langle \omega_i, h\rangle) \cdot (1 - F(\rho_{\mathsf{s}}, \langle \omega, h\rangle)),$$

where the exponent k controls the glossiness.

Diffuse.

$$f_{\mathsf{d}}(\omega_i, \omega_o) = \frac{\rho_{\mathsf{d}}}{\pi} \cdot [1 - F(\rho_{\mathsf{g}}, \langle \omega, h\rangle) \cdot (1 - F(\rho_{\mathsf{s}}, \langle \omega, h\rangle))]$$

3.1.2. Simulating Scattering

In the Monte Carlo context, a BRDF model needs to be evaluated explicitly for deterministic connections and, more importantly, it should be possible to draw directional samples from a distribution as similar as possible to f_r. When f_r is proportional to the probability density function $p(\omega)$, variance will be minimal. If this is not possible, at least large parts of the BRDF should be sampled, for example a factor in a composite formula. Such a sampling should also be cheap to evaluate.

The mode of reflection, or the layer of the BRDF is chosen by importance sampling. As discussed in Section 2.4, we pick a continuous random variable for λ while spawning a new path and transport a single wavelength at a time. The following probabilities depend on spectral quantities and thus the importance sampling simplifies greatly. If more wavelengths were transported with one ray, importance sampling would have to be done for only one (or an average) of them, and the other channels would need to be multiplied by correction factors which would introduce additional variance.

Specular. Since in the case of perfect specular reflection, the normal and the halfway vectors are the same, it is possible to do perfect importance sampling by the specular Fresnel term $p_r = F(\rho_s, \langle \omega_i, n\rangle)$, resulting in a path weight

$$X = \frac{f}{p} = \frac{F(\rho_s, \langle \omega_i, n\rangle)}{p_r} = 1.$$

Glossy. We want to use the glossy Fresnel term for importance sampling similar to the specular case. Unfortunately, to guarantee reciprocity, this term is not determined by the incoming direction ω_i only. So in a best effort attempt, we sample the glossy part using

$$p_g = (1 - p_r) \cdot F(\rho_g, \langle \omega_i, n\rangle),$$

and account for this in the weight

$$X = F(\rho_g, \langle \omega, h\rangle)(1 - F(\rho_s, \langle \omega, h\rangle))/p_g$$

after the halfway vector has been sampled. This approach works better for higher k, and is optimal for the direction of perfect reflection. The rest of f_g not contained in X is sampled using halfway vector sampling proportional to cos^k.

Diffuse. This part takes what remains, i.e. $p_d = 1 - p_r - p_s$, and the weight has to be adjusted by $\frac{1}{p_d}$. The outgoing direction ω_o is sampled using a cosine distribution, resulting in

$$X = \frac{\pi \cdot f_d}{p_d}.$$

3.1.3. Probability Density Transformation

The reverse operation to importance sampling is also possible. We can evaluate a closed-form probability density function $p^y(y)$ for a random variable y which has been obtained by transforming an initial random variable x with given probability density $p^x(x)$ by a bijection $f : x \mapsto y$:

$$p^y(y) = \frac{p^x(f^{-1}(y))}{|J_f(f^{-1}(y))|} \tag{3.6}$$

where J_f denotes the Jacobian of the function f, and $|.|$ the determinant. This relation can be explained e.g. by a comparison of the integrand in this example:

$$\begin{aligned} P(y \in \Omega_y) &= \int_{y \in \Omega_y} p^y(y) dy && (3.7) \\ &\quad \text{subst. } y = f(x) \\ &= \int_{x \in \Omega_x} \underbrace{p^y(f(x))|J_f(x)|}_{=p^x(x)} dx. && (3.8) \end{aligned}$$

This can be used to derive the density of ω_o given the density of the halfway vector h and the incoming direction ω_i, as used in many popular BRDF models such as [AS00], and geometrically explained in [Ren50]:

$$\begin{aligned} f(\lambda, \psi) &= \left(2\lambda, \psi \frac{\sin 2\lambda}{\sin \lambda}\right) = (2\lambda, 2\psi \cos \lambda) \\ J_f &= \begin{pmatrix} 2 & 0 \\ -2\psi \sin \lambda & 2 \cos \lambda \end{pmatrix} \\ |J_f| &= 4 \cos \lambda \\ \Rightarrow p(\omega_o) &= \frac{p(h)}{4 \cos \lambda} = \frac{p(h)}{4\langle h, \omega \rangle}. \end{aligned}$$

For an illustration of the variables, see Figure 3.2. The angles λ and ψ define the distance between ω_i and h. The output direction ω_o is obtained by reflecting ω_i about h, as formally expressed in the function f. The density transformation makes sure that the integral over the transformed probability $\int p(\omega_o) d\omega_o$ is still normalized.

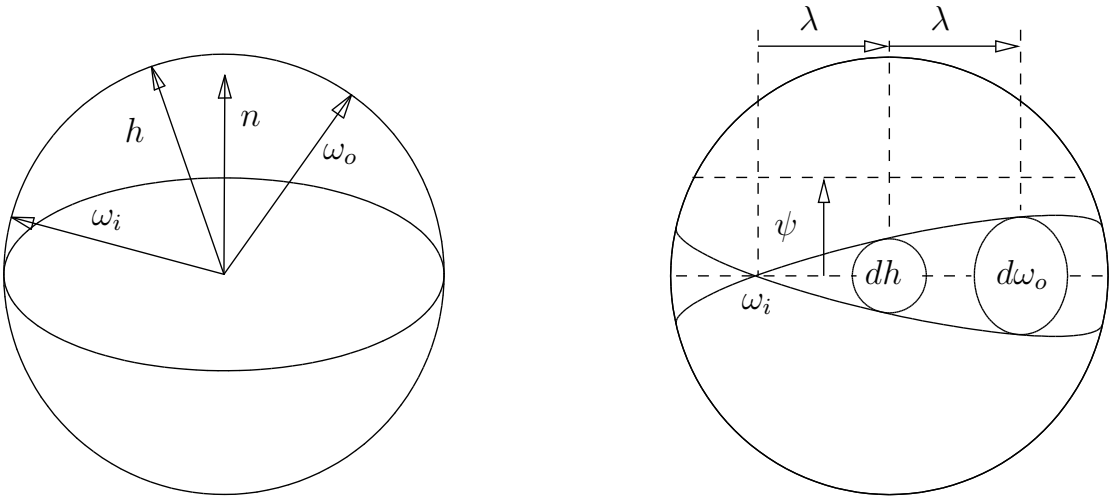


Figure 3.2.: The density $p(\omega_o)$ viewed as the transformed halfway vector density $p(h)$: the distance between ω_i and h can be expressed in the domain of the angles λ and ψ.

3.2. BRDF Lobes as Automorphisms on the Unit Disk

Most Phong-like and halfway vector-based analytical BRDF models suffer from energy loss near grazing angle, when up to half of the lobe points under the surface. This behavior is also very undesirable when importance sampling by this lobe is done. Since directions have to be rejected if they point under the surface, the procedure becomes very inefficient.

One strategy to create new energy conserving lobe models is to start with the sampling procedure, make sure no directions have to be rejected, and then create a closed-form expression for the distribution function f_g, as has been done in [EBJ+06] for halfvector-based BRDFs.

Consider the following general conformal mapping on the unit disk

$$f : z \mapsto w, \quad |z| < 1, |w| < 1 \tag{3.9}$$

$$w = k\frac{z - z_0}{\bar{z}_0 z - 1}, \tag{3.10}$$

where $z, w, k, z_0 \in \mathbb{C}$, $|k| = 1$ and $|z_0| < 1$ is the point the origin gets mapped to. All conformal automorphisms on the unit disk take this form [Jef05, Bie00]. Without the rotation (i. e. $k = 1$), it is sometimes called the Möbius transformation. The inverse Möbius transformation is given by

$$f^{-1}(w) = \frac{z_0 - w}{1 - \bar{z}_0 w}, \tag{3.11}$$

and the complex derivative ($f'(z) = \frac{df}{dx}$ since conformal mappings are analytic, i.e. differentiable in the complex sense):

$$f'(z) = \frac{\bar{z}_0 z_0 - 1}{(z\bar{z}_0 - 1)^2}. \tag{3.12}$$

Applying this transformation to a uniform distribution $p^z(z) = \frac{1}{\pi}$ on the unit disk and inserting it in Equation (3.6) yields the following density:

$$p^w_{z_0}(w) = \frac{p^z(f^{-1}_{z_0}(w))}{|J_f(f^{-1}_{z_0}(w))|} = \frac{1}{\pi}\left|\frac{(z\bar{z}_0 - 1)^2}{|z_0|^2 - 1}\right|^2, \tag{3.13}$$

where we used the fact that conformal mappings satisfy the Cauchy-Riemann equations and thus $|J_f| = |f'(z)|^2$ and the shortcut $z = f^{-1}(w)$.

We can use this transformation to define a closed-form lobe model. The incoming direction ω_i is first projected down to to unit disk and used to define the parameter z_0. Then a point w on the disk is sampled and an outgoing direction ω_o is constructed from it by adding the third component to the vector such that it lies on the upper unit hemisphere. Since the density $p^w_{z_0}(w)$ is based on sampling the unit disk followed by an automorphism on the disk, the resulting lobe is 100% energy conserving, i.e. no samples are discarded due to invalid directions such as under the surface.

To create a physically plausible lobe from Equation (3.13), we want the outgoing directions to be centered around the specular reflection, i.e. we choose $z_0 = -w_i$ where w_i is the complex projection of the incoming direction ω_i: $w_i = \langle\omega_i, a\rangle + i\langle\omega_i, b\rangle$, where the vectors a, b and the normal form an orthonormal basis at the surface point.

We can test this formula for Helmholtz reciprocity:

$$p^w_{-w_i}(w_o) = \frac{1}{\pi}\left|\frac{(-w_o\bar{w}_i - 1)^2}{|w_i|^2 - 1}\right|^2 \neq \frac{1}{\pi}\left|\frac{(-w_i\bar{w}_o - 1)^2}{|w_o|^2 - 1}\right|^2 = p^w_{-w_o}(w_i), \tag{3.14}$$

which, unfortunately, is not true in general. Since the asymmetry is intrinsic to the derivative $f'(z)$, it is very hard to construct reciprocal formulas on the base of a Möbius transform (using the inverse as initial distribution would work).

To control the appearance of the lobe, a cosine to the power of k distribution can be used as initial distribution. After the Möbius transform, the following density results (note that $w, z, z_0 \in \mathbb{C}$):

$$p(\omega_o) = \frac{\sqrt{1 - |z|^2}^{k-1}(k+1)}{2\pi}\left|\frac{(z\bar{z}_0 - 1)^2}{|z_0|^2 - 1}\right|^2 \tag{3.15}$$

$$\text{with } w = \langle\omega_o, a\rangle + i\langle\omega_o, b\rangle, \tag{3.16}$$

$$z = \frac{z_0 - w}{1 - \bar{z}_0 w}, \tag{3.17}$$

$$\text{and } z_0 = -(\langle\omega_i, a\rangle + i\langle\omega_i, b\rangle), \tag{3.18}$$

$$f_g = \rho \cdot p(\omega_o), \tag{3.19}$$

```
static void
moebius_sample(const float *omega_in, float *omega_out)
{
  // (a,b,n) form the local coordinate frame
  float complex z0 = dot(omega_in, a) + I*dot(omega_in, b);
  // sample z ~ cos^k on disc from x1, x2 uniform in [0,1):
  float r1 = x1 * 2.0f * M_PI;
  float cos_theta = powf(1.0f - x2, 1.0f/(k+1.0f));
  float sin_theta = sqrtf(1.0f - cos_theta*cos_theta);
  float complex z = cosf(r1)*sin_theta + I*sinf(r1)*sin_theta;

  // moebius transform w = f(z):
  float complex w = (z - z0)/(conjf(z0)*z - 1);

  // store in outgoing direction:
  float dir[3] = {crealf(w), cimagf(w),
    sqrtf(1.0f - cabsf(w)*cabsf(w))};
  for(int i=0;i<3;i++)
    omega_out[i] = a[i] * dir[0] + b[i] * dir[1] + n[i] * dir[2];
}
```

Figure 3.3.: Pseudo code for the Möbius lobe sampling, using c99 complex syntax. All temporary variables could have been declared `const`, which is omitted for brevity.

where again $a, b, n \in \mathbb{R}^3$ form an orthonormal basis around the surface normal n. The lobe f_g can then be fine tuned by a another spectral parameter ρ to make the surface darker and colored.

Pseudo code to sample from such a condensed lobe can be found in Figure 3.3, and the direct evaluation is shown in Figure 3.4.

3.2.1. Photon Map Importance Sampling

In Section 2.4, we mentioned the use of *photon map importance sampling* as technique to effectively reduce variance for a part of the path space, e.g. paths that reflect caustics seen through a mirror. This technique works by sampling outgoing directions around a preferred direction, where light is known to come from (see Figure 2.4). This is quite similar to importance sampling of BRDF lobes, and in fact the best existing implementation [Pha05] so far uses a Phong lobe to create directions around the incoming direction of the stored photon. This approach has

```
// omega_in pointing toward hitpoint, omega_out away from it.
// includes f_r * cos
static float
moebius_lobe(const float *omega_in, const float *omega_out)
{
  // (a,b,n) form the local coordinate frame
  float complex w  = dot(omega_out, a) + I*dot(omega_out, b);
  float complex z0 = dot(omega_in,  a) + I*dot(omega_in,  b);
  float complex z = (z0 - w)/(1.0f - conjf(z0)*w);
  float complex dfdx = (z*conjf(z0) - 1.0f)*(z*conjf(z0) - 1.0f)/
    (conjf(z0)*z0 - 1.0f);

  // make sure float doesn't kill sqrtf:
  float tmp = 1.0f - cabsf(z)*cabsf(z);
  if(tmp <= 0.0f) return 0.0f;

  // also apply cos^k:
  return powf(sqrtf(tmp), k-1.0f) * (k+1.0f)/(2.0f*M_PI) *
    (dfdx * conjf(dfdx));
}
```

Figure 3.4.: Pseudo code for the Möbius lobe evaluation, using c99 complex syntax. All temporary variables could have been declared `const`, which is omitted for brevity.

two problems: firstly, the Phong lobe will result in directions under the surface, which have to be rejected. Thus great care has to be taken to avoid unknown ray densities resulting in wrong Monte Carlo weights. The second problem is that the Phong lobe will not cover the entire incoming hemisphere at the surface point. To obtain an unbiased estimator it is thus necessary to use diffuse hemisphere sampling as second technique and combine the two by multiple importance sampling. Using a Möbius lobe instead solves both these problems, as they originate from the mismatch between the hemisphere that is intended to be sampled (incoming at point x) and the one that is actually sampled (centered around the direction of the stored photon). For grazing angles, this previous approach can be as bad that every second sample has to be discarded because the lobe is halfway under the surface.

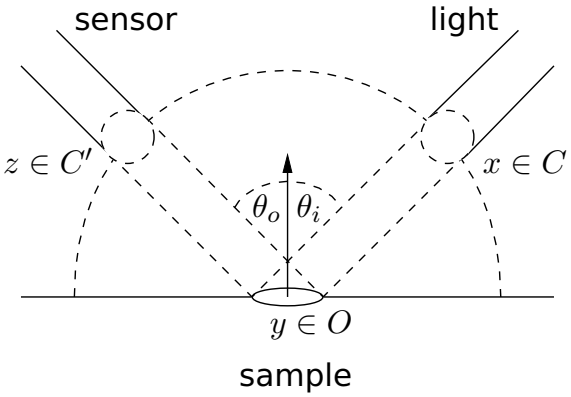


Figure 3.5.: The MA98 device geometry. Light travels from points $x \in C$ over $y \in O$ to $z \in C'$, with angles θ_i and θ_o between directions to the centers of the two circles C, C' and the normal.

3.3. BRDF Parameters from Sparse Data

We want to acquire data with the X-Rite MA98 portable multi angle spectrophotometer [XR10] and use it for rendering. The acquired data is dense in the spectral domain (31 wavelength samples from 400nm–700nm), while the angular domain is sampled very sparsely (ten outgoing directions for light positions at $45°$ and $15°$). The question is, whether a credible angular behavior can be achieved from this data, if an appropriate BRDF model is used for fitting. As the device is mostly targeted towards car industry, we use the car paint-like multi layer material.

3.3.1. Sparse Data Acquisition

The geometry of the apperatus is illustrated in Figure 3.5. The measurements are averaged over an oval O of about 12.5 mm^2 on the sample, illuminated through a circular aperture C of about 11mm^2. Light travels from points $x \in C$ on the light over points $y \in O$ to points $z \in C'$ on the sensor, with the same circular aperture as the light. Following the rendering equation (2.13) formulated as integral over geometry, the sensor response can be written as

$$\begin{aligned} S &= \int_C \int_O \int_{C'} \frac{\cos\theta_x \cos\theta_{xy}}{\|x-y\|^2} \cdot V(x,y) \cdot f_r(x,y,z) \cdot \\ &\quad \frac{\cos\theta_{yz} \cos\theta_z}{\|y-z\|^2} \cdot V(y,z) dx dy dz \qquad (3.20) \\ &= \int_C \int_O \int_{C'} \frac{\cos\theta_{xy}}{\|x-y\|^2} \cdot f_r(x,y,z) \cdot \frac{\cos\theta_{yz}}{\|y-z\|^2} dx dy dz. \qquad (3.21) \end{aligned}$$

In order to extract an approximate BRDF, we replace the evaluation at x, y, z by the center points of the circles and the oval $\bar{x}, \bar{y}$, and $\bar{z}$, respectively. Setting

$\|x-y\| = \|y-z\| =: d$, we can pull f_r out of the integral and solve for it:

$$S \approx \frac{1}{d^4} \cos\theta_i \cdot f_r(\bar{x},\bar{y},\bar{z}) \cos\theta_o \int_O dy \int_C dx \int_{C'} dz \tag{3.22}$$

$$= \frac{A(C)^2 A(O)}{d^4} \cos\theta_i \cdot \cos\theta_o \cdot f_r(\bar{x},\bar{y},\bar{z}) \tag{3.23}$$

$$\Rightarrow f_r(\bar{x},\bar{y},\bar{z}) = S \cdot \frac{d^4}{A(C)^2 A(O) \cos\theta_i \cdot \cos\theta_o}. \tag{3.24}$$

Equation (3.24) shows that f_r can be obtained by a simple scaling of the sensor response S, as even the cosine terms can be assumed to be constant for a given light/sensor angular configuration. Calibrated sensor output $M(\omega_i, \omega_o)$ is normalized so that the almost Lambertian material Spectralon results in $M(\omega_i, \omega_o) = 99.1$ for all directions. To transfer those measurements to valid BRDF values, we therefore need to divide them by 100π.

Unfortunately the assumption of local angular invariance does not hold for sharp glossy lobes or specular reflection. For our car paint like materials we have posed the restriction $k < 500$, since sharp lobes cannot be captured by the measurement device and will cause bogus fittings. But the clear coat layer exhibits strong specular reflection and needs special care during fitting. One further measurement gives an indication of the behavior of specular material in the device: a near perfect mirror resulted in $M(\omega_i, \omega_r) = 30,000$.

3.3.2. Metropolis Fitting

We fit one set of parameters (ρ_s, ρ_g, ρ_d, and k) per measured wavelength $\lambda \in [400nm, 700nm]$. This is done by initially evalutating the mean square error E of the layer BRDF defined by the parameters $\rho_s = 0, \rho_g, \rho_d$, and k and the measured data

$$E = \sum_j \left(f_d(j) + f_g(j) + f_s(j) - \frac{M(j)}{100\pi} \right)^2, \tag{3.25}$$

where j enumerates all light/sensor configurations (ω_i, ω_o) but the perfect reflection.

A major problem is that the value of ρ_s is not available, since it can only be reliably guessed from one single measurement. We work around this by evaluating the difference of the fitted BRDF and the measurement value at the specular configuration, and attribute this to the clear coat layer:

$$F\left(\rho_s, \cos\frac{\pi}{4}\right) = \max\left\{0, \frac{1}{30,000}\left(M(j) - 100\pi(f_d(j) + f_g(j))\right)\right\} \tag{3.26}$$

with $j = (\omega_i, \omega_r)$ this time. Given that, we use a simple look-up table-based numerical inversion of the Fresnel term F (Equation (3.5)) to extract ρ_s and re-evaluate the fitting error E with this value.

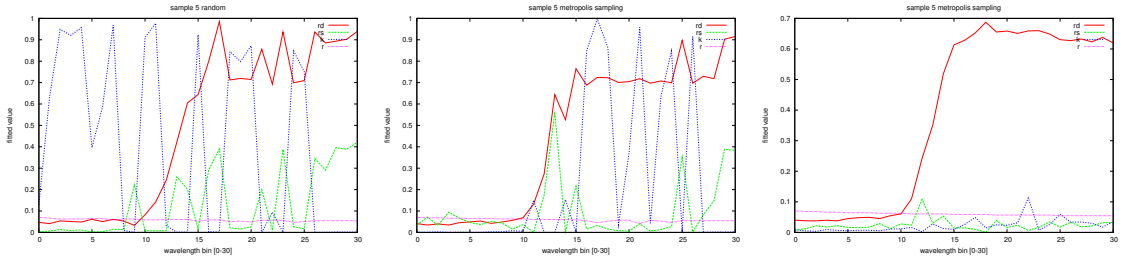


Figure 3.6.: Comparison of the different fitting methods. From left to right: random, Metropolis sampling, Metropolis sampling with smoothness term. The curves labeled rd, rs, k, and r correspond to the parameters ρ_d, ρ_g, k, and ρ_s, respectively. Note the different scale in the last one, as high random peaks are smoothed out.

To fit a BRDF which integrates nicely over wavelength, the parameters should form a smooth spectrum. Even though the material itself might not exhibit a smooth spectral distribution, this is desirable behavior as for example the roughness of the lobe is more consistent over the wavelengths. This can be enforced by adding a small penalty value to the error term, consisting of the difference to the parameters fitted to the previous wavelength sample:

$$E^* = E + \alpha \cdot \|P_{\lambda_i} - P_{\lambda_{i-1}}\|^2. \tag{3.27}$$

We used $\alpha = 0.05$ in all fittings.

With this error measure at hand, we explore the search space by an adjusted version of simulated annealing [KGV83]. More precisely, we use Metropolis-Hastings sampling to get samples with a distribution proportional to the reciprocal mean square error. This way, parts of the search space with a good fitting are explored in detail using small mutations. The inherent ergodicity makes sure the algorithm is not trapped in local minima. During this random walk, the sample with least mean square error E is kept as output.

A particular strength of this approach is that difficult situations like small peaks are handled well and no assumptions about the error function have to be made (such as e.g. differentiability).

The algorithm proceeds as in [MRR+53, Has70], but the acceptance probability for a tentative sample is replaced by

$$a = \min\left\{1, \frac{E^*_{state}}{E^*_{tent}}\right\}, \tag{3.28}$$

where E^*_{state} and E^*_{tent} are the mean square errors of the current state and the tentative sample, respectively.

Summing up, for each material do in parallel:

```
last_params = 0
for samples
{
  if large step
    params = rand
  else
    params = mutate(curr_params)

  m = error(params) + smoothness * sqdist(params, last_params)
  if m == new minimum, record parameters

  if rand < curr_m/m
  {
    curr_params = params
    curr_m  = m
  }
}
last_params = params
```

3.4. Results

Plots of the Möbius BRDF model can be seen in Figures 3.9 and 3.10. A combined multi-layer behavior can be observed: the lobe will always degenerate to a mirror-like specular behavior near grazing angle, producing an effect as if the paint was covered with a clear coat layer. This is also the reason why the BRDF formulation is unable to capture diffuse surfaces. The Möbius transform takes place regardless of the Phong exponent, and thus even a uniform input distribution will exhibit the Fresnel-like effect.

As a comparison with the Phong-based three-layer BRDF from Section 3.3, we replaced the micro-facet layer with a Möbius lobe. A few rendered results can be seen in Figure 3.11. These renderings have been obtained using partitioned path tracing, i.e. the path space has been partitioned into caustic paths (evaluated using light tracing) and eye-paths (evaluated using path tracing with next event estimation), to make sure the algorithm converges even using a non-reciprocal BRDF. The 100% energy conserving property gives a bright look near the object boundaries as well as in the cavities between the diffuse sphere in the middle and the sample material.

To asses the performance of the lobe with respect to how it represents real materials, we fitted it to Matusik et al.'s data sets [MPBM03a] and compared it to a standard reciprocal and energy conserving halfway vector-based variant of the Phong model. The results of these fittings can be seen in Figure 3.12. While

the absolute scale of the peak values of the BRDF lobe is represented quite well by the Möbius lobe, the shape of the lobes are not quite as well matched as by the Phong lobe. This is due to the behavior of micro-facet reflectance distributions, which exhibit a flatter and wider lobe near grazing angle, due to the larger projected differential surface $dA/\cos\theta$ which covers a broader range of micro-facet orientations. This is reflected by the density transformation term $1/(4\cos\theta)$ in the halfway vector Phong model.

For an idea of the effectiveness of the fitting algorithm, see Figure 3.6. The graphs show the fitted parameters for each wavelength for sample number 5 (for photographs and renderings see the third row in Figure 3.7). Pure random sampling exhibits a lot of noise over the wavelength domain, which indicates the global minimum has not yet been found. Metropolis sampling eliminates much of this noise but the algorithm doesn't seem to be sure whether to attribute the energy near the reflection direction to a near-specular lobe (k) or to the clear coat layer (ρ_s). The smoothness constraint takes care that this decision is taken consistently over all wavelengths.

In practice, it proved sufficient to use not more than about 10,000 runs per material to obtain a satisfactory fitting, which is of course not guaranteed to reach the global optimum. In an unoptimized CPU implementation, we fitted 30 materials in about a minute on a quad core computer.

As the sparse angular sampling makes numerical error analysis only hardly interesting, we only give visual comparisons of photographs and renderings of the fitted BRDF. Figures 3.7 and 3.8 show a selection of car paint samples with micro sparks and uniform paint in various colors.

3.5. Conclusion

We followed a general approach to create new BRDF models by first coming up with an algorithm for importance sampling new outgoing directions. Through density transformation, the resulting probability density function can be given in closed form, so that evaluation of the BRDF is possible. One advantage of this approach is that 100% energy conserving models can easily created, a drawback is that it is unclear if reciprocity can be ensured without energy loss.

As one instance, we investigated the use of the Möbius transformation together with a cosine distribution to create a BRDF lobe which is strictly defined inside the projected hemisphere, i.e. never samples invalid ray directions. This model is 100% energy conserving, but doesn't fulfill the principle of Helmholtz reciprocity.

We also fitted analytical models to sparsely measured data. While it is definitely not possible to capture the exact angular behavior of materials using the approach pursued in this section, faithful color reproduction as well as a credible fitting to a BRDF model including prior knowledge about the class of measured materials can be achieved. One might argue that the simple analytical models commonly used in computer graphics don't fit the behavior of real materials very well anyways

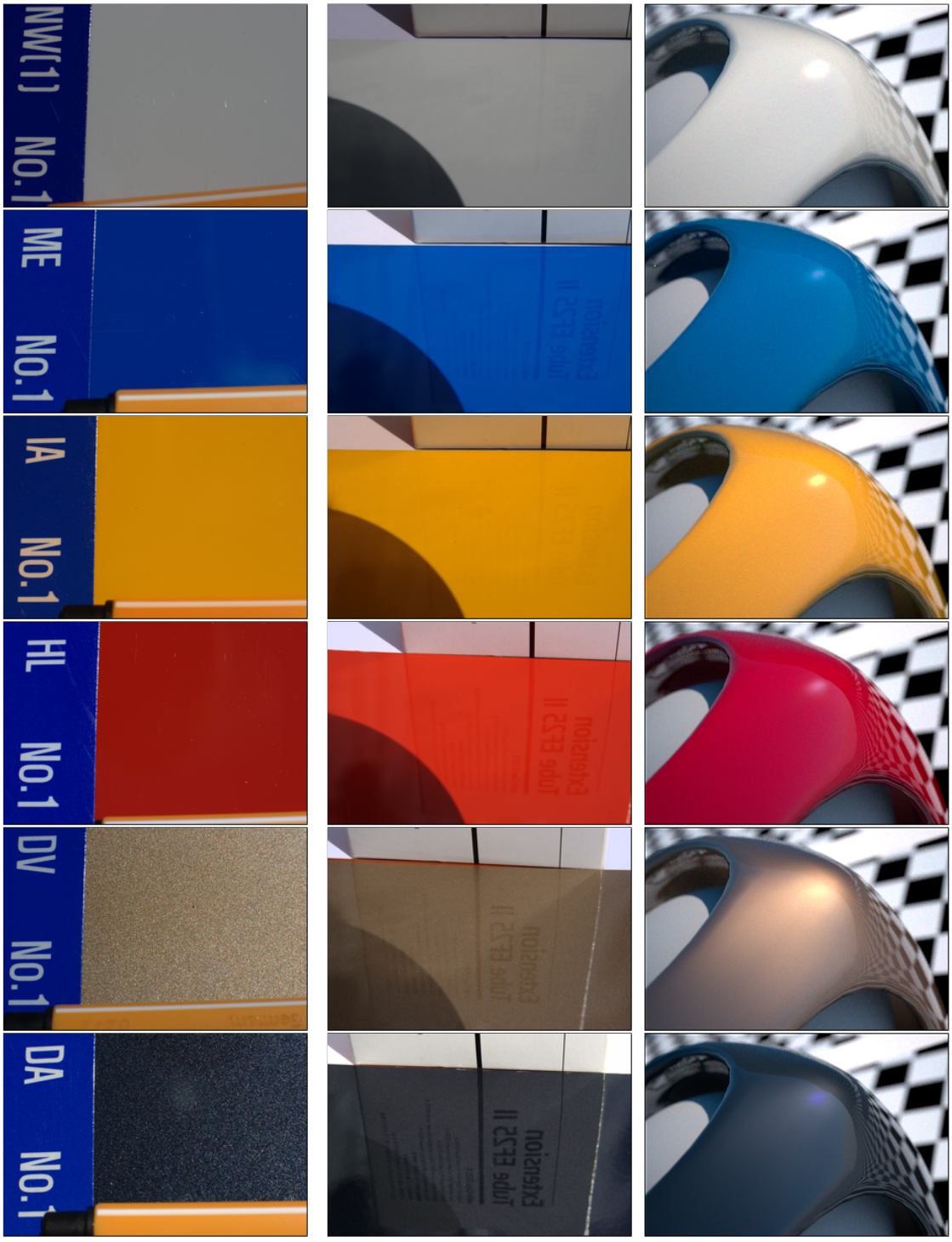


Figure 3.7.: Visual comparison of rendered images of the fitted three-layer BRDF (right) with photographs (left and middle columns). The photographs have been taken in direct sunlight, and the renderings have been illuminated by the reference solar spectral irradiance [Ame]. This figure is continued in Figure 3.8.

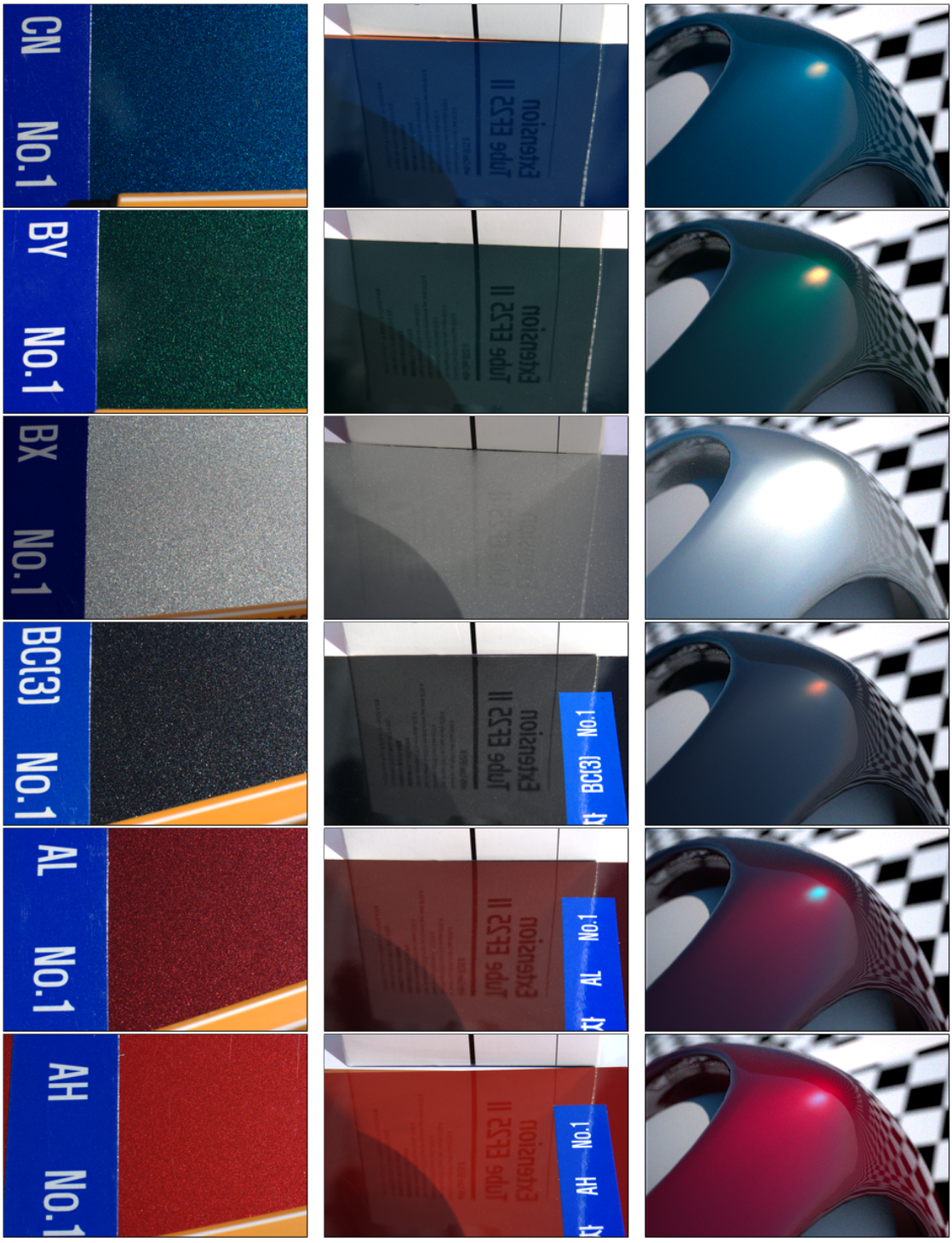


Figure 3.8.: Continuation of Figure 3.7.

(see for example the supplemental material of [NDM05]), so it is questionable how much better one can do with a more complex measurement device while still fitting the same BRDF model.

When not used as BRDF lobe, the property of the Möbius lobe to be nicely restricted to the hemisphere while still being non-zero everywhere on this domain can be used for other applications such as simplifying photon map importance sampling (see Section 2.4) or to construct radial basis functions on the hemisphere for scattered data interpolation.

These techniques contribute to robust rendering in two ways. First, spectral acquisition and rendering of reflection properties robustly reproduce color. Second, a function which creates samples exactly on the hemisphere can be used to create robust sampling strategies for BRDF lobes, as well as for importance sampling.

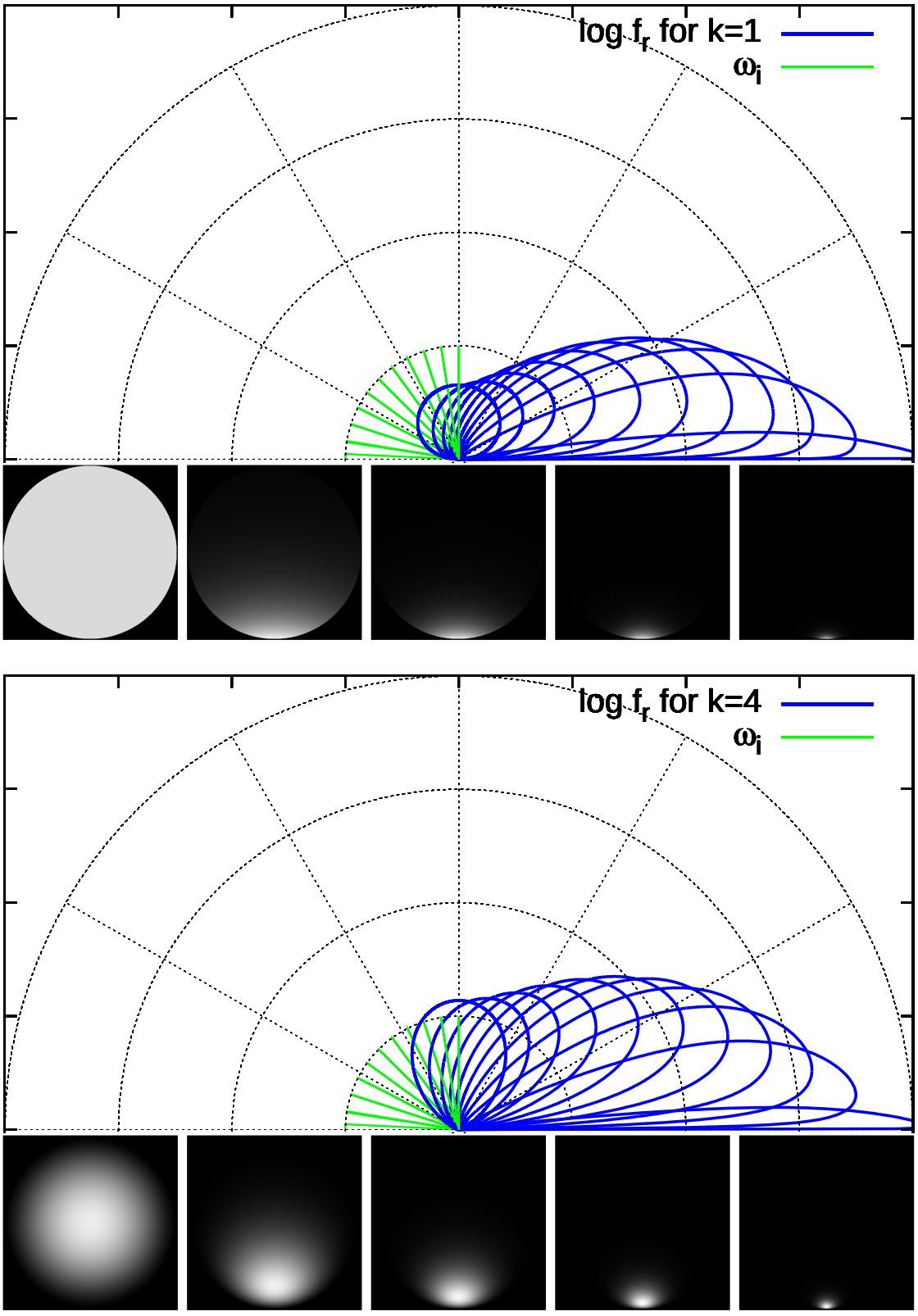


Figure 3.9.: The Möbius lobe for $k = 1$ and $k = 4$ and various incoming directions ω_i. The top row is a polar plot of a slice through the BRDF. To compress the values, the graph shows the logarithm of f_r. The bottom row shows the sampling density $p(\omega_o) = f_r \cdot \cos\theta_o$ on the projected hemisphere, for visualization purposes normalized to the maximum.

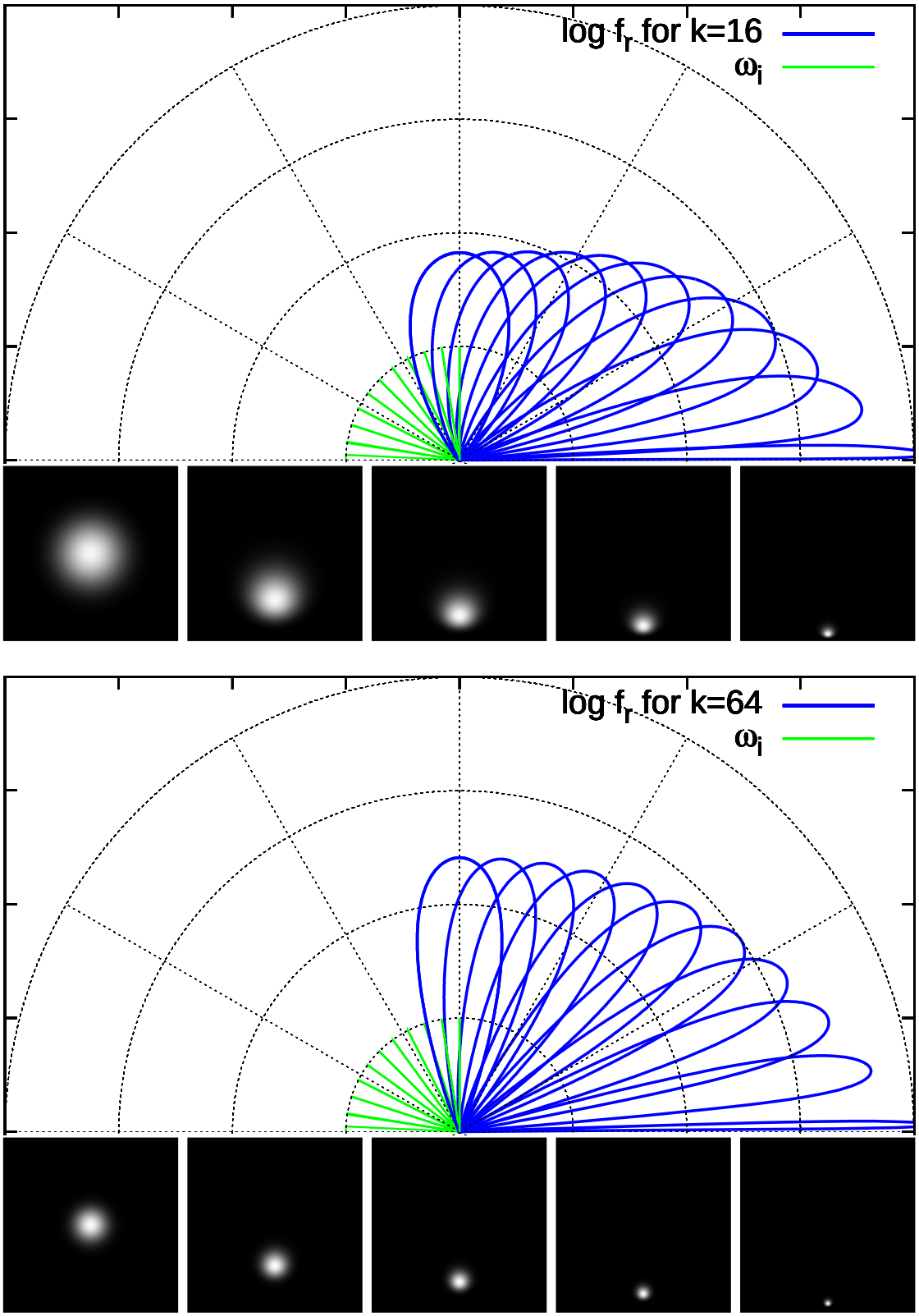


Figure 3.10.: This shows the same information as Figure 3.9, but for $k = 16$ and $k = 64$. Note how the lobe approaches a specular behavior with larger k as well as for grazing angles, while the lobe will always stay above the surface.

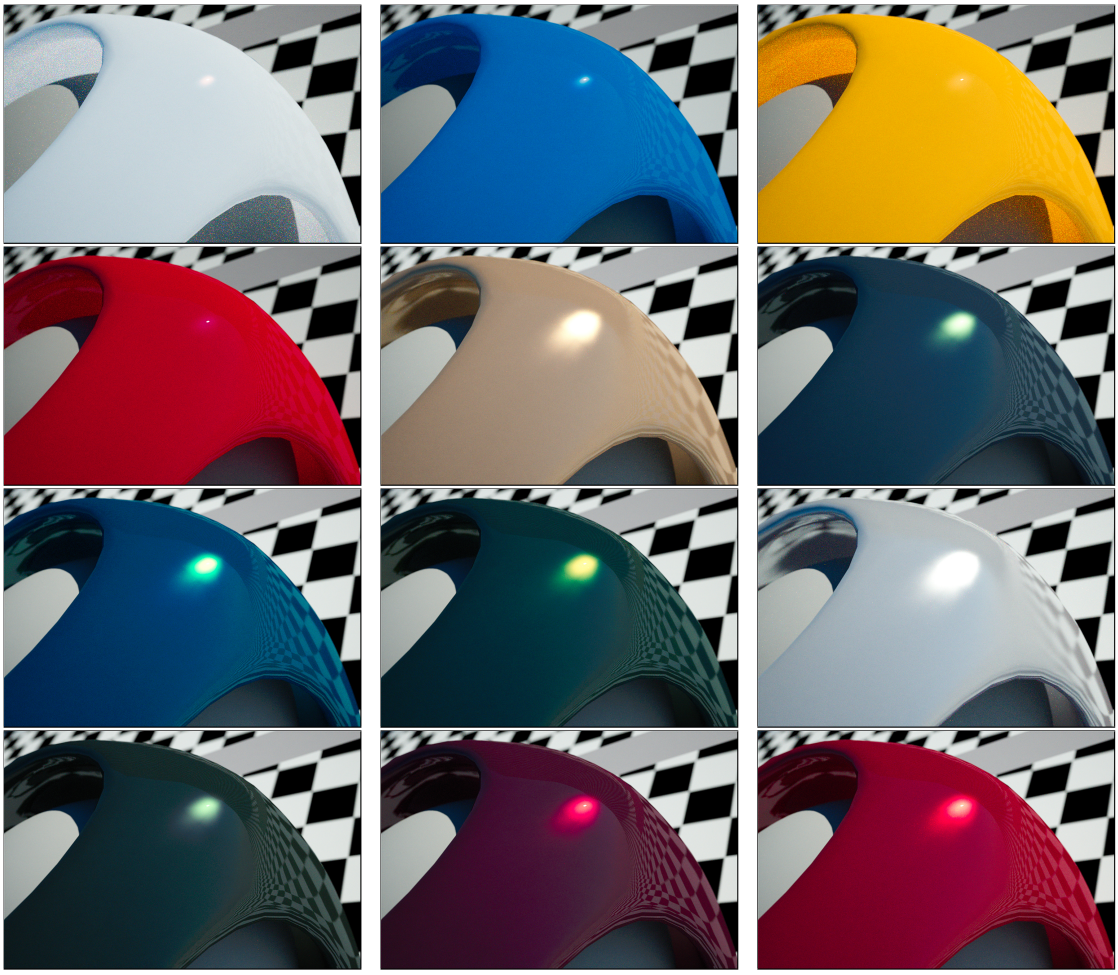


Figure 3.11.: These images show some of the samples from Section 3.3 rendered with Möbius lobes fitted to the data in the three-layer BRDF instead of the Phong model. This results in a different look, most notably the boundaries are brighter and the third image in the third row (sample 17, compare to the third row in Figure 3.8) nicely shows the smooth transition from glossy to specular for greater incoming elevation angles.

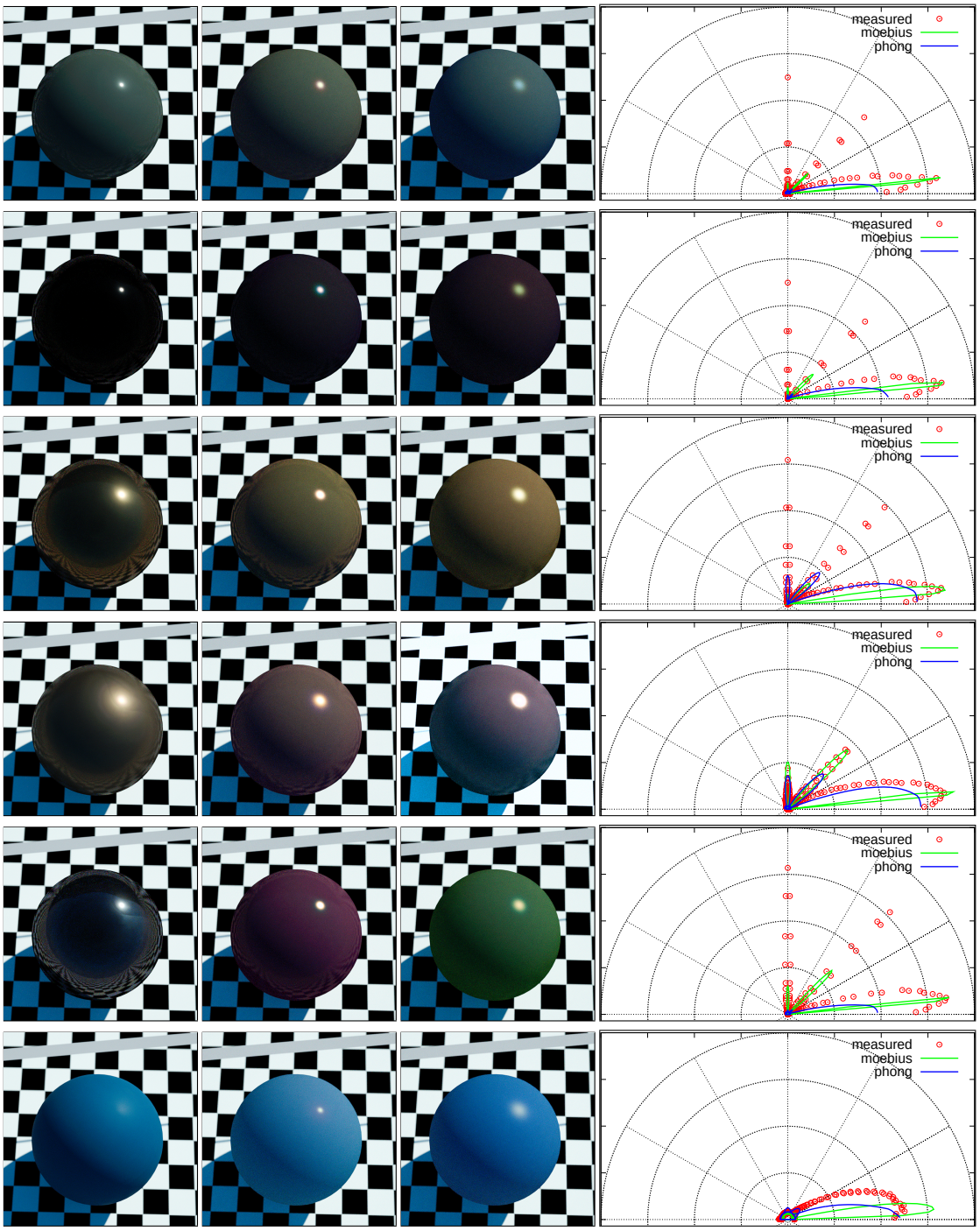
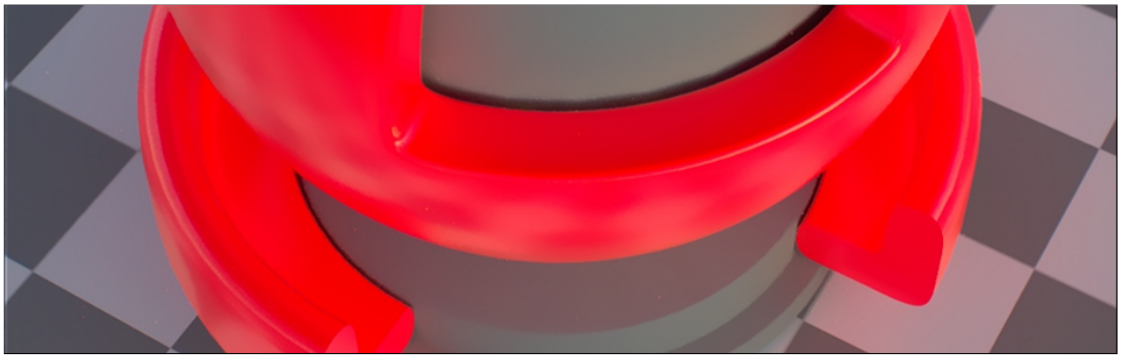


Figure 3.12.: Plots of fitted Möbius and Phong lobes in the three-layer BRDF. The Möbius lobe is in some cases able to fit the maximum peaks better than Phong (as in the first three rows: aventurnine, black obsidian, and brass), but fails to match the shape of the lobe (especially apparent in row four: alum-bronze). Row five (chrome-steel) is not represented well at all in the three-layer BRDF, so fitting results in random colors for both lobes. The last row shows an almost diffuse material (blue-rubber), where the energy conserving property of the Möbius lobe actually makes the shadows too bright.



“Es ist doch gerade die unendlich lange Kette wirrer Verzweigungen und Zufälle? die uns so fasziniert und dann gibts auch noch ein schönes Bild .. !”

Dr. Quade

4 Simulating Fluorescence

In fluorescent materials, light from a certain band of incident wavelengths is reradiated at longer wavelengths, i.e., with a reduced per-photon energy. While fluorescent materials are common in everyday life, they have received little attention in computer graphics.

Fluorescent materials change the wavelength of light upon reflection. This applies to many everyday materials, for instance human teeth, utility vehicle paints, detergents (fabric whiteners), or even ordinary photocopying paper. This shift of wavelength causes compelling visual effects if it occurs within the visible spectrum or turns UV radiation into visible light. In particular, many fluorescent surfaces appear brighter than perfectly white surfaces.

The underlying physical mechanism is well understood. A fluorescent medium consists of atoms or molecules that absorb incident photons at a given wavelength, and re-emit them after a short time (in the order of 10^{-8} s). During this time interval, the electrons of the fluorescent molecule remain in an excited state above the ground energy level. The re-emission of a photon occurs as the fluorophore relaxes to its ground state. Due to mechanical interaction with the surrounding molecules, some of the excitation energy is lost during this process, leading to a change of wavelength, or Stokes shift. As required for conservation of energy, except in the case of multi-photon interactions, this shift always occurs towards

longer wavelengths, corresponding to a loss in per-photon energy.

We include fluorescence into light transport simulation as an advanced spectral effect (Section 4.1), verify our simulation against measurements, and show an application of volumetric fluorescence from solar cell research. In Section 4.2, a combined way to express reflection and fluorescence at the surface is introduced and formalized for the first time, and measurements as well as renderings of such materials are discussed. This section is joint work with Matthias Hullin, who is the expert on the measurement side, whereas I joined the project mainly for the bispectral rendering. Both projects are rendered in the same framework.

Related Work. There are several publications about simulation of fluorescent light in various fields of science, for example by Welch et al. [WGRK+97], and Susila and Naus [SN07]. In computer graphics very little research in this domain has been done. Glassner [Gla94] presented a formulation of the rendering equation including phosphorescence and fluorescence, i.e., a mathematical model for global energy balancing which includes these phenomena. After his work, apparently there has not been any further investigation of fluorescence phenomena in computer graphics until 2001. Wilkie et al. implemented a rendering system including fluorescence and polarization using Stokes vectors and Müller matrices [WTP01]. In 2006 they provided an analytical BRDF model for fluorescent surfaces [WWLP06], but this approach only works for a finite set of wavelengths.

Heidler [Hei82] developed a Monte Carlo model for fluorescent concentrators in 1982. His model was made for an efficiency analysis for the concentrator. Carrascosa et al. [CAU83] were first to describe ray tracing of fluorescent concentrators. In the last few years Burgers et al. [BSKvR05, BSBvR06] and Schüler et al. [SKG+07] used Monte Carlo ray tracing for simulations of fluorescent concentrators and quantum dot solar concentrators, respectively.

The history of data-based BRDF models in the context of computer graphics goes back to the early nineties, when Ward [War92] measured and modeled the BRDF of anisotropic materials. The first larger material database of 61 different, albeit sparsely sampled BRDFs emanated from the CUReT project [CUR96]. Later on, Matusik et al. [MPBM03a] measured more than 100 different materials (similar setup to Marschner et al. [MWLT00]), from which they derived a generative BRDF model. Ngan et al. [NDM05] compiled an overview of different models and how well they approximate BRDF measurements. Many of these BRDF models allow for spectrally varying reflectance distributions, such as the Cook-Torrance BRDF [CT81], but do not model bispectral distributions as needed for the reproduction of fluorescence.

Within the field of fluorometry, bispectral measurements are a long-established technique [LJA97]. In fact, the concept of a reradiation matrix dates back over half a century [Don54]. Due to the high dimensionality of the reflectance and reradiation function, researchers usually put more focus on the spectral dimension and constrained themselves to very sparse angular sampling of BRDFs, typically

at $0°/45°$ or $0°/10°$ when performing spectral or bispectral measurements [AM01, GT94, HDC07]. In order to vary between these angular settings Holopainen et al. [HMI08] proposed a carefully calibrated bispectral goniometer setup, but the limited angular range and resolution prevents sampling a full BRRDF.

More within the field of reflectance capture, [PB96] constructed a spectral BRDF measuring gantry featuring a tunable monochromatized light source and broadband receiver, which makes it suitable for spectral, but not for bispectral measurements.

For computer graphics purposes the phenomena of reradiation and reflection can be treated in an unified manner, albeit being physically different. We will henceforth refer to both phenomena as “bispectral reflectance” while keeping our terminology as compatible as possible with the metrology and physics literature.

4.1. Direct Simulation

Figure 4.1.: Photograph of the fluorescent concentrator samples which we used for the measurements.

To correctly handle fluorescence in the computer graphics context, a simulation model has to be chosen, input data needs to be acquired for the spectral charac-

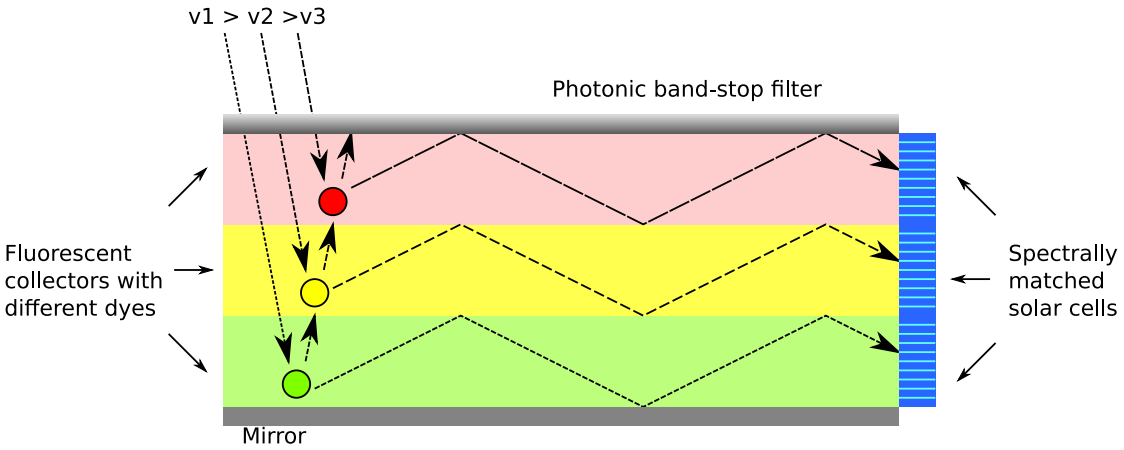


Figure 4.2.: Schematic of a fluorescent concentrator. The absorbed light is re-emitted at a different wavelength in the dye and then guided to the solar cell, with a very low probability of re-absorption. One can improve this by using a concentrator stack, as shown in the figure.

teristics of the material, and the simulation needs to be verified against measurements. We performed these steps in the context of solar cell research [BHD+08], in particular for fluorescent concentrators (see Figures 4.1 and 4.2).

Fluorescent concentrators have been developed to enhance the production of energy via solar cells and to reduce costs for this. They can concentrate both direct and diffuse light, which increases the efficiency of solar cells especially in cloudy weather.

A fluorescent concentrator is made from PMMA (acrylic glass) and contains a dye. Figure 4.2 illustrates the basic setup for one application of a fluorescent concentrator: If light enters the concentrator and hits a dye molecule it will be absorbed and re-emitted at a different wavelength according to the photoluminescence spectrum (PL-spectrum).

Since the PL-spectrum is shifted to longer wavelengths as compared to the absorption spectrum (see Figure 4.4), light is unlikely to be re-absorbed and therefore travels through the medium mostly undisturbed. By designing the fluorescent concentrator in a certain shape, light is trapped inside and can be guided to the solar cell due to total internal reflection.

Our work is based on the dissertation of Zastrow [Zas81] and the publications of Peters, Goldschmidt et al. [GGGW06, PGL+07] about fluorescent concentrators. It focuses on the physically correct simulation of fluorescent concentrators in order to gain deeper knowledge about the physical processes involved and to optimize the concentrator.

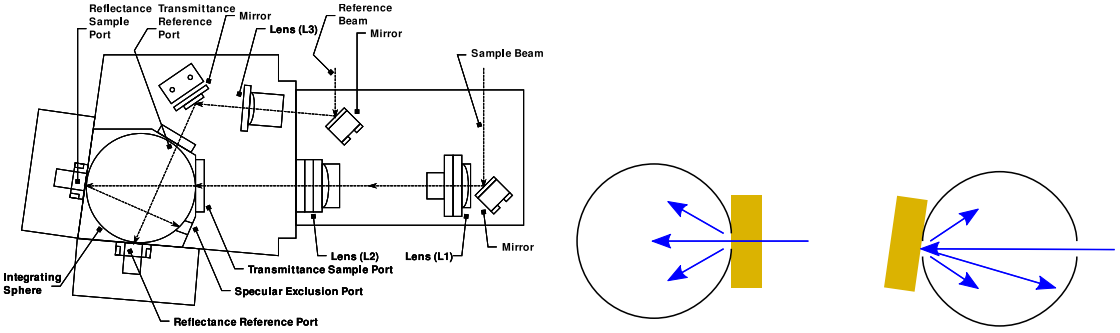


Figure 4.3.: Left: the full apparatus with the integrating sphere. Right: schematic of the measurement of transmission (left) and reflection (right) in the Ullbricht sphere.

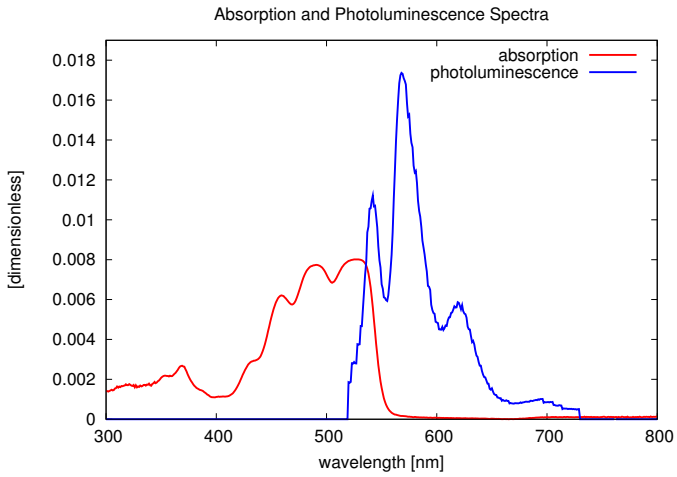


Figure 4.4.: The absorption and mean photoluminescence spectra.

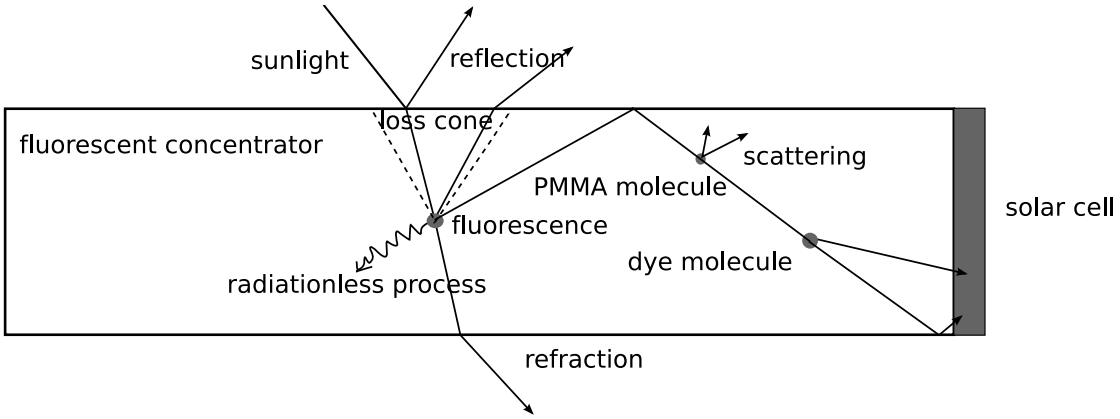


Figure 4.5.: Processes in a fluorescent concentrator.

4.1.1. Model

Parts of the processes in a fluorescent concentrator are well known and can be described by analytical models. But for example the behavior of the dye is still not fully understood and difficult to measure in an experimental setup. The required input parameters are reconstructed from the measured absorption and mean photoluminescence spectrum. The photoluminescence spectra are hard to measure because of multiple scattering which takes place in the dye. Therefore we restrict ourselves to a mean PL-spectrum which is used for simulation, independent of incoming wavelength λ_i. We used a range from 300 to 800 nanometers of the AM 1.5 spectrum [Ame] for the input wavelengths. The dye in the concentrator we used to test our model was BA241.

4.1.2. Verification by Experiments

To evaluate and improve the correctness of our simulation model we reproduced a couple of measurements that were made with real fluorescent concentrators. Amongst others we determined the absorption and the reflection of the concentrator through simulation. The measurements were made using a photospectrometer and an integrating sphere (see Figure 4.3). The challenge of the experimental measurement is the adjustment of the apparatus and the determination of the desired quantity in spite of measurement errors. For a comparison with the measured data it is necessary to fit the simulation parameters as best as possible to the experiment.

Transmission Experiment. This experiment captures all light which passes through the sample without absorption event and without being reflected at the top. The sample is attached in the apparatus as shown in Figure 4.3. This way, the transmittance of the material can be measured. The Ullbricht sphere is used to integrate all light entering the sphere, which can then be measured with a single sensor. The simulation results match the experimental data pretty closely, as can be seen in Figure 4.6 (top). The curve of the simulation data is a bit lower as the experimental curve for wavelengths above 500 nm and a bit higher for lower wavelength. This might be an indicator for inaccuracies in the calculation of the absorption that was used as an input for our simulation.

Reflection Experiment. The name of this experiment stems from the fact that most light detected here is due to reflection on the boundary (see Figure 4.3, right). The incoming light beam hits the sample with an angle of 8 degrees. The sensor will detect all light reflected or leaving the sample at the top. There might also be some rays that are scattered in the concentrator material (or absorbed in the dye and re-emitted) and led back to the sensor without being reflected anywhere. The results of this experiment are shown in Figure 4.6 (bottom). As in

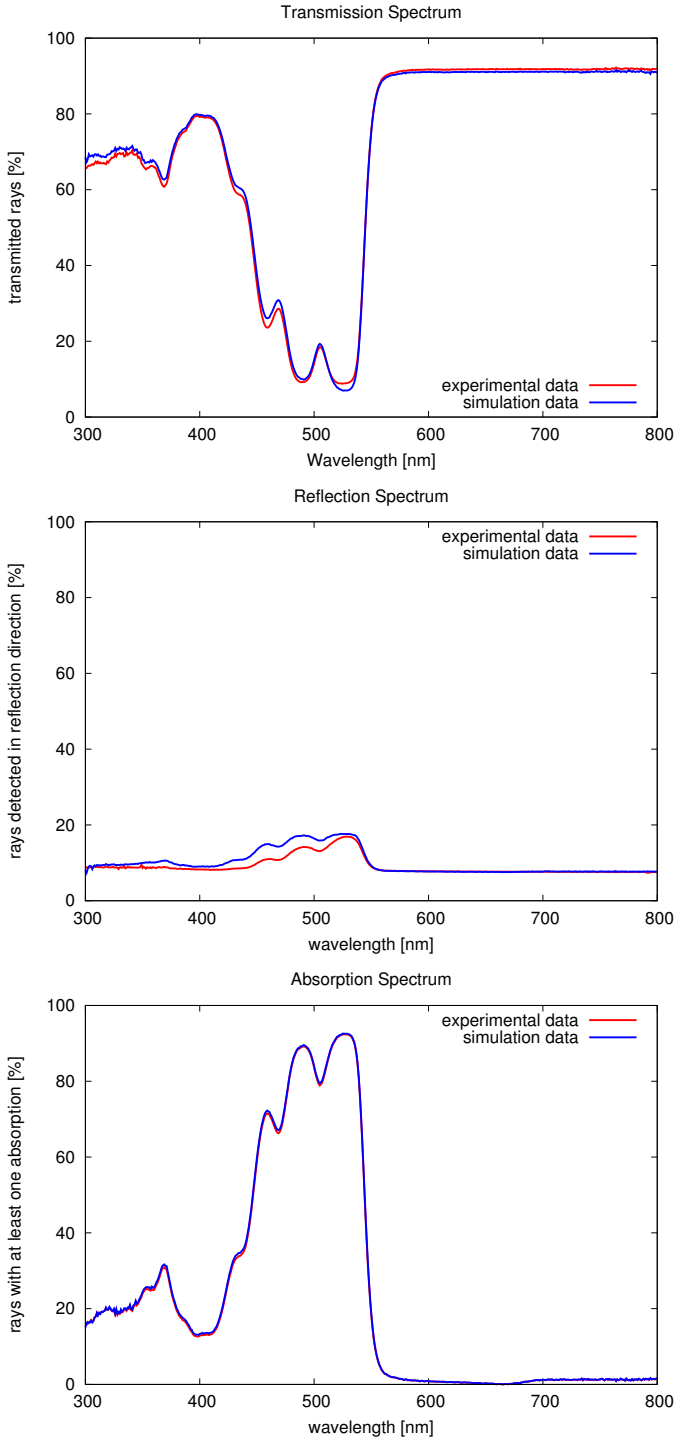


Figure 4.6.: Results for the transmission experiment (top), the reflection experiment (middle), and the absorption experiment (bottom). The top two graphs show data from our simulation in comparison to the measured data, the absorption graph compares the simulation to the data that was calculated using the data from the transmission and the reflection experiments.

the transmission experiment the simulation curve is a bit higher for wavelengths lower than 500 nm.

Absorption Experiment. Using the data from the transmission and the reflection experiment the fraction of light experiencing at least one absorption event on its way through the concentrator can be calculated as Absorption $\approx 1 -$ Reflection $-$ Transmission. In our simulation we can directly estimate the rays that had an absorption event. Figure 4.6 (bottom) shows the comparison of the calculated absorption and the absorption in our simulation, which fits perfectly. This proves the correctness of our absorption simulation.

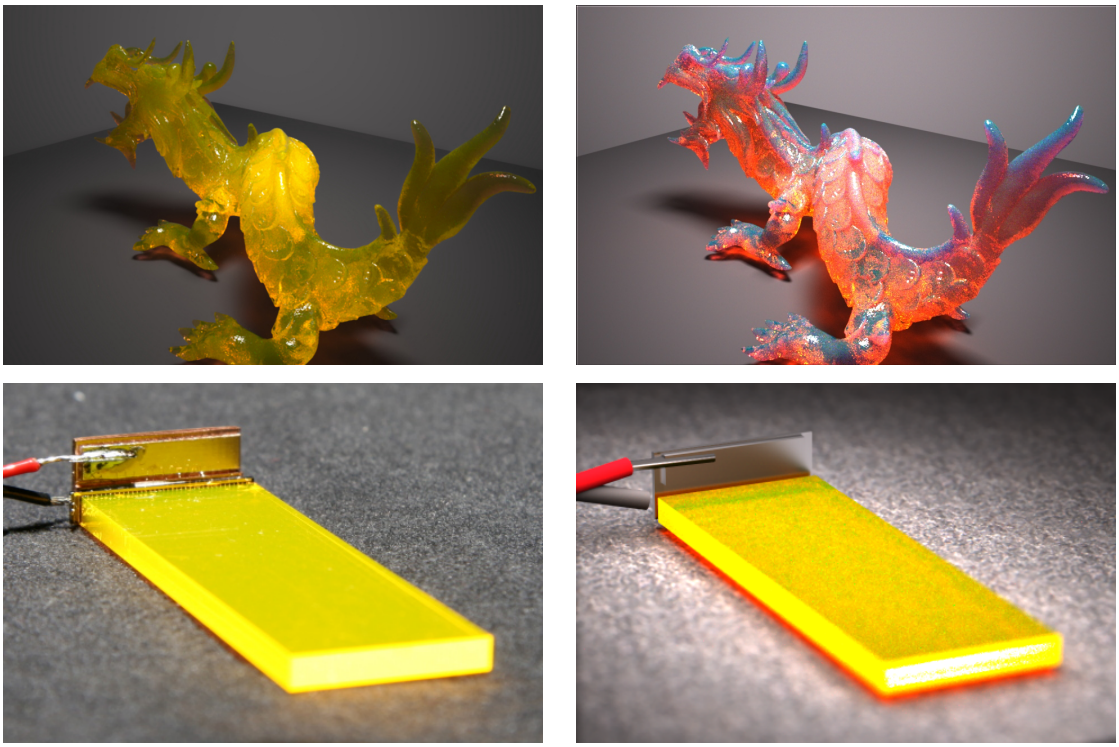


Figure 4.7.: Top: a dragon with fluorescence in the medium (left) and the same dragon with absorption only. Bottom: a photograph of a fluorescent concentrator (left) and a rendering (right).

4.1.3. Rendering

The spectral data obtained this way was included in the spectral rendering framework described in Chapter 2. Light tracing (i.e. starting at the light sources) is done exactly as Figure 4.5 suggests. In the event of fluorescence, a new wavelength λ_o is importance sampled according to the mean PL-spectrum, using the

inversion method [Sob94]. Adjoint transport (i.e. starting at the camera is slightly less efficient, as λ_o is already fixed when a fluorescent particle is reached. That is, the PL-spectrum needs to be evaluated and applied to the path weight before a new incoming wavelength λ_i can be sampled. This essentially means we have to rely on Metropolis sampling to importance sample the right wavelengths when starting new paths at the camera.

Resulting images can be seen in Figure 4.7. The dragon on the top left is rendered with the data acquired from the sample in the photograph just under it. The image on the top right is the same dragon, but rendered with absorption only, to visualize the Stokes shift. The simulated light guiding effect can be clearly observed in the bottom right rendering.

4.2. Fluorescent Surface Radiance Transfer

In this section, we integrate the concepts of fluorescence and bi-directional reflectance distribution functions into the *bispectral BRRDF* that can describe general fluorescent (and non-fluorescent) materials and the bidirectional dependency of their wavelength-preserving reflectance and their wavelength-shifting reradiation.

In optics, the *bispectral luminescent radiance factor* is commonly used to describe fluorescent materials. This is inconvenient for our purposes, as it defines fluorescence relative to a perfect, non-fluorescent diffuser.

Note that reflection and reradiation are different physical mechanisms and are treated as such throughout the scientific literature. In a computer graphics context, however, it makes sense to abstract reradiation as an instantaneous phenomenon and treat it in the same way as reflectance.

Fluorescence is a phenomenon taking place in particles of a participating medium. For practical purposes, it is desirable to incorporate fluorescence into the standard surface reflection notion of a bidirectional reflectance distribution function (BRDF). In particular, this approach includes directional effects without wavelength shift, such as Fresnel reflection.

That is, we extend the well-known concept of the BRDF to account for energy transfer between wavelengths, resulting in a *Bispectral Bidirectional Reflectance and Reradiation Distribution Function (bispectral BRRDF)*. Unfortunately, no bidirectional reradiation measurements of fluorescent materials have been available so far. Using a bidirectional and bispectral measurement setup, we acquired reflectance and reradiation data of a variety of fluorescent materials, including vehicle paints, paper and fabric, and compare their renderings with RGB, RGB$\times$RGB, and spectral BRDFs [HHA+10]. Our acquisition is guided by a principal component analysis on complete bispectral data taken under a sparse set of angles. We show that in order to faithfully reproduce the full bispectral information for all other angles, only a very small number of wavelength pairs needs to be measured at a high angular resolution.

Radiance	$L(\omega)$	$\left[\frac{\mathrm{W}}{\mathrm{sr}\cdot\mathrm{m}^2}\right]$
Spectral Radiance	$L(\omega,\lambda)$	$\left[\frac{\mathrm{W}}{\mathrm{sr}\cdot\mathrm{m}^2\cdot\mathrm{nm}}\right]$
Spectral Irradiance	$E(\lambda)=\int_\Omega L(\omega,\lambda)\mathrm{d}\omega$	$\left[\frac{W}{m^2\cdot nm}\right]$
Irradiance	$E=\int_\Lambda\int_\Omega L(\omega,\lambda)\mathrm{d}\omega\ \mathrm{d}\lambda$	$\left[\frac{W}{m^2}\right]$

Table 4.1.: Definitions of spectral quantities; ω refers to directions and λ to wavelengths.

4.2.1. Bispectral Rendering Equation

Light transport considering energy transfer from one wavelength to another, in order to account for fluorescence, can be expressed by the bispectral rendering equation:

$$L(\omega_o,\lambda_o) = L_e(\omega_o,\lambda_o) + \tag{4.1}$$

$$\int_\Omega\int_\Lambda L(\omega_i,\lambda_i) f_r(\omega_o,\omega_i,\lambda_o,\lambda_i)\cos\theta d\lambda_i\, d\omega_i, \tag{4.2}$$

which in contrast to the standard rendering equation (2.12) requires an additional integration over all incident wavelengths λ_i. In order to stay consistent with Glassner [Gla94], we denote the wavelengths by λ_o and λ_i (as opposed to λ and μ that are often used in optics and other engineering fields).

Bispectral BRRDF. The bispectral rendering equation includes the *bispectral BRRDF* $f_\mathrm{r}(\omega_\mathrm{o},\omega_\mathrm{i},\lambda_\mathrm{o},\lambda_\mathrm{i})$ that describes the angularly dependent reflectance for any pair of wavelengths. Its definition is different from the well-known definition of a spectral BRDF (Equation (2.11)), which cannot represent fluorescent materials. Furthermore, it is more general than the use of directionally independent reradiation matrices as proposed by Donaldson [Don54].

Before we provide the general bispectral BRRDF, let us briefly recall the definition of a spectral BRDF. The required spectral quantities are repeated from Section 2.2 in Table 4.1. Note that the spectral quantities feature a different unit compared to their non-spectral counterparts. Following Nicodemus et al. [NRH+77], the differential reflected spectral radiance $\mathrm{d}L_\mathrm{o}(\omega_\mathrm{o},\lambda_\mathrm{o})$ due to the incident differential spectral irradiance $\mathrm{d}E(\lambda)$ from direction ω_i is given as:

$$\mathrm{d}L_\mathrm{o}(\omega_\mathrm{o},\lambda) = \mathrm{d}E(\lambda) f_\mathrm{r}(\omega_\mathrm{o},\omega_\mathrm{i},\lambda) \quad \left[\frac{W}{sr\cdot m^2\cdot nm}\right], \tag{4.3}$$

with ω_i and ω_o being the incident and outgoing directions. The spectral BRDF $f_\mathrm{r}(\omega_\mathrm{o}, \omega_\mathrm{i}, \lambda)$ for a single wavelength is therefore defined as the ratio of differential reflected spectral radiance to differential incident spectral irradiance:

$$f_\mathrm{r}(\omega_\mathrm{o}, \omega_\mathrm{i}, \lambda) = \frac{\mathrm{d}L_\mathrm{o}(\omega_\mathrm{o}, \lambda)}{\mathrm{d}E(\lambda)} = \frac{\mathrm{d}L_\mathrm{o}(\omega_\mathrm{o}, \lambda)}{L_\mathrm{i}(\omega_\mathrm{i}, \lambda)\cos(\theta_\mathrm{i})\mathrm{d}\omega_\mathrm{i}} \quad \left[\frac{1}{sr}\right]. \tag{4.4}$$

It follows that the unit of the spectral BRDF is $\left[\frac{1}{sr}\right]$, which is the same as for non-wavelength dependent BRDFs $f_\mathrm{r}(\omega_\mathrm{o}, \omega_\mathrm{i})$, although the units for L and E differ in the spectral vs. non-spectral case.

We now generalize Nicodemus' derivation of the BRDF to account for cross-wavelength energy transfer by the bispectral BRRDF and show that its unit differs from the spectral BRDF. Referring to the bispectral rendering equation (Eq. 4.1) the differential reflected and reradiated spectral radiance (differential with regard to the incident direction ω_i *and* the incident wavelength λ_i) is due to incident double differential *(non-spectral)* irradiance for ω_i and λ_i:

$$\mathrm{d}^2L_\mathrm{o}(\omega_\mathrm{o}, \lambda_\mathrm{o}) = \mathrm{d}^2E \cdot f_\mathrm{r}(\omega_\mathrm{o}, \omega_\mathrm{i}, \lambda_\mathrm{o}, \lambda_\mathrm{i}) \quad \left[\frac{W}{sr \cdot m^2 \cdot nm}\right], \tag{4.5}$$

and hence the *bispectral BRRDF* may be defined as

$$f_\mathrm{r}(\omega_\mathrm{o}, \omega_\mathrm{i}, \lambda_\mathrm{o}, \lambda_\mathrm{i}) = \frac{\mathrm{d}^2L_\mathrm{o}(\omega_\mathrm{o}, \lambda_\mathrm{o})}{L_\mathrm{i}(\omega_\mathrm{i}, \lambda_\mathrm{i})\cos(\theta_\mathrm{i})\mathrm{d}\omega_\mathrm{i}\,\mathrm{d}\lambda_\mathrm{i}} \quad \left[\frac{1}{sr \cdot nm}\right]. \tag{4.6}$$

The bispectral BRRDF is a general way to represent fluorescent materials as it does not make any assumptions about the material.

In the discretized case, an individual sample of the bispectral BRRDF for the directions $(\omega_\mathrm{i}, \omega_\mathrm{o})$ expresses the energy transfer from the incoming spectrum to the reflected spectrum as a matrix over λ_o and λ_i, see Figure 4.17. While the diagonal entries refer to reflection at the same wavelength, the fluorescent effect is represented by the off-diagonal part. As there is typically no transfer from longer to shorter wavelengths (towards higher energy), the upper triangle will remain black.

4.2.2. Measurement Setup

In order to acquire isotropic bispectral BRRDFs, we have built a fully automated, image-based measurement device. It closely follows the design of Matusik et al. [MPBM03a] for isotropic BRDFs but with the added capability to emit and acquire at specific wavelength bands (Figure 4.8).

Measurement Device. The spectral filters used are LCD-based Lyot filters (CRi VariSpec VIS10/35 mm) whose transmission bands are about 10 nm–20 nm wide and range from 400 nm to 720 nm. We apply additional polarization scrambling

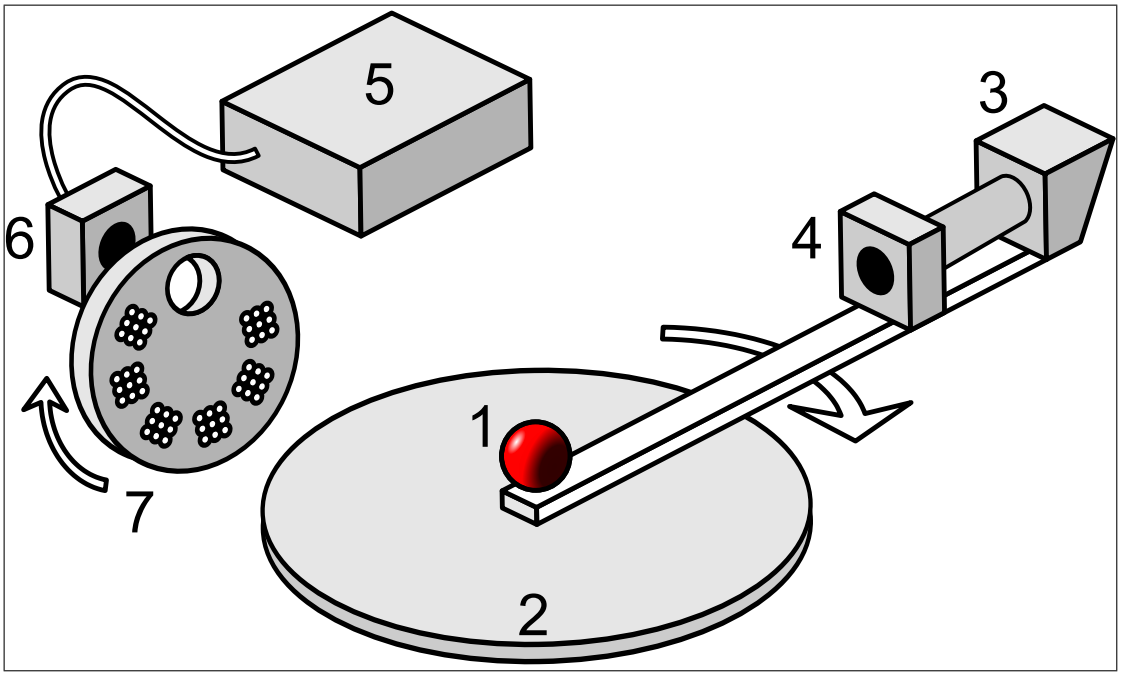


Figure 4.8.: A depiction of our setup. A sample sphere (1) is mounted on a turntable (2), to which a digital monochrome still camera (3) is attached. The camera is equipped with a visible-spectrum tunable filter (4). The sphere is illuminated by a light guide coupled xenon light source (5) with another tunable filter (6) mounted in front; near-UV light is generated with LEDs that can be selected using a motorized wheel (7). On the exit aperture of (6) and the entry aperture of (4), we attach optical depolarizers.

optics to undo the linear polarization from the LCD filters so as not to bake any unwanted side effects into the measured BRRDF. Figure 4.9 illustrates the strong influence of polarization both on the specular and the diffuse reflection for a sample with clear coat. Our experiments show that even non-coated, apparently diffuse surfaces do not necessarily completely decorrelate the polarization state.

As light source we employ a xenon arc lamp coupled into a light fiber (XION medical Xenon R180), whose light has a flat and stable spectrum (measured using a spectroradiometer) but rather weak blue and UV output, especially after passing the spectral filter. We therefore add LEDs for better coverage of this range (370 nm–420 nm in 10 nm steps). The camera is a monochrome, digital still camera (Jenoptik ProgRes MFcool), with which we acquire high-dynamic-range images using exposure series from 1 ms to 16 s.

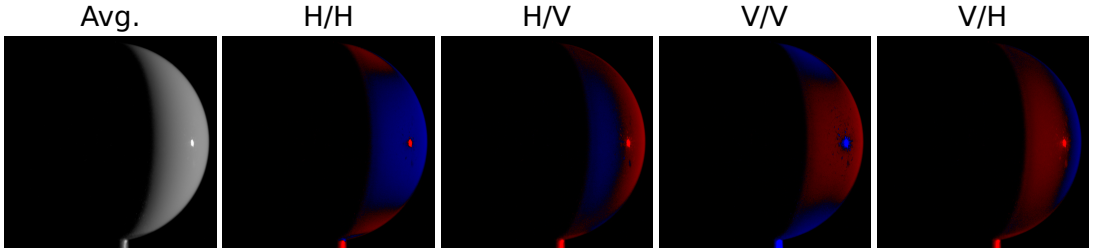


Figure 4.9.: Influence of polarization on reflection from a sphere: Average image and difference images (red: negative) depending on the polarization state of light source and observer, where “H/V” stands for “horizontal in, vertical out”. Near the Brewster angle, the specular highlight is contained almost exclusively in the V/V component. Also, note the variation in the diffuse regions.

Data Acquisition. The straightforward way of acquiring a bispectral BRRDF is to capture images at all turntable rotations β for every pair of wavelengths $(\lambda_\mathrm{o}, \lambda_\mathrm{i})$. For practical reasons we constrain ourselves to 20 nm steps in the range from 380 nm to 720 nm for λ_i and 400 nm to 720 nm for λ_o, amounting to 170 images per β as the upper triangle of the bispectral matrix can be ignored. We vary β in the range of 5° to 170°. For highly specular materials a stepping of $5°$ is chosen to sufficiently sample the sharp highlight while we take a coarser sampling of 10° for materials of lesser angular bandwidth.

Sample Geometries. Depending on the material, we use two different sample geometries: coated spheres for the paints and a custom-made piecewise cylindrical object (Figure 4.19) wrapped in stripes of paper or fabric.

Due to the varying normals of the shapes each surface point will be illuminated and viewed from a slightly different direction. From simple geometric considerations we can determine $(\omega_\mathrm{o}, \omega_\mathrm{i})$ for every pixel captured under a specific turntable rotation β.

4.2.3. PCA-based Acquisition

For storage and further processing, we discretize the data for each wavelength pair in 32^3 bins using the $(\theta_\mathrm{o}, \theta_\mathrm{i}, \phi_\mathrm{diff})$ parameterization chosen in [MPBM03a]. For the strongly specular materials, 64^3 bins are used. Bins that are not populated due to the coarse sampling of the turntable position are filled in by diffusion.

We perform an adaptive, PCA-steered measurement, i.e. we first acquire full bispectral data sets for a small number of turntable angles. After performing the PCA decomposition, we acquire dense angular data only for the sparse bispectral

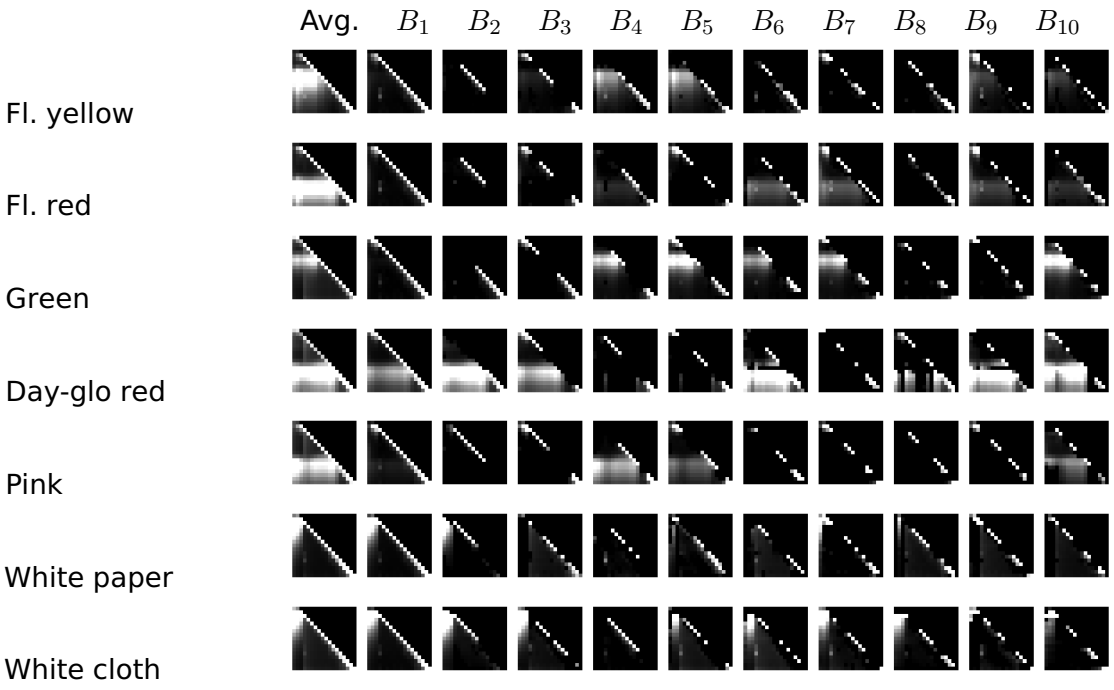


Figure 4.10.: Average and the first 10 principal components B_i.

basis that is required for a good reconstruction. Only a small set of wavelength pairs needs to be measured.

We opt for a data-driven representation of the bispectral BRRDF because we do not want to make strong *a priori* assumptions about the spectral and angular behavior of the material. Particularly, we know that due to the different nature of fluorescent reradiation and specular reflection, the overall spectral and angular variation of most bispectral BRRDFs (say, a diffuse red material with a white highlight) is not strictly separable in the form

$$f_\mathrm{r}(\omega_\mathrm{o}, \omega_\mathrm{i}, \lambda_\mathrm{o}, \lambda_\mathrm{i}) = f^\lambda(\lambda_\mathrm{o}, \lambda_\mathrm{i}) f^\omega(\omega_\mathrm{o}, \omega_\mathrm{i}). \tag{4.7}$$

This is unfortunate, since it would have allowed us to measure angular and spectral dependencies f^ω and f^λ separately and simply compute the bispectral BRRDF as the outer product of both functions. On the other hand, it is always possible to expand f_r into a series of separable terms:

$$f_\mathrm{r}(\omega_\mathrm{o}, \omega_\mathrm{i}, \lambda_\mathrm{o}, \lambda_\mathrm{i}) = \sum_n f_n^\lambda(\lambda_\mathrm{o}, \lambda_\mathrm{i}) f_n^\omega(\omega_\mathrm{o}, \omega_\mathrm{i}). \tag{4.8}$$

Since fluorescence, due to its physical nature, is only weakly directional, a PCA will yield such a decomposition of low rank. We exploit this insight and perform a dense bispectral measurement under only a sparse set of turntable angles (0°, 70° and 150°, each of which corresponds to a 2D slice of the BRDF). The

measurements from this sparse set of turntable positions contain samples from a lot of different angles of incidence and exitance. The bispectral correlations found in these measurements can then be transferred to other angles. Figure 4.15 illustrates this for measurements taken at two turntable positions.

We assemble a matrix F which contains all bispectrally-valued BRRDF samples with the average value $\bar{f}$ subtracted, and compute its SVD. Selecting the n eigenvectors with the greatest eigenvalues the basis B is assembled. Examples of such eigenvectors can be seen in Figure 4.10.

Measurement Basis. Since our narrowband filter assembly only allows for the sampling of wavelength pairs $(\lambda_\mathrm{o}, \lambda_\mathrm{i})_k$, we cannot measure in the PCA basis directly. The estimation of a bispectral sample f from the sparse measurement $f' = \{f_k(\lambda_\mathrm{o}, \lambda_\mathrm{i})\}$ calls for a basis transformation P^+, which we obtain as the pseudo-inverse of the matrix P which in turn is composed of the subset of rows of B corresponding to $(\lambda_\mathrm{o}, \lambda_\mathrm{i})_k$:

$$f \approx BP^+(f' - \bar{f}) + \bar{f}. \tag{4.9}$$

The selection of suitable wavelength pairs $(\lambda_\mathrm{o}, \lambda_\mathrm{i})_k$ influences the stability of the approximation which is correlated to the condition number of P. Starting with the wavelength pair corresponding to the greatest entry in B, we follow a greedy strategy selecting the set which brings $\mathrm{cond}(P)$ closest to 1. Our approach is inspired by Matusik et al. [MPBM03b] who reduced the number of measured directional samples for ordinary BRDFs using a similar analysis.

4.2.4. Rendering

For the renderings, we used the framework as described in Chapter 2, i.e. we split path space in two unidirectional parts. This is quite necessary in this setting, as paths will change their wavelength on the way, making deterministic connections between paths almost impossible. These would be simpler if a full spectrum (not only a continuous random variable in that domain) would be transported. But given the problems with this approach mentioned in Chapter 2 and the fact that deterministic connections often suffer from bad importance sampling, we believe partitioned path tracing is the right choice. One additional criterion comes into play with fluorescence: the materials are not reciprocal by nature (the wavelength shift is the other direction). So connecting paths from different directions at different wavelengths deterministically by simply evaluating the bispectral BRRDF is possible, but will also make the problem of bad importance sampling worse by one dimension.

We use Metropolis sampling to guide sampling by global importance, also including the spectral domain. This is more general than partial importance sampling by fitted, analytical BRDF models for each (λ_i, λ_o) pair, which would also not represent the shape of a measured lobe very well. Data-driven importance

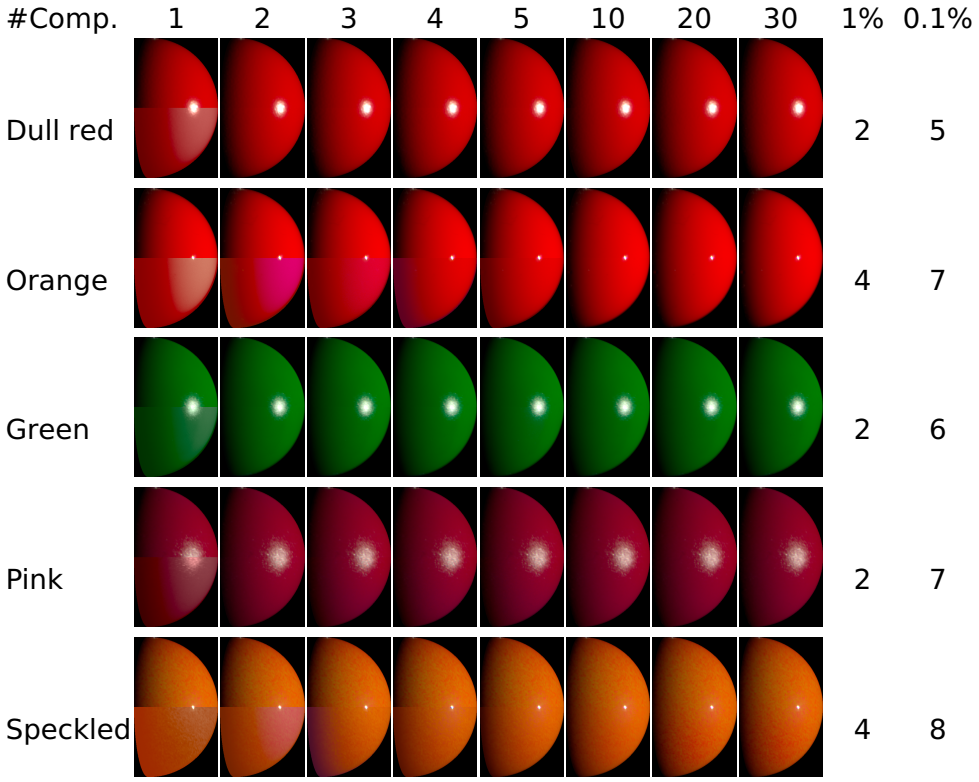


Figure 4.11.: Image-based same-angle reconstruction of full bispectral data (20 nm resolution) from a small number of acquired bispectral samples. As a reference, we provide ground truth in the top half of each image, and the number of basis vectors required for the residual energy to drop below 1% and 0.1%, respectively. Note that in the case of strong specular highlights (Orange, Speckled), the numerical error does not reflect the visual difference well.

sampling by bins (i.e. multi-dimensional wavelet importance) with correction factors for linear interpolation would be an improvement.

When evaluating the bispectral BRRDF $f_r(\theta_\mathrm{o}, \theta_\mathrm{i}, \phi_\mathrm{diff}, \lambda_\mathrm{o}, \lambda_\mathrm{i})$, a specific reflectance sample is obtained by multilinear interpolation from our bispectral BRRDF representation. Memory requirements can be stripped down by storing the BRDF as very few coefficients along with the PCA bases created during acquisition.

As polarization has already been scrambled during measurement, it is ignored while rendering.

In Figure 4.21, we demonstrate that fluorescence does in fact require bispectral modeling of sufficient resolution. After reducing the measured bispectral BRRDF samples to a 3×3 (RGB×RGB) matrix by integrating over the RGB spectral curves,

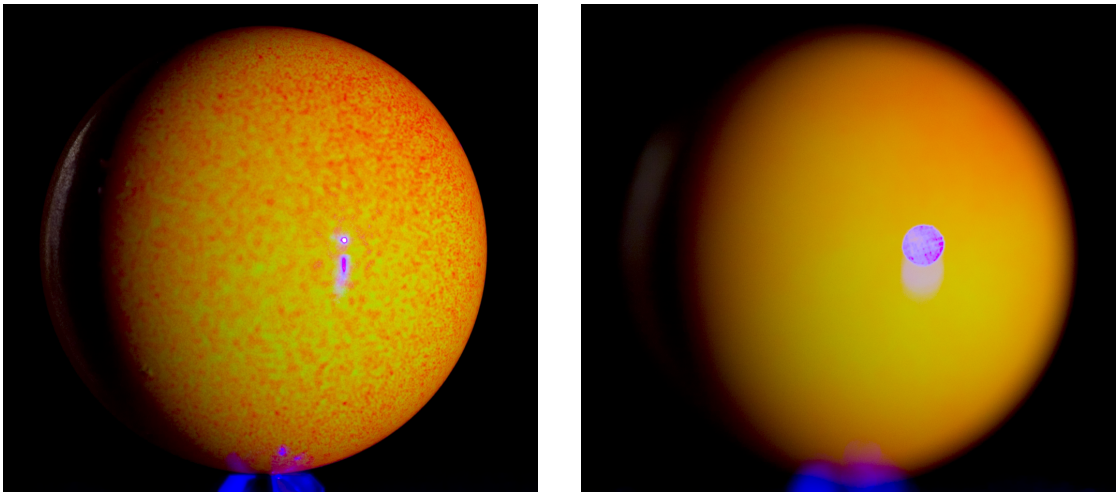


Figure 4.12.: The *Speckled* sample, while not a homogeneous BRRDF, represents the class of mixed materials with angularly dependent visibility of the individual components. Note the red shift towards grazing angles in the defocused shot (right). Photos taken under 420nm light.

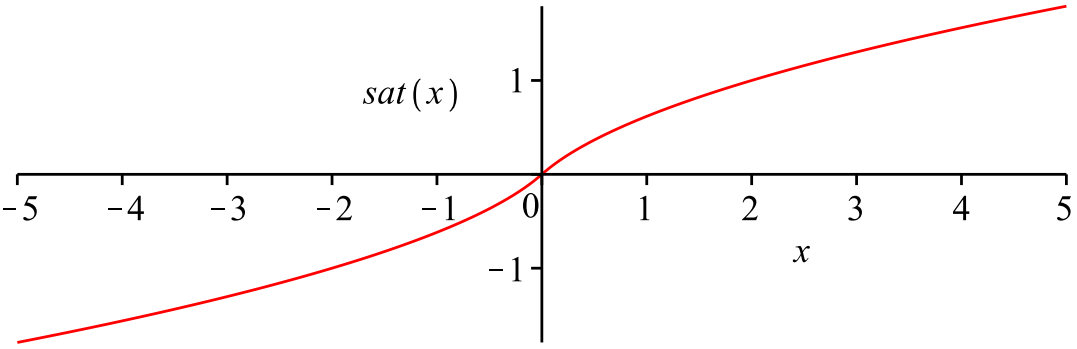


Figure 4.13.: As saturation function $\tilde{x} = sat(x)$, we chose the inverse of $x = sign(\tilde{x})(\tilde{x}^2 + |\tilde{x}|)$, which is linear for small values of $|x|$, but of $\mathcal{O}(\sqrt{x})$ asymptotically.

the renderings show clear differences to the full bispectral BRRDFs. Especially for the Yellow sample, the coarse RGB×RGB representation is unable to reproduce reradiation, which is sharply centered around 540 nm. We also integrate the bispectral BRRDF into a spectral BRDF by assuming a uniform illuminant spectrum. Again, the differences can clearly be seen. While they are less pronounced because the illumination in this scene is similar to the spectrum used for the conversion, slight deviations in color and intensity can still be made out. Reducing

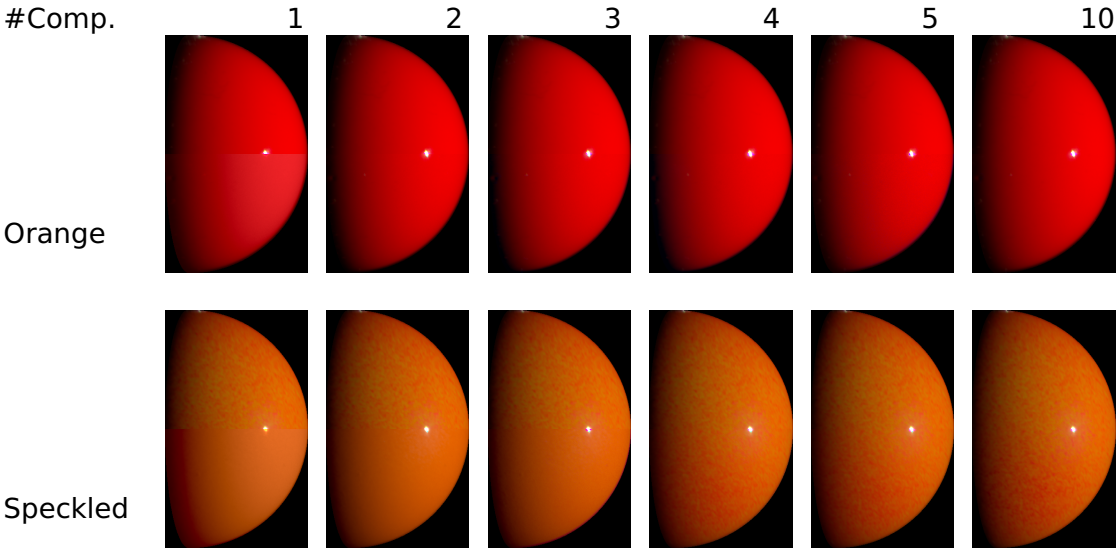


Figure 4.14.: By applying a saturation function on the covariance matrix, the convergence in the non-specular parts is significantly improved.

the bispectral BRRDF to a simple RGB BRDF (again assuming a uniform incident spectrum) shows obvious differences. These differences are most pronounced for non-white spectra as demonstrated in Figure 4.20.

4.3. Results

Using our measurement device and acquisition scheme, we have captured bispectral BRRDFs of a number of fluorescent materials, including fluorescent paints with or without clear coating as well as paper and white cloth (shown in Figure 4.16).

The strength of the reradiation therefore depends heavily on the illuminating spectrum as demonstrated in Figure 4.18 and Figure 4.19. In Figure 4.19, RGB photographs are compared against renderings under a 5600K illuminant. As our illumination system contains near-UV LEDs, we can even capture materials such as the paper sample which exhibit significant reradiation in the blue to UV range. The effect is clearly visible in Figure 4.18. Our captured bispectral BRRDFs faithfully reproduce the fluorescence in both images.

Same-angle Reconstruction Using Standard PCA. In Figure 4.11, we show a reconstruction of a single-angle bispectral measurement from the PCA basis for the same angle. The error measure provided is given by the residual energy as determined by the SVD and relates to the full bispectral dataset with 170 wavelength pairs, not just the resulting sRGB color vectors as shown in the figure.

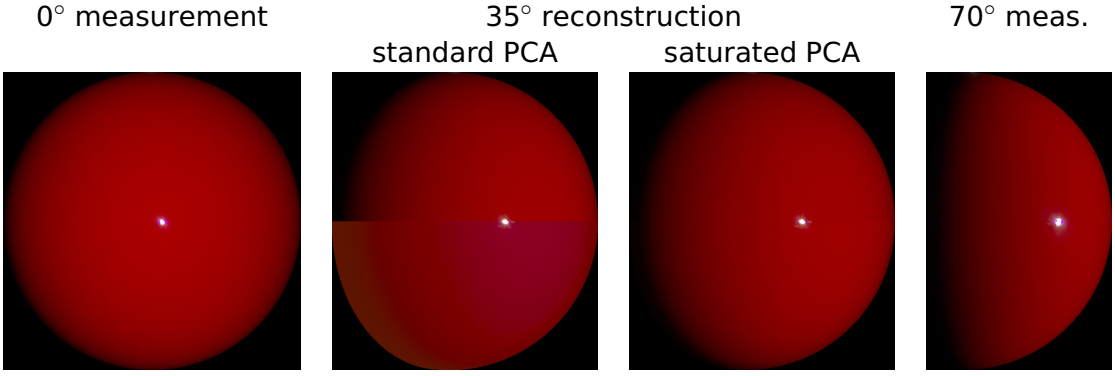


Figure 4.15.: Reconstruction of an intermediate angle, which was not sampled for the PCA, from 5 wavelength pairs using the standard and saturated PCA approaches. Again, the top half of the reconstructed images is ground truth. The PSNR has been computed separately for the highlight area and the rest.

Although the fidelity increases with the cardinality of the measurement basis, the visual appearance even at a numerical error as low as 1% or 0.1% is not always fully satisfactory. It is due to the least-squares nature of the SVD that materials with particularly strong specular highlights (Orange, Speckled) attract the attention of the first few eigenvectors at the expense of a slower convergence in the nonspecular regions. Remarkably, a decent reconstruction is often achieved even before the inclusion of the first measurement of an off-diagonal wavelength pair. In all five examples, the first 9 components were based purely on wavelength-preserving measurements.

Speckled Dataset. The “Speckled” material (Figure 4.12) is a fluorescent yellow sphere onto which we manually applied a non-uniform layer of red speckles. While this sample does not have a homogeneous BRRDF, we included it as it represents a material with complex microstructure resulting in directionally dependent fluorescence, as can be seen in the defocused image (Figure 4.12 right).

Taming the Specular Highlight. It is to be expected that a signal decomposition based on a L^1 measure would no longer overemphasize the importance of the specular highlight. However, the few L^1-PCA approaches in existence are computationally rather expensive and, due to their nonlinear optimization scheme, offer no guarantee of global convergence [BBdF96, KK03]. In order to emulate a similar behavior using a standard PCA, we apply a saturation function on the values in

the covariance matrix (and its inverse after the reconstruction), which dampens the high pixel values in the highlight region (Figure 4.13). As a consequence, the visual convergence is sped up considerably (Figure 4.14).

Angular Dependency. For most materials (except Speckled), the resulting principal components show that the fluorescent entries in the reradiation matrix $\lambda_i \neq \lambda_o$, taken for themselves, carry a rather weak angular dependency, i.e., indicating reasonable separability. However, as soon as the full matrix, the full bispectral BRRDF including the non-fluorescent elements, is considered, at least two, most often even more separable functions are required for faithful reproduction.

Angle Transfer. Earlier on, we assumed that the bispectral decomposition of an angular BRRDF slice can be transferred to the spectrally sparse measurements taken under different angles. In Figure 4.15, we provide experimental evidence for this assumption. Using a PCA basis that was computed using fully bispectral measurements at turntable angles of 0° and 70°, we reconstruct a fully bispectral intermediate slice at 35° out of only five measured wavelength pairs.

Our PCA-guided measurement routine allows us to drastically reduce the acquisition cost. As an example, let us assume a sampling of 5° steps from 0° to 150° for the turntable position. If all 170 bispectral entries are captured for each angle, the total time of optical exposure amounts to approx. 45 hours. By performing the full bispectral measurement under three turntable orientations only, and by measuring only 5 out of 170 wavelength pairs for the remaining angles, the acquisition time drops to 5.5 hours.

4.4. Conclusion

We conducted a physically based Monte Carlo simulation of fluorescent concentrators employing ray tracing. The fluorescent dye in the participating medium was measured in a complex apparatus using an integrating sphere and a photospectrometer which results in a fine resolution in the wavelength domain. Unfortunately multiple scattering in the concentrator probe prevents simple measurement of the photoluminescence spectrum of a dye molecule, so the mean PL-spectrum had to be used.

For simple rendering of fluorescent surfaces we have provided the definition of the bispectral BRRDF, which enables us to model directionally-dependent fluorescent behavior. The proposed image-based measurement setup can acquire such bispectral reflectance and reradiation functions efficiently.

Even though real-world BRRDFs are not directly separable into spectral and angular functions, we can apply a PCA-steered acquisition scheme that only measures relevant bispectral samples of the BRRDF, resulting in a significant speedup (approx. 9:1), rendering such acquisition practical.

One of the shortcomings of a standard PCA in this context is its optimization of an L^2 error function. As a consequence, the importance of specular highlights is often overemphasized, which leads to slower convergence in the non-specular regions. While a L^1-based decomposition could potentially resolve this issue, we reached a simpler solution that is just as effective. By reweighting the covariance matrix before performing the SVD, we are able to improve the reconstruction fidelity for very small numbers of measured components. This also extends to spatially varying materials. For most of our materials, we can reach a visually indistinguishable reconstruction using only 5 or even fewer measurements per angle (as compared to 3 for a standard RGB measurement). This should allow for efficient acquisition of even higher-dimensional functions (anisotropic bispectral BRRDFs, bispectral BTFs or reflectance fields) in the future.

Summing up, this chapter included the rendering of fluorescence into light transport simulation, both in a fluorescent medium and at the surface. The robustness of both simulations has been verified against measured data.

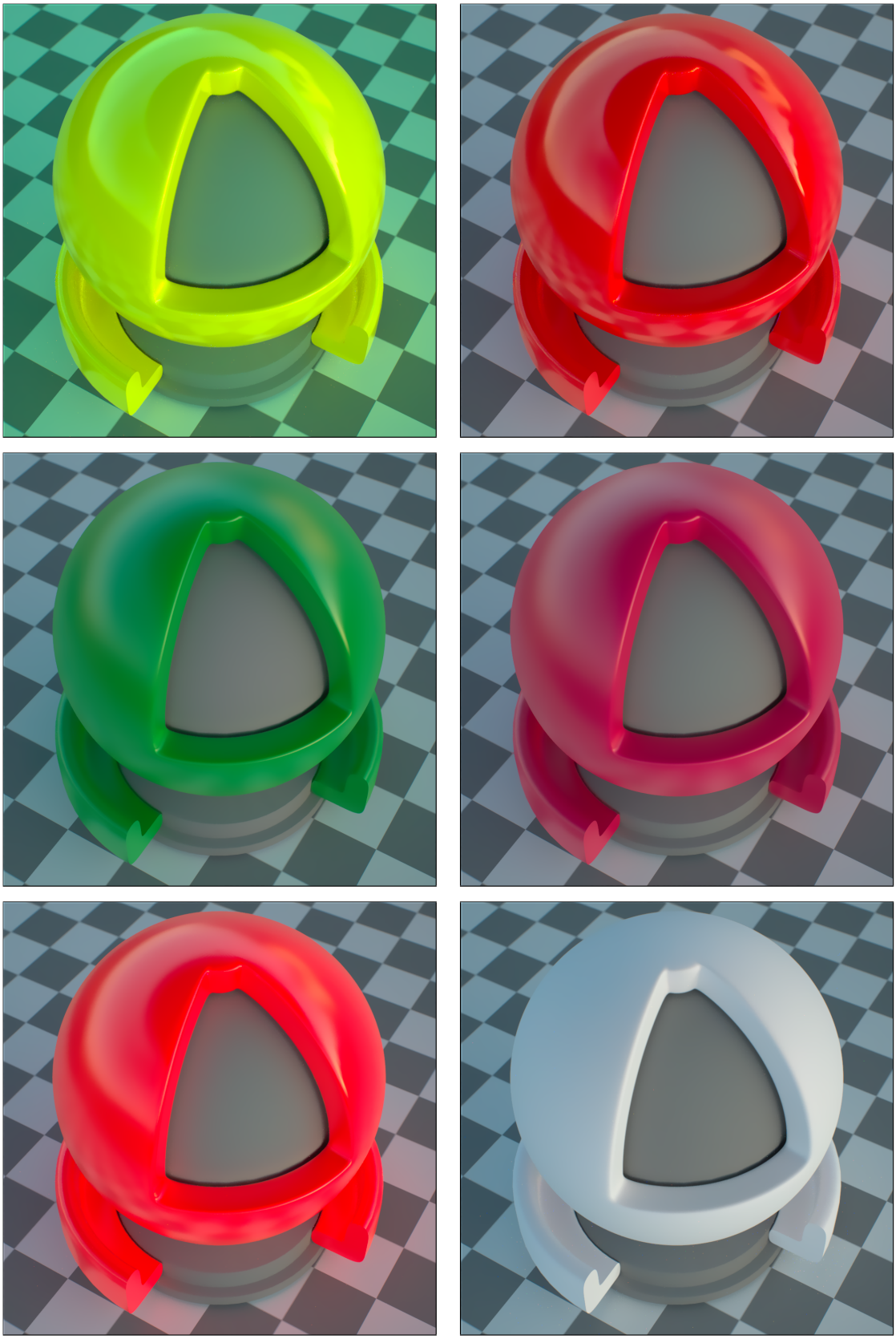


Figure 4.16.: Six measured BRRDFs, rendered using two-directional Metropolis guided path sampling (LT+PTDL), illuminated by a spectrally measured sky. From top left to bottom right: fluorescent yellow, red, green, pink, day-glo red, and white paper.

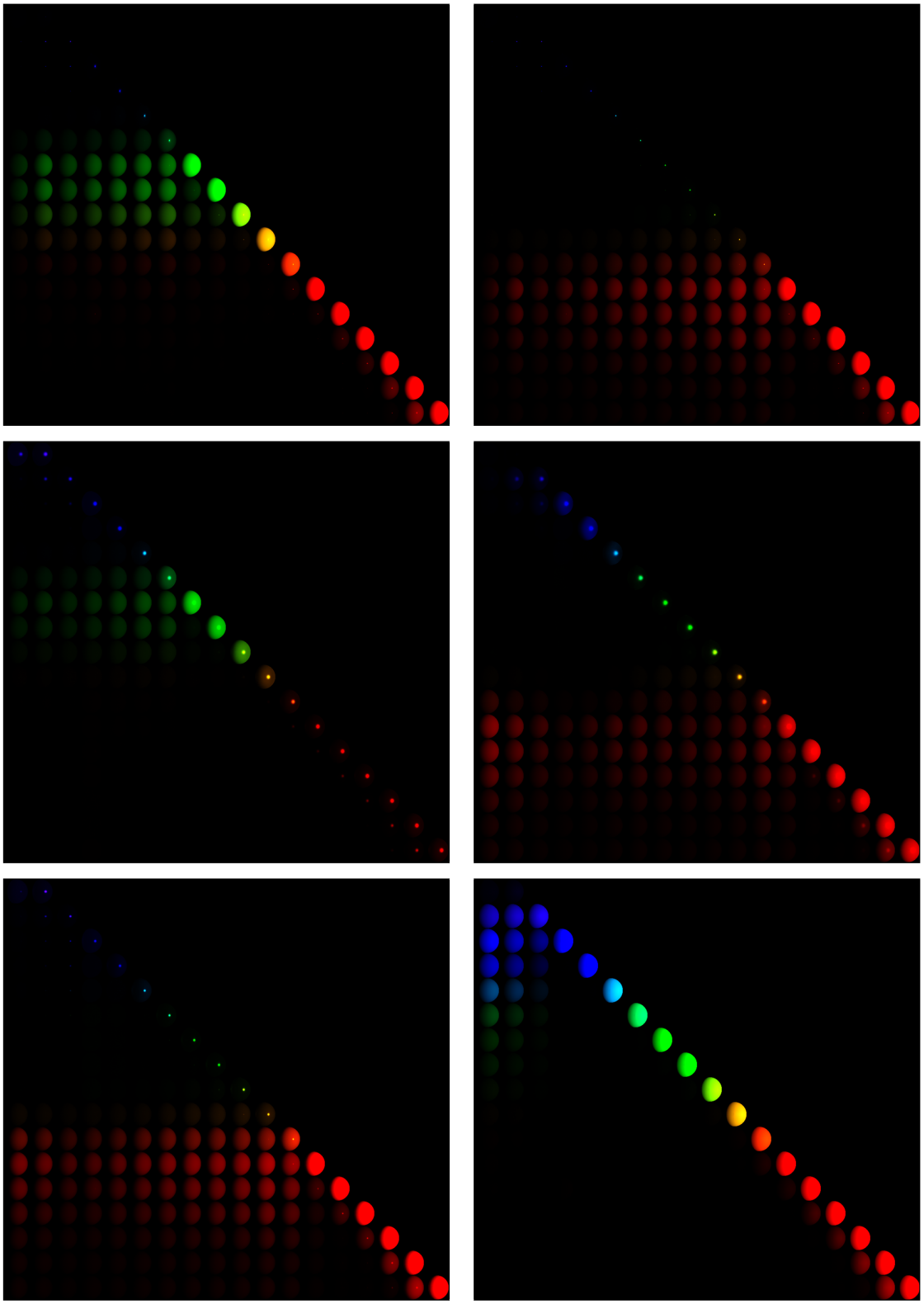
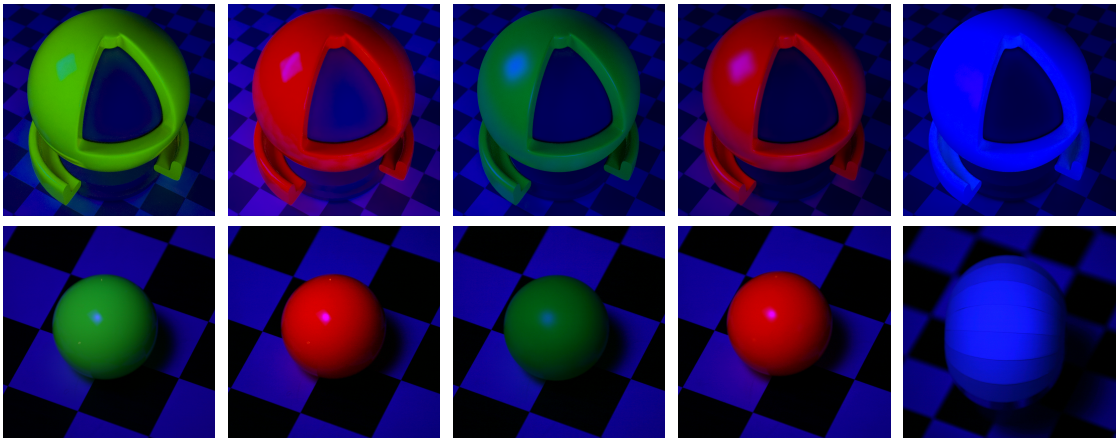
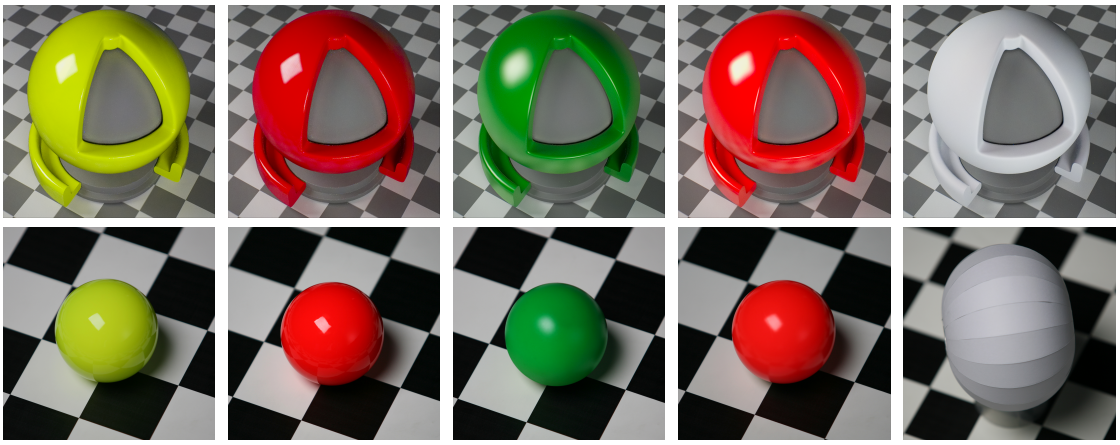


Figure 4.17.: The angular configurations viewed in reradiation matrices for the same materials as in Figure 4.16. From top left to bottom right: fluorescent yellow, red, green, pink, day-glo red, and white paper. The input wavelength λ_i is left to right, and λ_o is top to bottom.



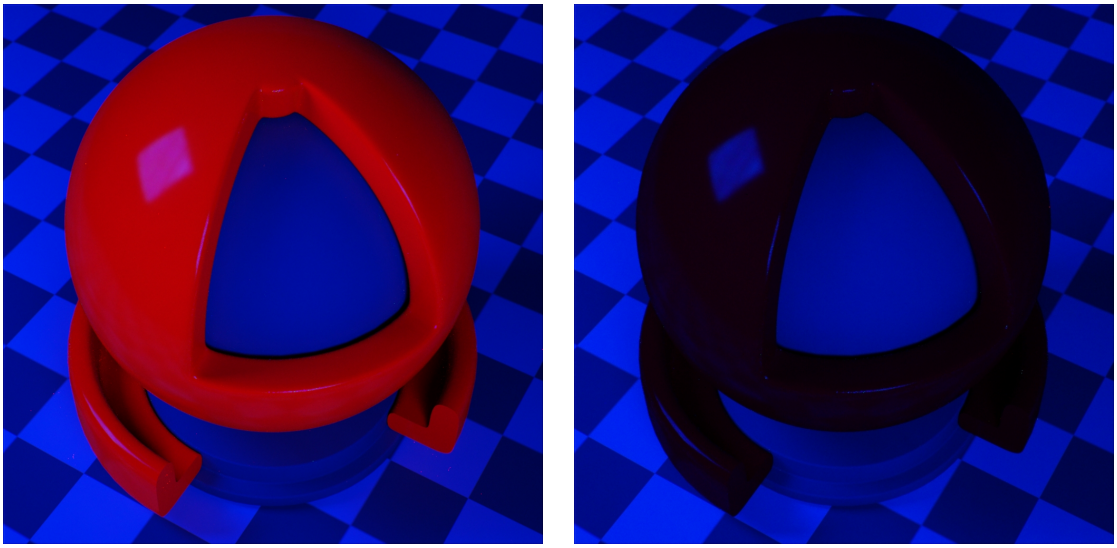
(a) Yellow (b) Red (c) Green (d) Dull Red (e) Paper

Figure 4.18.: Renderings (top row) and photos (bottom row) of different materials under UV light (400 nm).



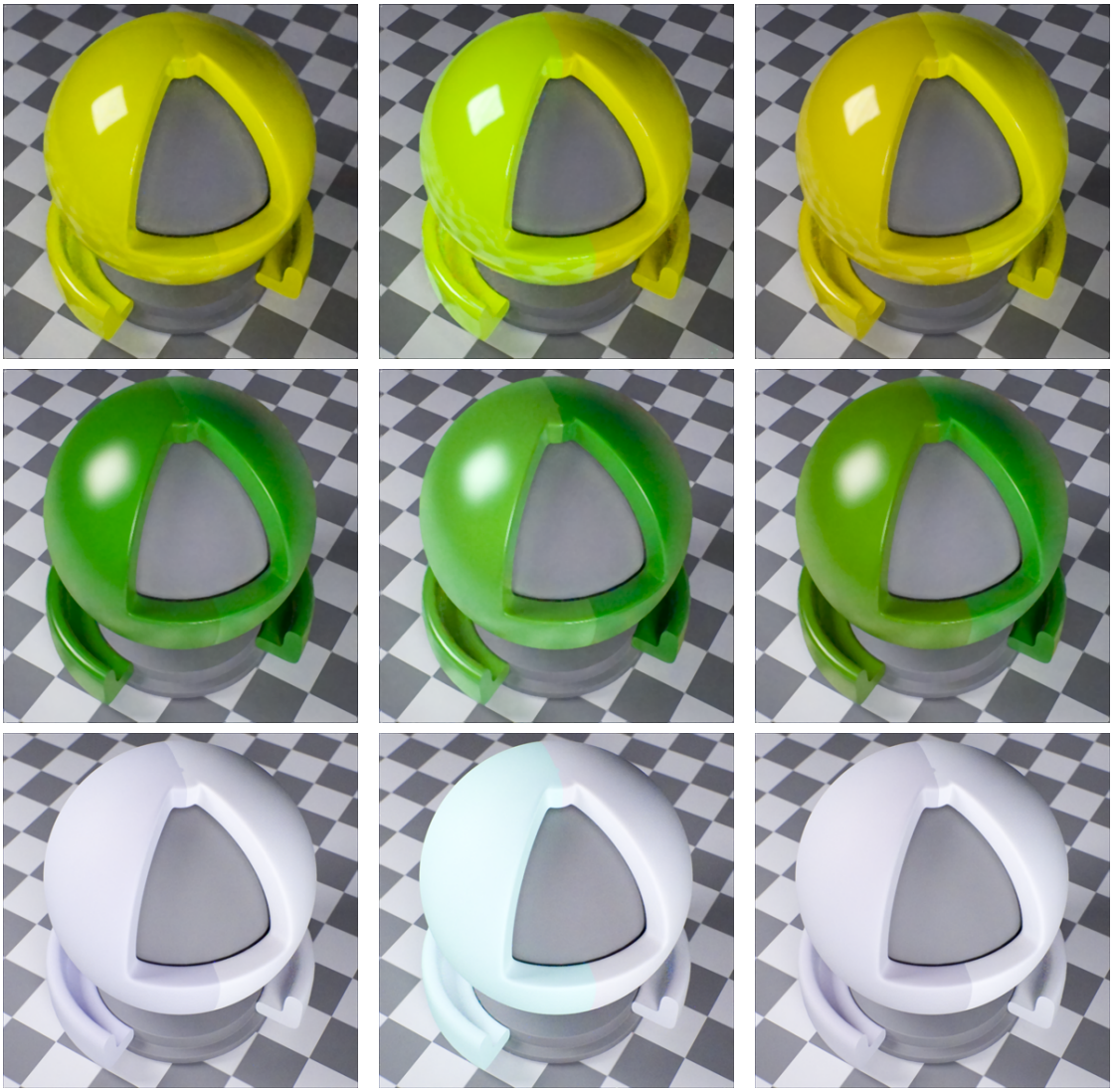
(a) Yellow (b) Red (c) Green (d) Dull Red (e) Paper

Figure 4.19.: Renderings (top row) and photos (bottom row) of different materials under 5600K illumination.



(a) bispectral BRRDF (b) RGB BRDF

Figure 4.20.: Measured fluorescent red bispectral BRRDF (a) compared to a simple RGB vector valued BRDF (b) under blue illumination. Note the cross-color reflectance from blue to red in the case of the full bispectral BRRDF. The RGB BRDF cannot represent these complex color shifts and fails to reproduce the fluorescent effect.



(a) Spectral (16 bands) (b) RGB×RGB (c) RGB

Figure 4.21.: Comparison renderings using 3 different measured fluorescent materials. Full bispectral BRRDF measurements (right half of each sphere) are compared to spectral measurements, RGB×RGB reradiation matrices, and standard RGB BRDFs.

“Many people regard arithmetic as a trivial thing that children learn and computers do,...”

D. Knuth, The Art of Computer Programming, Vol. 2

5 Ray Tracing Precision

One of the seemingly most trivial long-term problems in ray tracing is self intersection. It has been well known for a long time and studied in literature quite a lot (e.g. [WPO96, Wäc08]) and results from numeric inaccuracies in floating point arithmetic. It describes the situation, when a secondary ray is unfortunate enough to start below the actual surface due to rounding errors and quantization. This is just one example where special code is needed to work around calculation errors. Incorrectly reported intersections can also result in light leakage through a triangle. Possible workarounds include ε-thresholds in the form of offsets which are applied to the intersection points, or re-evaluation of the intersection in double precision.

This gives rise to the question whether general floating point arithmetic could be replaced by a specialized number system to save area and power on specialized hardware and at the same time be able to give exact error bounds and warranties about the intersection point.

In this chapter, we compare a variety of common ray/triangle intersection tests with respect to precision and their suitability for a fixed point implementation in custom hardware. The division as most complex operation will receive some extra attention and the impact of approximation is studied. Finally, one particular

ray/triangle test turned out to give the best results and an example implementation is given in C as well as a pipelined VHDL version.

Related Work. The most general algorithm to compute visibility is ray tracing [Whi80, Gla89, Shi00], where one differentially small ray of light is traced through the scene, i.e. a line segment is intersected with the geometry and the closest intersection is returned. A lot of work has been done to make ray tracing fast, see for example Havran's thesis [Hav01] and Wald et al.'s state of the art report [WMG+07] and references in both these documents for a starting point. A main drawback of ray tracing is the random memory access, which can easily be the bottleneck of the whole computation [Dre07].

A coherent way to determine visibility of a whole raster at the time is rasterization. This technique iterates over all geometry and intersects it with a pixel raster by scanline conversion. The closest intersection is chosen by a z-buffer, where distances to the eye point are stored. A particularly nice feature of this method is that it potentially has a constant memory footprint (the z-buffer), independent of geometric complexity. This is not true any more as soon as bounding volume hierarchies are used to accelerate geometry sorting, or transparency comes into play. Raster-based methods have their application mostly in games, as raster points don't always lie exactly where an underlying algorithm needs the evaluation.

The Reyes architecture [CCC87] is worth a special mention here, as it has been very successfully used in movie industry ever since it was introduced. This algorithm is using stochastic sampling to include motion blur and depth of field effects. Subdivision surfaces and displacements are used to create sub-pixel-sized micro-polygons which give a lot of geometric detail.

Although introduced to create images from a view port, rasterization can be used to compute various secondary effects in the same way. This includes for example shadow maps for direct illumination, and cube maps and convolutions for specular and glossy reflections.

The main difference between ray tracing and raster-based algorithms is the order in which computations are performed. While ray tracing can generally explore the whole path space at once and precisely cull away unneeded geometry, raster-based approaches exploit coherence much better and perform all computations on a block of memory (i.e. a texture) in one block after which the memory can be freed. On the other hand, Monte Carlo ray tracing comes with a complexity independent of the dimension (see Section 2.3), whereas always processing the full raster results in exponential running time.

Advanced composite techniques employ precomputation of potentially visible sets (PVS), e.g. [BMW+09], or use baked visibility maps [PFA+10] for precomputed radiance transfer (PRT) [KSL05].

Geometry. To represent geometry in a digital form, there exist a few approaches. These can be classified into explicit and implicit representations. Implicit ones,

such as distance fields and iso-surfaces only give information about the surface location with respect to a point to test. Explicit representations on the other hand give direct access to the surface points. The simplest one, which is often used in real-time rendering and fast ray tracing, is triangle meshes. This is sometimes done in vertex-index form, where shared vertices are only stored once and triangles only refer to it via an index. Real-time ray tracing systems often store their meshes in redundant 9-floats per triangle form, duplicating the vertex data, to increase locality in memory accesses. If this piecewise linear (C_1-continuous) representation is not smooth enough it is common to interpolate per vertex shading normals.

More sophisticated smooth geometry can be created by subdividing polygon meshes, e.g. using Catmull-Clark subdivision [CC78]. This geometry can then be approximated by b-splines [LS08] to gain random access to sampling points on the surface. The same is given for non-uniform rational b-splines (NURBS) [PT95], another common way to store geometry. This property can be used to create quad or triangle meshes for simple rendering of these high-level primitives. Random access is especially important when level-of-detail (LOD) is used to simplify the meshes in unimportant areas, e.g. far away from the camera.

A special case worth mentioning is the class of Bézier patches, especially bicubic Bézier patches. These consist of 16 points arranged in a 4×4 grid, where only the corner points lie on the surface, and the rest is used to define the tangent planes.

On top of these surfaces, additional detail can be created using bump- or normal maps, or parallax mapping [Tat05]. These techniques are increasingly better approximations of displacement maps, where a surface is sampled with a sufficiently large number of vertices, which get displaced along their normal by a value read from a displacement texture. This way, a lot of detailed geometry can be created.

5.1. Arithmetic

Looking at our problem definition above, it now remains to choose an appropriate number system. There are two issues to be identified as challenge in Equation (5.5). The first is the range of the variables, where the volume will be the largest. The other is the division, which is slow in some number systems. For example in Cuda, computation of the reciprocal is still four times as expensive as multiplication or addition [NVI09, Sec. 5.1].

Integer, fixed point arithmetic is very simple to implement on custom hardware, and the analysis about precision can easily be done. The only concern is the dynamic range of intermediate values in the calculation, if excess bit widths have

to be avoided. This has been studied in the context of ray/triangle intersection by statically discarding bits [HK07, HRB+09].

Rational numbers seem an intuitive way to handle the division. For ray/triangle intersection, however, there is really only one division to be done. This would be necessary with rational numbers as well in order to convert the output result at the very end. The same applies when using polynomials for numerator and denominator [Knu81].

Floating point (ANSI/IEEE Std 754-1985) is the most commonly used number format on computers. It is the base of CPU's SSE instructions (e.g. i686, PowerPC, IBM Cell) as well as general purpose GPU computing, and has been studied in depth [Gol91].

For single-precision, the 32 bits are interpreted as $se_7e_6 \cdots e_0 m_{22} m_{21} \cdots m$ and the corresponding value would be $f = (-1)^s \cdot 1.m_{22}m_{21} \cdots m_0 \cdot 2^{e-127}$, in the regular case (i.e. no denormalized number or not-a-number or infinity is indicated).

That is, the range of representable numbers is divided in two stages: the mantissa bits m are the equispaced fixed point, and the exponent bits e switch the scale to larger steps when moving away from zero. As an intuitive error measure for calculations, *units in the last place* (ulp), are used. The advantage of this measure is that it depends on the exponent, and thus gives a meaningful error for the full range of values. This has been exploited for ε-tricks in ray tracing [Wäc08].

Logarithmic numbers are a bit similar to floating point numbers. They store the logarithm of the absolute value of the number along with its sign. This approach promises fast division by just subtracting the logarithms. Regular summation gets a lot more complicated though, and we want only one division per ray/triangle test, but possibly three dot products. So just to avoid the division, we need one logarithm and one exponentiation, to convert to and from the logarithmic number system. These calls are unfortunately slower and less precise than a regular reciprocal (e.g. the regular Cuda `log2` has an error of 3 ulps, and only the approximate function `__logf` achieves the same speed as the division). So this number system is only useful for us, if very cheap approximations of logarithm and exponentiation can be found.

5.1.1. Approximate Computation

To approximate functions numerically, there exist various iterative methods. Most of them are based on series expansions which sum up simple to evaluate terms until the desired precision is met.

The Taylor series is a fundamental series expansion which uses the derivatives of the function at a point x_0:

$$f(x) \approx \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n$$

is also called Maclaurin series for $x_0 = 0$. Conditions for convergence and order of the approximation error have to be evaluated for the particular function and the domain x lives on.

Newton's method is a general technique to find the zeroes of a function by improving an initial coarse estimate. It works by evaluating the first order Taylor approximation (i.e. the tangent) at the coarse guess and using the zeroes of the tangent to find a better guess. This can be shown to have quadratic convergence and is especially useful to refine division (Newton-Raphson division).

Complementing this, it is possible to track the ranges of the required intermediate values for integer and fixed point, as well as the errors caused by quantization and rounding, using some well-studied techniques.

Interval arithmetic can be used to move an interval through the computations (see for example [Moo66, Kea96] for an introduction):

$$\mathbf{x} + \mathbf{y} = [\underline{x} + \underline{y}, \overline{x} + \overline{y}], \tag{5.1}$$

$$\mathbf{x} - \mathbf{y} = [\underline{x} - \overline{y}, \overline{x} + \underline{y}], \tag{5.2}$$

$$\mathbf{x} * \mathbf{y} = [\min\{\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}\}, \max\{\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}\}], \text{ and} \tag{5.3}$$

$$1/\mathbf{x} = [1/\overline{x}, 1/\underline{x}] \text{ if } \underline{x} > 0 \text{ or } \overline{x} < 0, \tag{5.4}$$

where $\mathbf{x}$ and $\mathbf{y}$ are input intervals. While this is great to propagate uncertainty about the input, error bounds are always worst-case and tighter bounds for a more restricted input might exist. This is addressed by an extension which uses the first order Taylor expansion, called *affine arithmetic* [dFS04].

5.1.2. Division

Division it is the most complex and costly operation in ray tracing, both in terms of die area and clock cycles. There are lots of efforts to make this instruction faster, e.g. by replacing the division by an approximate reciprocal and requiring additional iterative refinement. The Cuda platform for example generates an error of 2 ulps for x/y and 1 ulp for $1/x$ if the switch `-prec-div=true` is not explicitly given [NVI09]. Unfortunately, distance calculation (via division) and rejection of the intersection point by depth test are closely related and thus this part is also very sensitive in terms of precision. This is why this section takes a closer look at

some approximations and the impact on the intersection point. Formally, we want to calculate the unsigned division

$$n/d = \left\lfloor \frac{n}{d} \right\rfloor, \text{ for } d \neq 0,$$

and expect the result for signed input to have the sign of $n \cdot d$. Additionally, we are not interested in the remainder, but need to make sure that the result already includes all the necessary precision.

Precise calculation in fixed point. If we want to store the largest and the smallest possible result, the fixed point division would occupy

$$\begin{aligned} \texttt{Q:q} &:= \frac{\texttt{N:n}}{\texttt{D:d}} \\ &:= \texttt{N+d:(n+D)} \end{aligned}$$

bits, where the notation indicates the number of bits before the decimal points followed by a colon followed by the number of bits after the point. It follows that the distance from a point O to the triangle (a, b, c) with normal n along the ray direction ω

$$t = \frac{\langle a - O, n \rangle}{\langle \omega, n \rangle} \in \frac{\texttt{n:m}}{\texttt{1:2m}} = \texttt{n+2m:1+m},$$

when world space points are quantized in `n:0` and directions in `1:m` bits. Since we are only interested in distances on a discrete grid quantized in `n:0` bits, we can constrain the output to `n+1:0` (one bit more to be able to represent lengths on the diagonal) and optimize the divide unit with respect to that [Han07]. Another optimization is to strip the same number of leading zeroes from the numerator and the denominator before computation [Knu81]. Unfortunately, the number of required clock cycles is still in the order of output bits, which is usually much more than needed for the rest of the triangle intersection test.

Approximate reciprocal in fixed point. A fast, coarse approximation of the reciprocal for fixed point values can be obtained by a right shift [War02]. This is precise when dividing by a power of two. We can also take advantage of the bits which are discarded by the right shift and use these for a linear interpolation to the result when dividing by the next power of two. The c source code to calculate the approximate reciprocal of a 32-bit fixed point number in $[0, 1)$ is as follows:

```
uint32_t arcp32(uint32_t d)
{
  // no interpolation:
  // return (1 << (__builtin_clz(d)));
```

```
  const uint32_t i = __builtin_clz(d);
  // linear interpolation:
  const uint32_t mi = 31 - i;
  const uint32_t m = d ^ (1 << mi);
  uint32_t ms;
  if(mi > i) ms = m >> (mi - i + 1);
  else       ms = m << (i - mi - 1);

  return (1 << i) - (ms);
}
```

Where `__builtin_clz()` is the gcc intrinsic to count leading zeroes. The corresponding function for the intel compiler would be `_bit_scan_reverse()`. The result of this function can then be refined using Newton-Raphson iteration:

```
uint32_t arcp32_newton(uint32_t d)
{
  int64_t r = arcp32(d);
  // newton iteration + rounding: (int)(r*(2.0 - r*d) + 0.5)
  return (r*(0x100000000ll - r*d) + 0x40000000) >> 31;
}
```

Repeated evaluation of this iteration might well be worth the effort, as it has quadratic convergence. Error plots for the approximate reciprocal and after application of one iteration Newton-Raphson can be seen in Figure 5.1.

5.2. Analysis of Ray/Triangle Intersection Tests

As an essential element of any ray tracer, there are a lot of ray/triangle intersection tests, and a lot of publications have been made about it (e.g. [Bad90, MT97, Jon00, HM00, GD03, Chi05, KS06, Woo06]). Just about all approaches thinking about a ray and a triangle have been tried, and applied to various specialized architectures. This includes barycentric coordinates [Bad90], spatial triple products [MT97, GD03] (one cross and one dot product), as well as Plücker coordinates [Jon00], and all that also vectorized in SSE-code [Ben04].

When thinking about all these tests in a formal way, it can be quickly shown that all these approaches are algebraically equivalent, just different in the way of thinking about the problem [Eri07].

Formally, we search the distance t a ray has to travel along direction ω starting at the ray origin O, such that the intersection point $h = O + t\omega$ will lie in the triangle spanned by the vertices $a, b,$ and c, if there is an intersection. To test a hit point for inclusion in a triangle, the barycentric coordinates $u, v, 1 - u - v$ can be used

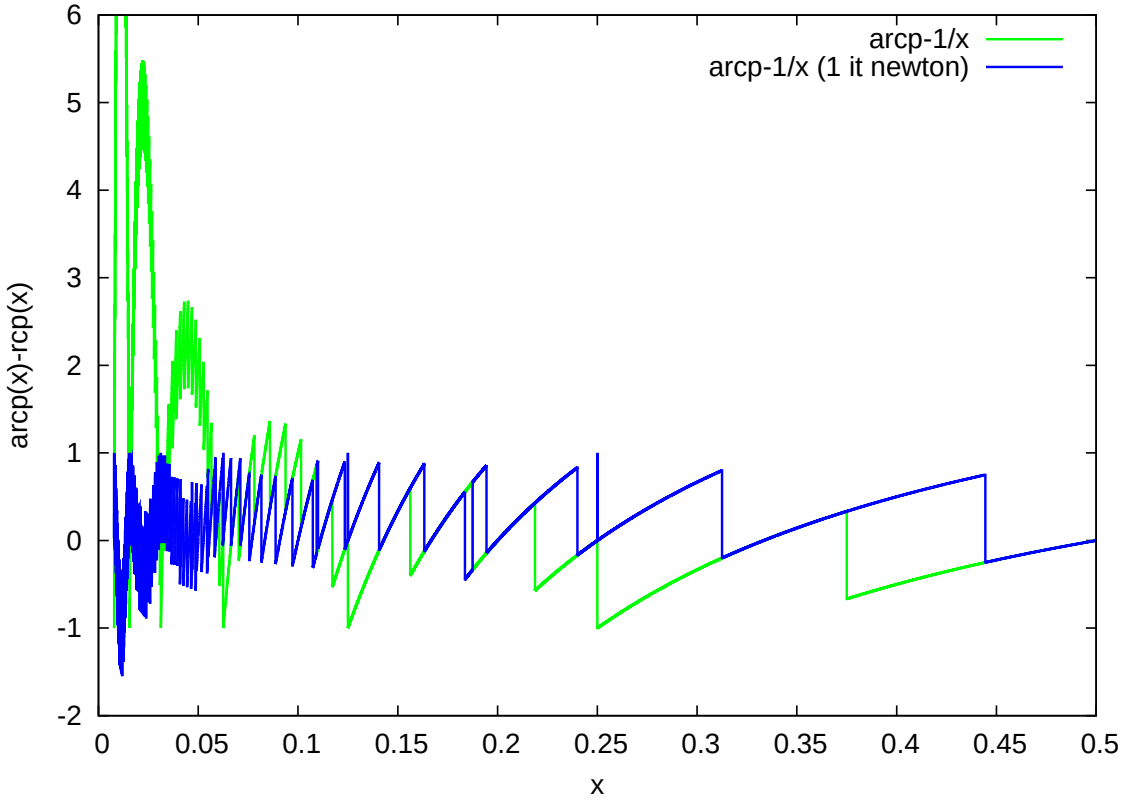


Figure 5.1.: Approximate fixed point reciprocal for 32-bit fixed point values in $[0, 1)$ (interval only shown up to 0.5). Even after one iteration Newton-Raphson, the error still exceeds 1 ulp for small input values. This will show up in the triangle intersection at grazing angles, where $\langle \omega, n \rangle$ is small.

and the point is inside the triangle, iff $u, v \geq 0$ and $u + v \leq 1$. The barycentric coordinates are defined as the ratio of the areas A:

$$\begin{aligned} u &= \frac{A(a,h,c)}{A(a,b,c)} \\ v &= \frac{A(a,b,h)}{A(a,b,c)}. \end{aligned}$$

Extending this to the volumes V yields:

$$u = \frac{V(O,a,h,c)}{V(O,a,b,c)} = \frac{\langle \omega, (h-a) \times (c-a) \rangle}{\langle \omega, (b-a) \times (c-a) \rangle} \tag{5.5}$$

$$t = \frac{\langle a - O, (b-a) \times (c-a) \rangle}{\langle \omega, (b-a) \times (c-a) \rangle}, \tag{5.6}$$

and these volumes can be expressed in multiple ways (as faces of the pyramid triangle/ray origin [Chi05], barycentric coordinates [Bad90], volumes [GD03], determinants [MT97], or Plücker coordinate tests [Jon00]), since the cross product can be expressed in terms of area, or the normal vector:

$$n = (b - a) \times (c - a) \text{ and } ||n|| = 2 \cdot A(a, b, c).$$

The remainder of this section describes some common ray/triangle intersection tests and discusses their properties with respect to precision.

5.2.1. Barycentric Coordinates-based Tests

Möller and Trumbore [MT97] as well as Guigue and Devillers [GD03] in their addendum describe a ray/triangle intersection test based on barycentric coordinates. Both classify a point by testing the barycentric coordinates and then perform a division to calculate the distance of the intersection point. The difference is that Möller and Trumbore calculate the determinant of a matrix consisting of edge vectors and the ray direction, whereas Guigue and Devillers calculate signed volumes of tetrahedra.

5.2.2. Badouel's Test

This test has been described by [Bad90], optimized by [Wal04] and implemented in integer arithmetic in VHDL [HK07]. It is also based on barycentric coordinates, but uses some clever precomputation and projection to a 2d domain before evaluating the barycentric coordinates, which saves instructions. Since the intermediary values are stored instead of the original triangle to keep the same memory footprint, animations are a bit harder to achieve. Also these values are hard to quantize without loss of precision in the case of fixed point arithmetic.

5.2.3. Plücker Coordinates-based Test

This test was introduced in [Jon00] and is based on the theory of Plücker coordinates. First, two lines are transformed into their Plücker coordinate representation (which consists of six numbers for 3d lines), and then the relative orientation can be tested by an inner product in that space. A brilliant summary can be found on Christer Ericson's Blog [Eri07], along with an explanation why this is actually equivalent to the barycentric coordinate approach. Since expanding a triangle to three Plücker coordinates either uses a lot of memory for precomputed values or has to be amortized by intersecting a lot of rays at the time, this test has mainly been used in the context of ray packets in a SIMD fashion.

5.2.4. SSE-based Tests

Employing vector instructions to speed up computations by either intersecting multiple rays with one triangle at the time, or one ray with multiple triangles has received quite some attention (e.g. [Wal04]). Unfortunately these instructions often operate on a subset of the full IEEE floating point standard (there are no denormalized numbers), so these tests are usually less precise.

5.2.5. Transformation-based Test

It is possible to reduce a general ray/triangle intersection to the intersection of a transformed ray with the unit triangle (i.e. with vertices $(0,0,0),(1,0,0),(0,1,0)$). This was described in [HM00, AMH02] and later used as a clever way to reduce die area in custom hardware, by reusing the transformation unit [Woo06]. On the downside, this involves a 3×3 matrix inversion and the transformation of the ray as other sources of numerical inaccuracy.

5.2.6. Chirkov-Style Test

This test is based on Chirkov's article [Chi05]. One of the ideas is to use the three planes spanned by the ray origin and each edge of the triangle to test the ray direction for intersection with the triangle. It turns out that the resulting equations are algebraically equivalent to the tests for the barycentric coordinates, with one important difference. The intermediate results (the normals of the three planes) directly correspond to intuitive quantities in world space, and therefore the precision analysis is simplified. So if we can guarantee to be able to classify a point correctly as inside or outside the triangle given a quantized edge plane normal, this test will operate correctly, even at reduced precision. Naturally, coarser quantization comes at a cost in terms of precision, but in this case the quantization gap can be visualized in world space, as margin around the boundary of the triangle (see Figure 5.2). This property can be used to construct a conservative test, i.e. assure that no rays tunnel through triangles.

Deriving a numerically robust variant of the algorithm results in the following steps:

- Calculate the edge normal vectors $e_{ab} := (O - a) \times (b - a)$.
- Calculate the corresponding barycentric coordinate $u := \langle e_{ab}, \omega \rangle / D$ with $D = \langle \omega, n \rangle$, and make sure this dot product is rounded conservatively towards zero for values > 0. Calculate v similarly.
- Calculate $t = abcO/D$ with $abcO = |a, b, c, O| = \langle u, c - a \rangle$
- If just the hit point and not the distance is required, but the division needs to be avoided, this last step can be replaced by calculating the intersection

point using u, v, and sorting by the l_1-norm $||O - h||_1$ to find the closest intersection.

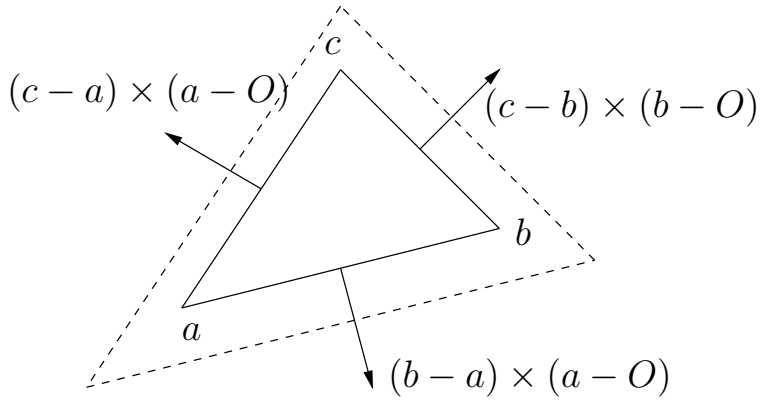


Figure 5.2.: Inclusion in the triangle is tested by the three dot products $\langle \omega, e_* \rangle$ between the ray direction and the three normals visualized in this figure. The dashed lines indicate the boundary of the area where $\langle \omega, e_* \rangle \leq 0$ in machine precision.

Quantization of the Normal. One source of excess bit widths is the calculation of the normal, as the cross product involves multiplications, which usually double the required bit width. Fortunately, we are only interested in the precise direction of the normal. Actually we only want to classify a point to be on the correct side of the plane, even after quantization of the cross product.

Since the input points are already quantized to world space positions (say, in m bits), it is enough to put the error below one half of a world space unit (to match a correctly evaluated and rounded value):

$$\varepsilon \left(\frac{\langle p - a, n \rangle}{\|n\|} \right) < \frac{1}{2}, \tag{5.7}$$

where a is a point on the plane, n the surface normal and p a point to test.

Suppose we want to strip the normal by s bits. We will introduce an error $\varepsilon = 2^{s-1}$ when round to nearest is applied. In world space, the error we get on the distance measure will be

$$\varepsilon = \frac{3 \cdot 2^m \cdot 2^{s-1}}{\|n\|}. \tag{5.8}$$

This suggests to make the shift dependent on the magnitude of the normal. If we choose s to depend on the largest normal component in a way that $\|n\|_\infty < 2^{s+m+2}$

$$\|n\|_\infty = \boxed{0 \cdots 0 \quad 0_{s+m+2}} \boxed{1_{s+m+1} * \quad \cdots \quad *_s} \boxed{*_{s-1} \quad \cdots},$$

i.e. we keep $m+2$ bits for the normal and choose $s = \max\{0, \lceil \log_2 \|n\|_\infty \rceil - m - 2\}$, we can get an estimate of ε:

$$\varepsilon = \frac{3 \cdot 2^{m+s-1}}{\|n\|} \leq \frac{3 \cdot 2^{m+s-1}}{\|n\|_\infty} < \frac{3 \cdot 2^{m+s-1}}{2^{m+s+2}} = \frac{3}{8} < \frac{1}{2}, \tag{5.9}$$

since the L_2 norm $\|n\| \geq \|n\|_\infty$.

This calculus assumes that the dot product is evaluated precisely, which is easily possible, but will need $3 \cdot 2 + 2^m + 2^{m+2}$ bits to store the result. To save bits here we can strip off some of the least significant bits, but will introduce additional quantization errors. For the Chirkov-style intersection test, this error can be represented in world space, as distance to the edges of the triangles. That is, it is easy to apply conservative rounding in a way so triangle edges will become larger and thus avoid holes between triangles at the cost of some false positives.

5.2.7. Subdivision-based Test

Approximate intersection calculation using subdivision (i.e. approximation of a surface with patches of lower degree) has a long tradition (e.g. [Pat93, CMP96]) and has been applied to intersection calculation for ray tracing [DK06]. This approach lends itself well to integer arithmetic: the triangle (a_0, b_0, c_0) is subdivided into four triangles (a_1, b_1, c_1), the ray is intersected with the four bounding boxes of these sub-triangles, and the algorithm recurses if there is an intersection. The recursion stops if the bounding box size is $[0, 1)^3$ units, i.e. cannot be subdivided any further in machine precision. One disadvantage is obvious, the algorithm will need $\mathcal{O}(\log l)$ steps to terminate, when l is the longest side of the triangle along the axes. This is the best case, when always just one bounding box actually intersects the ray. The worst case occurs when the triangle is viewed under grazing angle, and all boxes of size $[0, 2)^3$ are intersected and all $[0, 1)^3$-sized boxes are missed. The complexity is as bad as $\mathcal{O}(l)$ in this case, as all boxes have to be intersected.

The advantage is that only divisions by two are needed for the subdivision. This operation can be done in a lossless way in floating point arithmetic, within bounds: just decrease the exponent. When integer arithmetic is used or the exponent is already at the lowest value, or the numbers are stored in an absolute scale (i.e. in world space, not as edge lengths), rounding errors occur by shifting out the least significant bit. Holes between triangles are still avoided because two triangles which share an edge will perform the same computations on that edge. This even works for different rays, provided that the ray/axis aligned bounding box intersection works precisely. This is due to the nature of the test which only depends directly on the world-space representation of the primitive to intersect, not the ray or some intermediate value.

When a triangle is subdivided in fixed point, a worst-case rounding error of $\varepsilon_r = 0.5$ (the least significant bit is stripped) occurs. So for example subdividing

rounding mode	$\varepsilon(h)$
round half to nearest even (standard float)	5 ulps
truncation	7 ulps
round half away from zero	8 ulps

Table 5.1.: The effect of different rounding modes applied to a one-dimensional recursive subdivision: worst-case error for the intersection point h, measured in units in the last place.

triangle j into four triangles $j+1$, the following point will be calculated:

$$\begin{aligned} a_{j+1} &:= \frac{a_j + b_j}{2} \\ \Rightarrow \varepsilon(a_{j+1}) &= \frac{\varepsilon(a_j) + \varepsilon(b_j)}{2} + \varepsilon_r, \end{aligned}$$

the two previous errors are added up and a potential new rounding error is introduced. In the worst case, this happens every iteration between the two points with the worst error so far.

Luckily, the default rounding mode of floating point arithmetic avoids exactly this case: rounding is performed to the next even number equally far away. That is, as always only one bit is shifted out, the number is either precise (the sum was even) or exactly in the middle (the sum was odd). So rounding up in only half the cases distributes the rounding error up and down and it is thus cancelled out stochastically.

The effect of this rounding mode can be seen in Table 5.1. The first scenario for this test is as simple as possible. A one-dimensional interval has been subdivided given a certain coordinate t (in analogy to the barycentric coordinates u, v for the case of the 3d-triangle). The bits of this coordinate are directly used as path through the recursion and the corresponding point of intersection h is calculated this way and directly using the closed-form formula. The error of the recursive method is then measured in units in the last place. The test is run over all input parameters in the range of floating point numbers in $[1.0, 2.0)$, which is equivalent to a 23 bit fixed point number.

5.2.8. Look-up table-based Test

A reader of Dr. Dobb's Journal suggested to use a precomputed look-up table for small (16×16 pixels) triangles in 2d [Had00]. This thought experiment ended up with 8 MB of memory. Also, memory accesses seem to be the limiting factor on new computing architectures, and for 3d there is no intuitive smallest voxel unit, except the machine resolution. So as interesting as it sounds, it will not be feasible this way. Nevertheless, this finite way of thinking about geometry is the same that

inspires voxel-based renderers, such as sparse voxel octrees (as made popular by John Carmack, also see Section 5.3).

5.2.9. Improving Shading Normals

No matter which triangle intersection algorithm is used, it might be desirable to use per-pixel normals to make the geometry appear smoother. Shading normals are a powerful tool to make meshes with low triangle counts look smooth. Unfortunately they also introduce a lot of problems, e.g. with adjoint transport [Vea97], or the so called *terminator problem* [WPO96], which can be seen in Figure 5.3 (left). The triangle boundaries are clearly visible due to shadow rays which expose the real geometry. A trick to overcome this (fighting a hack with another hack) is to push the intersection point out of the triangle similar as if the shading normals defined the tangent planes of a Bézier patch.

Such a behavior can be achieved in a few lines of code, as in Figure 5.3 at the bottom. The shading normals `n` are stored in a `struct` of the same type as the triangle `tri`. The intersection point `hit` is updated by projecting the three distance vectors from the three triangle vertices to the intersection point `hit.hit` onto the planes defined by the shading normals. These three candidates are then averaged using the barycentric coordinates of the hit point. This ensures smooth transitions at the edges to the neighboring triangle.

As a side effect, this will also solve the self-intersection problem inside the triangle. But for transparent surfaces, it will actually worsen it and thus can’t be used.

5.3. Finite Precision Geometry

All machine computations have to be done in finite precision. This effectively reduces the 3d space to a finite grid of representable points. It is thus possible to create all geometry in voxels of appropriate size stored in a sparse octree and just intersect this geometry representation, which is the acceleration structure at the same time (e.g. [Han03, CS08]). This way, no more triangle data is needed, procedural geometry can be represented the same way, and data can be modified individually by a user. On the downside, this will use a lot of memory. So to make it possible to ray trace such geometry, a very compact data representation is needed, as well as out-of-core techniques.

For an impression of the data usage, see Figure 5.4. This scene consists of an octree of depth 12, which needs 1.7 GB of memory. Still the detail in the close-up is insufficient. As a node has on average about four non-empty children, memory increases quite significantly when one more level is added.

As an experiment, we created a cache of octree nodes with a memory layout as shown in Figure 5.5. This structure stores data for eight voxels and a pointer to potential children. That is, we store the hash key to the child entry in `data`,

```
float hitu[3], hitv[3], hitw[3];
for(int k=0;k<3;k++)
{ // get distance vectors from triangle vertices
  hitu[k] = hit.hit[k] - tri.v[2][k];
  hitv[k] = hit.hit[k] - tri.v[1][k];
  hitw[k] = hit.hit[k] - tri.v[0][k];
}
// project these onto the shading normals n
const float dotu = fminf(0.0f, dot(hitu, n.v[2])),
            dotv = fminf(0.0f, dot(hitv, n.v[1])),
            dotw = fminf(0.0f, dot(hitw, n.v[0]));
for(int k=0;k<3;k++)
{ // and push the distance vectors out onto the planes
  // defined by the shading normals
  hitu[k] -= dotu*n.v[2][k];
  hitv[k] -= dotv*n.v[1][k];
  hitw[k] -= dotw*n.v[0][k];
}
// the final hitpoint is the barycentric mean of these three
for(int k=0;k<3;k++) hit.hit[k] =
      (1-hit.u-hit.v)*(tri.v[0][k] + hitw[k])
      + hit.v        *(tri.v[1][k] + hitv[k])
      + hit.u        *(tri.v[2][k] + hitu[k]);
```

Figure 5.3.: The terminator problem with shading normals, solved by adjusting the hit point similar as if the normals defined a Bézier patch. The source code shows the simple adjustment.

Figure 5.4.: A car rendered using an octree with texture information at the leaves. The images were created using an octree of depth 12, i.e. the smallest possible resolution is AABB width times 2^{-12}. The close-up to the star reveals this.

```
typedef struct octree_cache_entry_t
{
  uint64_t data; // key << 8 | mask
  uint8_t normal[14]; // 8(nr:2 np:6) 8(nq:6)
  uint8_t cr, cb;
  uint8_t Y[8];
  uint8_t shader[8];
} // 8 + 32 bytes
octree_cache_entry_t;
```

Figure 5.5.: The octree structure stored in one cache line.

together with an eight-bit mask which indicates non-empty children. Additionally, a normal (using 14 bits, as the look-up table approach of [RL00]) is compressed as a point on a cube: `nr` stores two bits indicating the side of the cube (normals are flipped towards the ray, so only three sides have to be stored), and `np` and `nq` give the position on that square. The base texture of the voxel is compressed in YC_bC_r using chroma subsampling. Finally, a shader index is stored per voxel.

This layout enables geometry MIP-mapping (or levels of detail), as the inner nodes already contain all information needed for rendering.

For the ray traversal, we use a combination of a parametric algorithm [RUL00] and path coding [Mor66]. Since we store the Morton code in one 64-bit integer ($3 \times 18 = 56$ and 8 reserved bits), the tree depth is limited to $n = 18$, which means the voxel grid resolution is limited to 4 mm inside a 1 km^3 cube. The tree traversal returns the intersection point along with its box size, so self-intersection can be

effectively avoided.

The Morton code is also used to create a hash for the node cache (as [Eri05]) and to address the voxel position and size when it has not been found in the cache. In this case, the least recently used octree node is removed from the cache to free a slot for the requested node. The data is then streamed from disk or procedurally computed, depending on the input data. Isosurfaces or procedural noise are evaluated using interval arithmetic to create consistent voxel boundaries.

An interesting aspect about such a cache is that it can be used to create infinite detail: the tree traversal has to be adjusted to use entry points at deeper levels in the tree, the Morton code can be used to find the neighbor without stepping up to the parent. This way, the root node and the coarser levels can be swapped out and the cache slots can be used for more local detail.

Figures 5.6 and 5.7 show procedural isosurfaces and perlin noise, respectively. The performance on a single core of an Intel Core2 Duo processor was about 50k–500k rays/second, but varying largely depending on the cache size, the level of detail selection, the camera position, how expensive the procedural computation was, as well as how coherent the rays were cast. While for eye rays the cache hit rates quickly exceed 99% when the camera position is not changing and the cache is large enough, the performance stays underwhelming. Also, a single cache will not scale over multiple cores very well, because of lock contention. It might also be argued that this kind of procedural geometry is not very practical as input, and it would be beneficial if the representation used for rendering would be more closely related to the representation used for modelling. Another concern is the use of secondary rays, which is the only good reason to use ray tracing over rasterization-based approaches. In such an application, the cache hit rates will not be as friendly as reported above.

We will address these issues in the next chapter. Instead of machine-precision boxes, we will use micro-polygons, which are used as smallest units in production renderers. This makes it possible to use acceleration structures which adapt better to the geometry than a mid-split octree, such as bounding volume hierarchies. The caching system will be replaced by a reordering scheme which transparently increases locality of memory accesses and computations.

5.4. Results

To be able to compare the algorithms for different number systems, the input vertices have been transformed to be equally well representable in all systems: the scene has been scaled to fit the range $[1, 1.5)$ in floating point, where the mantissa directly corresponds to fixed point in `[0x0, 0x3fffff]`. The full width of the mantissa, `[0, 0x7fffff]` is then needed to encode the distance to the triangle, as $t \leq \sqrt{3} \cdot$ AABB width.

The results can be seen in Table 5.2 as well as Figures 5.8 and 5.9. This test is designed not only to compare 23-bit fixed point directly to floating point, but is also

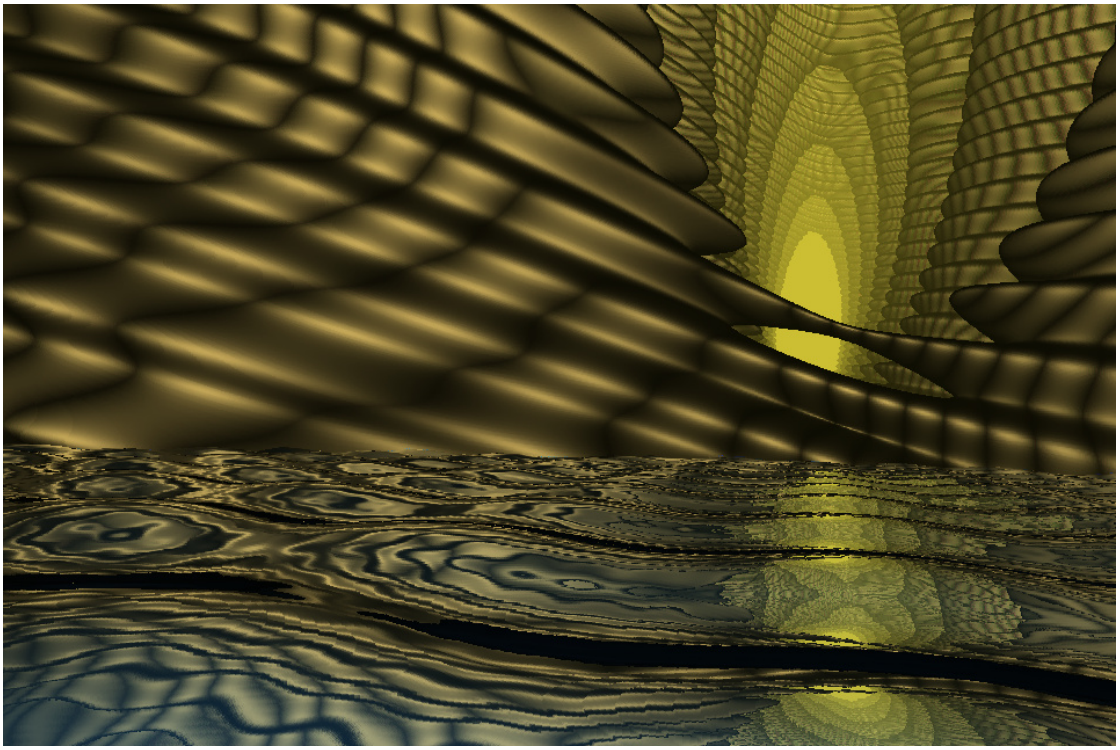


Figure 5.6.: Isosurfaces ray traced in real-time using an octree as cache.

very floating point friendly: inside the bounding box, no switching of exponents is required (actually, it is not even possible). To show the effect of mixing floating point exponents, the bounding box has also been scaled by 10^4 and 10^{-3}. As ground truth, a `long double` version of the Chirkov-style test has been used.

Rays are also generated the same way for floating point and fixed point: the mantissa is randomly chosen and converted to floating point and fixed point rays. In total, 100,000 rays are generated this way and intersected with all triangles (69,451 for the bunny and 12,748,510 for the power plant), resulting in about $7 \cdot 10^9$ and 10^{12} intersection tests for Figure 5.8 and 5.9, respectively. A typical bi-directional path tracer for a global illumination application would require to cast thousands of paths per pixel, i.e. $1280 \cdot 1024 \cdot (5 + 5 + 25) \cdot 1000 = 45 \cdot 10^9$ rays for a eye path and light path length of five bounces and a moderate screen resolution. So assuming only one ray/triangle intersection per ray, the numbers in Table 5.2 give a rough idea of how many wrong intersection one can expect for one image.

The single false positive intersection in the Chirkov-style integer test stems from an intersection on the edges of the triangle. It is not due to distance calculation, as the distance t was always evaluated correctly. Using the approximate reciprocal `arcp32` with one iteration Newton-Raphson (see Section 5.1.2) creates a lot of falsely reported intersections, which can be seen in column *Chirkov int arcp*.

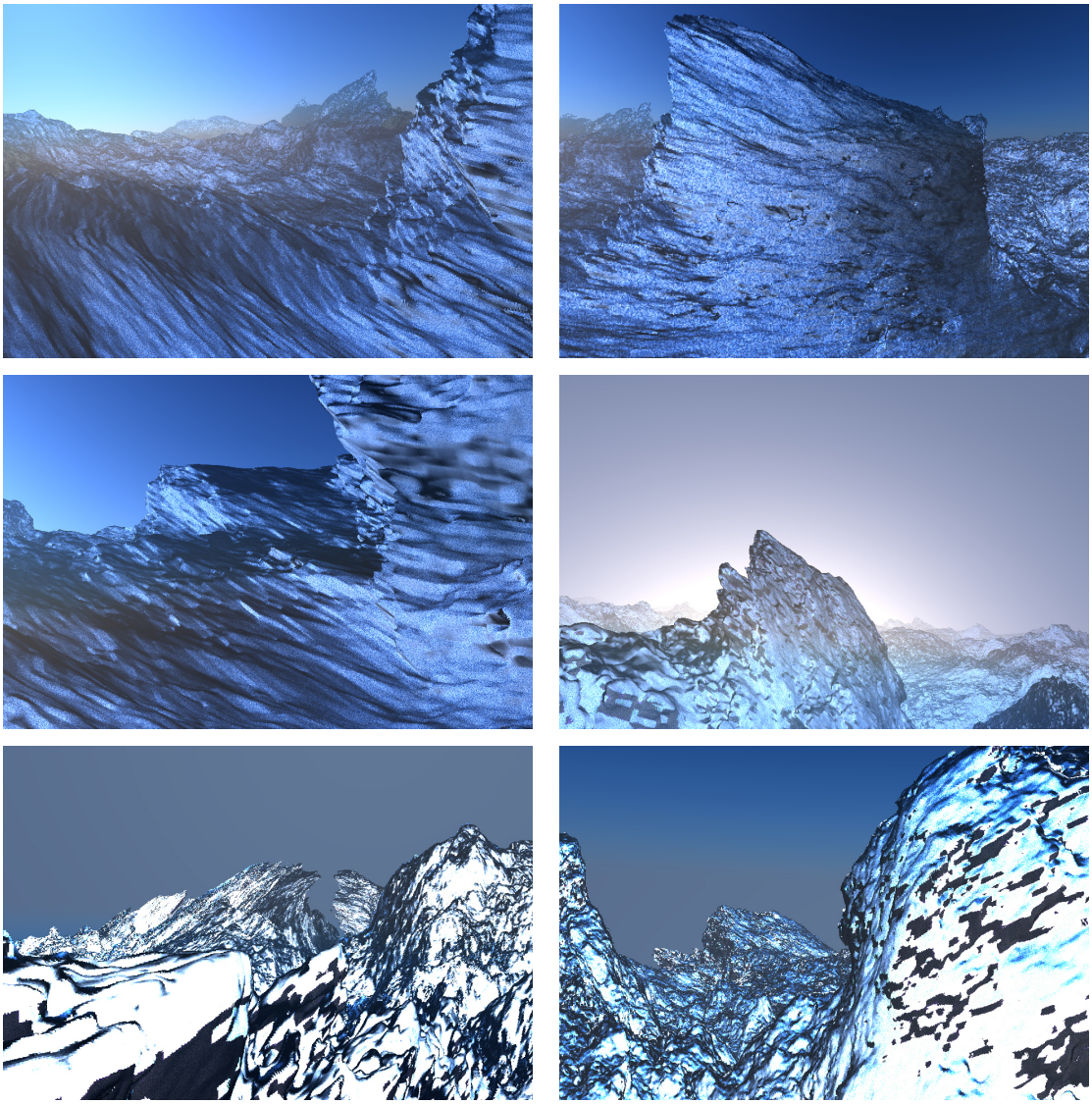


Figure 5.7.: Nine octaves of perlin noise ray traced using an octree as cache. The images are all taken in the same landscape.

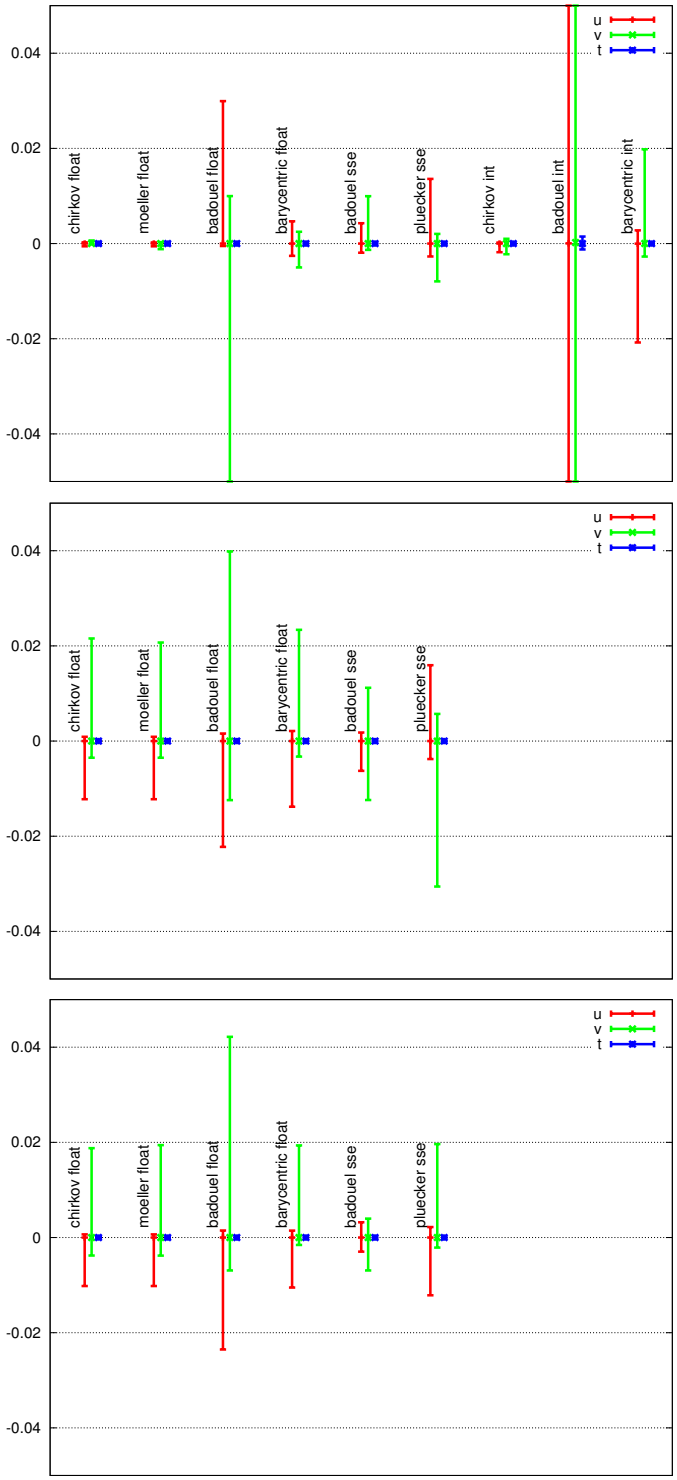


Figure 5.8.: Precision analysis for the Stanford bunny scan model. Top row: the bounding box has been scaled to fit $[1.0, 1.5)^3$. Middle: the same scaled up by 10^4. Bottom row: the same scaled by 10^{-3}. The bars indicate the mean value and the minimum and maximum deviation. The precision is evaluated over the correctly classified intersections. So the SSE version of the Badouel test seems to be more precise, but the wide deviations are cut off because the test failed in that case.

	Error type	Chirkov float	Möller float	Badouel float	barycentric float	Badouel SSE	Plücker SSE	Chirkov int	Chirkov int arcp	Badouel int	barycentric int
$[1, 1.5]^2$	α	1	0	2	0	2	1	1	11	134	2
	β	0	0	1	3	2	2	0	59	150	5
$10^4 \cdot [1, 1.5]^2$	α	1	1	3	1	2	1				
	β	0	0	2	2	2	6				
$10^{-3} \cdot [1, 1.5]^2$	α	3	3	3	3	6	5				
	β	3	3	2	2	2	7				

Table 5.2.: Precision analysis of different triangle tests with respect to α- (false positives) and β-errors (missed intersections). These numbers have been generated using 100,000 random rays intersected with the 69,451 triangles of the Stanford bunny.

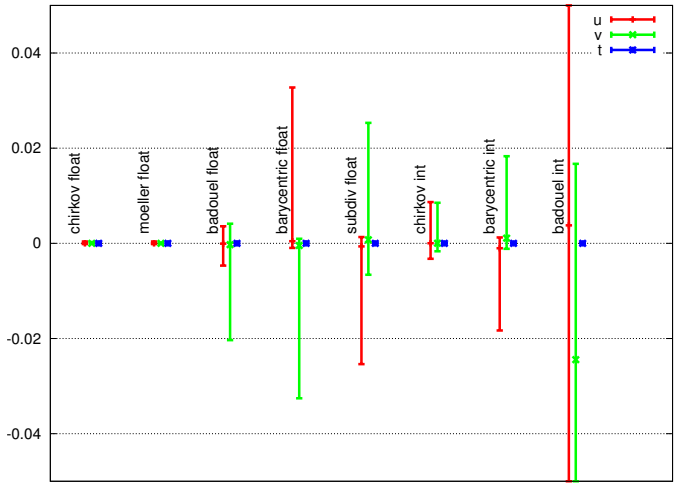


Figure 5.9.: This graph shows the same statistics as 5.8, but with the power plant model scaled to fit $[1.0, 1.5)^3$. The subdivision algorithm has only been tested on this model, as it contains large triangles which reveal the numerical issues.

5.5. Conclusion

In this chapter, we investigated a number of ray/triangle intersection tests with respect to their suitability for numerically robust hardware implementations. The epsilon-free Möller-Trumbore test does not perform too badly after all. However it exhibits problems when different exponents are in play. The subdivision-based test does not always coincide with the `long double` version. But this test assures that no ray can tunnel through a crack between two triangles. A carefully implemented Chirkov-style integer test gives the same safety, at the cost of some false positives on the edges of the triangle (but fewer as the subdivision-based test). As the dot products used to decide whether the ray passes the edge inside or outside the triangle are calculated the same way for neighboring triangles (permuting the two vertices only gives a sign and is the same even in floating point arithmetic), the worst thing that could happen is that both triangles claim the first intersection. Also, false positives are not as fatal as missed intersections. For example, a ray connecting to the sun might easily transport such a lot of energy that a whole region might be illuminated due to tunneling through the triangle.

The error in the u, v coordinates shown in Figures 5.8 and 5.9 are to be read very carefully: as triangles are very small, the barycentric coordinates can easily refer to a hit point which is not representable in this precision. So most of the time u and v have more bits then could possibly be filled with useful information.

As it turned out to work best for fixed point arithmetic and is thus a candidate for a robust, precise hardware implementation, the C source code for the Chirkov-style test can be found in Appendix A.1. A pipelined VHDL version of this test (8 clock cycles deep, synthesizes with 390 MHz on a Xilinx Virtex5 FPGA) is listed in Appendix A.2. This code has been tested in the context of a VHDL implementation of a Quad-BVH [EG08, DHK08] ray tracer.

"If displacements are nice and easy, you're writing a rasterizer. If it is ugly and slow, it's a ray tracer"

Pete Shirley at Eurographics 2009

6 The Rayes Architecture

This chapter discusses an approach to ray trace micro-polygon geometry with motion blur, defocus, and global illumination [HKL09, HKL10]. This ray tracing scheme is able to handle highly complex geometry modeled by the classic approach of surface patches tessellated to micro-polygons, where the number of micro-polygons can exceed the available memory. Two techniques allow us to carry out global illumination computations in such scenes and to trace the resulting incoherent sets of rays efficiently. First, we rely on a bottom-up technique for building the bounding volume hierarchy (BVH) over tessellated patches in time linear in the number of micro-polygons. Second, we present a highly parallel two-stage ray tracing algorithm, which minimizes the number of tessellation steps by reordering rays. The technique can accelerate rendering scenes that would result in billions of micro-polygons and efficiently handles complex shading operations.

In movie production, extreme geometric detail, complex shaders, and motion blur are needed to obtain visually compelling images. The Reyes architecture [CCC87] successfully deals with these challenges using a rasterization approach. The use of physically-based ray tracing is getting more and more common in the movie production, partly because the artists' experiences from real-world lighting design can be easily carried over. For this benefit, even the long render times are accepted [Gri09]. To retain the strengths of the Reyes architecture in

a general ray tracing setting, we propose a two-level hierarchy approach, using reordering of computations instead of caching. After traversing a top-level hierarchy, rays are sorted to bundle those, which intersect the same bounding volume. Any necessary operation to be carried out in this volume, e.g. tessellation or loading a complex shader or bidirectional reflectance distribution function (BRDF) [NRH+77], is thus performed a minimum number of times. This results in significantly improved data locality, which allows us to fully ray trace computationally complex (procedural) displacements efficiently, i.e. corresponding to billions of micro-polygons without instancing (see Figure 6.1). Existing production pipelines can easily be extended to use our method, since the algorithm works on the same two-level data, such as displaced subdivision surfaces and sub-pixel-sized micro-polygons.

Rendering such scenes requires one to tessellate the free form patches or procedural displacements, which can be quite expensive with regard to computation and memory consumption. We accelerate ray tracing and global illumination by exploiting the two-level hierarchy of such scenes: The top-level hierarchy (Section 6.2) organizes the list of surface patches. After traversing the top-level, all rays are sorted according to patches they possibly intersect, increasing locality and minimizing the number of tessellation steps. The bottom-level consists of the micro-polygons which are tessellated on-the-fly on-demand. The micro-polygons of one patch are diced into a micro-polygon buffer, and a high-quality bounding volume hierarchy (BVH) is constructed in linear time in the number of micro-polygons (Section 6.1), exploiting the regular topology of a diced patch. Furthermore, the number of tessellation steps during rendering is reduced by adapting the level of detail (Section 6.1.2).

Altogether, the architecture collapses the inherent recursive nature of ray tracing to allow for better vectorization and combines the strengths of tracing ray packets [WBWS01], fast incoherent mono-ray traversal [DHK08], and rasterization: Our technique inherently handles displacements and procedural geometry, supports simple shader authoring and large depth complexity. It optimizes the utilization of memory bandwidth and coherence and furthermore is highly parallel.

Related Work. A lot of work has been done to be able to render complex geometry [SBB+06, LYM07, LYTM08] not specialized for the Reyes architecture. The fundamental assumptions and design principles of the Reyes image rendering architecture have allowed to model and render diverse and complex content, as postulated in the original publication [CCC87]. The concepts were so fundamental, that the many extensions (e.g. [HL90, LV00]) seamlessly complemented the basic architecture. As one of the design principles was to keep expensive ray tracing to a minimum, it is not surprising that the addition of minimal ray tracing turned out to be restrictive. The most recent ray tracing extension was profoundly described in [CFLB06]. With our technique we demonstrate how a ray tracing system can deal with the same complexity in geometry modeling and shading while adding

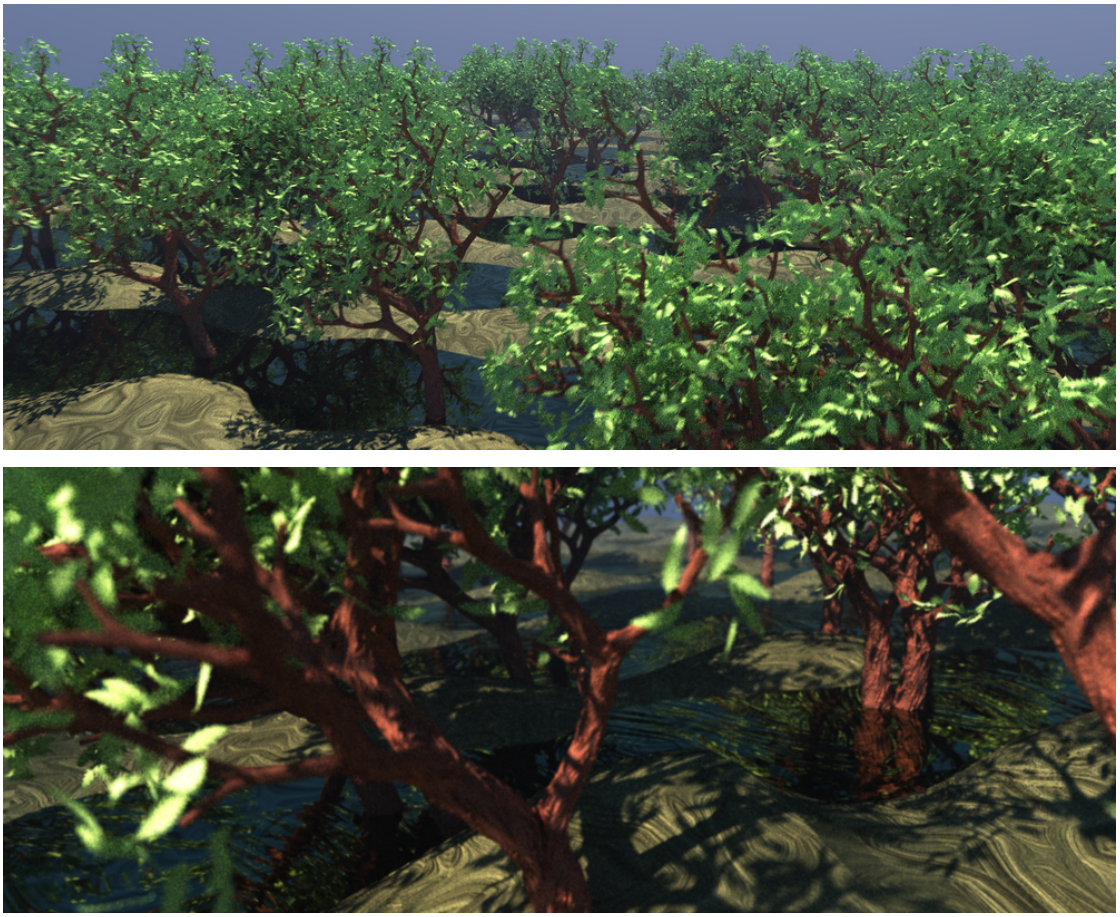


Figure 6.1.: A forest with 100 trees and geometry shaders including displacements for bark and leaves. Fully tessellated, it would consist of over $108 \cdot 10^9$ triangles. Thanks to level of detail on-demand geometry creation and ray sorting, only around 170 million triangles are actually created, without the use of caching. The geometry is created and fully path traced without instancing, evaluating global illumination from the sun and sky, and motion blur at a resolution of 1920×768 with 32 samples per pixel in 5:04 minutes on a 2.83 GHz quad core Q9550 (bottom image).

the benefit of simple Monte Carlo-based global illumination computation using path tracing.

Key to our system is the reordering of rays to increase locality for ray tracing massive data which, in rather general settings, has been investigated before [LMW90, PKGH97, NFL07, BBS+09]. Our approach, however, directly benefits from the intrinsic data locality of the common two-level modeling approach: large surface patches in the top-level, and displacements or procedural details

and complex shading at the bottom-level. This approach is particularly common in games, for example using parallax occlusion mapping [Tat05]. Ray tracing displaced primitives using tessellations [SB87], caches [PH96] or direct grid-like traversal [SSS00] has been investigated in depth, also on the GPU for geometry images [CHCH06]. The GPU can also be used to dice/tessellate Reyes patches [PO08, EL10]. In [BBLW07], an on-demand, recursive BVH traversal scheme for subdivision surfaces without displacements was introduced which is optimized for ray packets. Acceleration structures for ray tracing have been build in complexity below $\mathcal{O}(N \log N)$ before [HMF07]. In Section 6.1 we describe a simple method to explicitly construct a hardware-friendly acceleration structure in linear time.

In our two-level approach rays are reordered, grouping active rays which potentially intersect the same patch. Similar to the approach taken in the Kilauea render system [KS02] the resulting *(ray, patch)* lists can then be processed in parallel without caching strategies.

Related work [HQL+10] evaluates only the first few dimensions of the light transport problem (defocus, motion blur, and limited transparency) on the GPU and achieves about the same time as our system to render one frame.

The Razor architecture [SMD+06] was designed to alleviate similar problems of ray tracing, to obtain better data access and computation patters. It uses current results of that time to accelerate ray tracing for the processors available by then. Today, cache lines (and fetches) get larger and data parallelism is getting wider, GPUs being the extreme example. Thus, linear memory access has become more important, and simple streaming of large blocks is often more efficient than highly recursive tree traversals and on-demand builds with a lot of branches. The Razor system does neither use displacements or pluggable artist-driven geometry shaders, so it is not optimized to avoid these computations, nor does it perform well for the excessive level of detail needed for production. To implement level of detail, the authors use a set of pairs of kd-trees for every two adjacent levels of detail, which are merged together in one acceleration structure. Additionally, at the lowest level, the vertices are stored in a 5x5 grid, which is created on-demand and traversed as in [SSS00]. Our method on the other hand comes along with two levels of hierarchy, has implicit levels of detail (in the upper levels of the bottom-level BVH), and can be diced, displaced and built with improved memory access and data parallelism.

An impressive, cache-based system was introduced by Pharr et al. [PH96, PKGH97]. It relies on a set of different caches for the rays, geometry and textures.

A very fast out-of-core micro-polygon ray tracer running on the GPU [PFHA10] has been used to precompute visibility for production. It uses tessellated micro-polygon meshes with level of detail already applied as input and takes advantage of intrinsic coherence of the rays to maintain very high cache hit rates.

Our ray reordering technique is simpler and transparently increases locality for rays, textures, BRDF data and geometry at the same time. Additionally, as

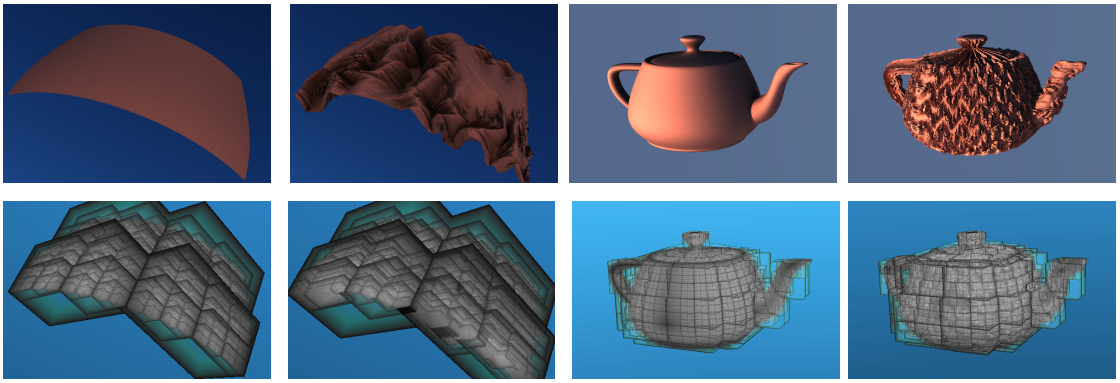


Figure 6.2.: The top row shows a surface patch and the teapot without (left) and with displacement mapping (right). In the bottom row the bounding volume hierarchies implied by the micro-polygon array topology are visualized by rendering them transparently and darkening their contours. Differences between the hierarchies are difficult to spot, which indicates that reasonable displacement does not much affect the efficiency of the implied acceleration data structure.

soon as the required cache sizes get too large, caching did not perform well in our target setting with incoherent rays.

Level of detail (LOD) has been added to both Reyes [CHPR07] and ray tracing architectures, e.g. using multi-resolution meshes [SMD+06] or simplification [YM06]. For ray tracing, the choice of LOD is commonly based on ray differentials [Ige99]. We show that in the case of our architecture, simpler mechanisms may be used.

6.1. Efficient Ray Tracing of Arrays of Micropolygons

To intersect a bottom-level patch with a set of rays, it has to be diced and displaced, evaluating geometry shaders. The resulting micro-polygons are stored in the micro-polygon buffer, which represents $2^m \times 2^m$ micro-polygons as a two-dimensional array of $(2^m+1)\times(2^m+1)$ vertices, where each four adjacent vertices define one micro-polygon.

Surface patches must implement a tessellation method, that computes the micro-polygon vertices by either sampling or subdividing a surface patch, applies trimming and displacement, and stores interpolated (s,t) texture coordinates. Vertices are displaced along interpolated per-vertex displacement normals. To avoid holes between adjacent patches with different level of detail, conservative bounding boxes are needed for coarser tessellations, i.e. coarse displacements have to span all the possible range of the finer ones. This is done by min-max MIP maps on textures and interval arithmetic on procedural noise and patch geometry. Afterwards a loop over all micro-polygons evaluates whether or not the

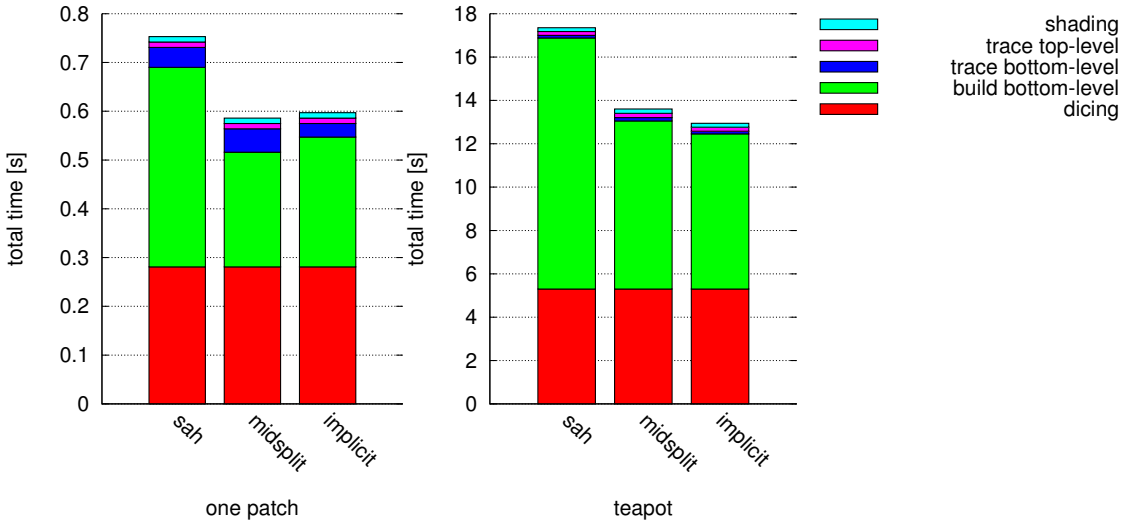


Figure 6.3.: Timings comparing bottom-level construction strategies. On the left, a full SAH build of the micropolygon BVH is done, in the middle, a spatial median was used as split plane candidate, on the right is implicit construction. The one-patch scene and the teapot is as is Figure 6.2 (displaced version). These timings have been taken for primary rays only.

micro-polygon is clipped or trimmed. Unless the micro-polygon is discarded, its bounding box, color from texture, and normal by vertex differences are computed and stored.

Such a tessellation method must be aware of the resolution of the micro-polygon buffer. In case of insufficient resolution, surface patches must be split and the parts have to be processed separately.

6.1.1. Implicit Acceleration Hierarchy in Linear Time

The number of $4^m = 2^m \times 2^m$ micro-polygons and their topology suggest using a complete quad-tree of axis-aligned bounding boxes as acceleration hierarchy for ray tracing.

Constructing the bottom-level hierarchy starts by determining the conservative bounding boxes for each of the 4^m micro-polygons by calling the tessellation method. The bounding volumes of the inner nodes of the hierarchy are updated in a bottom-up manner using min-max MIP maps (similar to [CHCH06]). Trimming is implemented by marking bounding boxes as empty. They do not need to update their parent boxes and can also be handled transparently during ray traversal. Since the memory for the micro-polygon buffer data structure is allocated once for the whole rendering process, we always store the complete tree and do not

compress empty bounding boxes.

Although, in general, complete trees for ray tracing cannot be recommended [Wäc08, Sec. 2.4.1], this concept is very appropriate for tessellated surface patches: Unless the patch is overly curved or extremely displaced, the array topology very well represents spatial proximity as illustrated in Figure 6.2 and results in fast ray tracing (see Figure 6.3).

While the construction time for a spatial acceleration structure typically is $\mathcal{O}(n \log n)$ in the number of triangles [WH06, Wal07], our bottom-up construction of the complete quad-tree is linear in the number of nodes $\sum_{i=0}^{m} 4^i \in \mathcal{O}(4^m)$ and thus linear in the number of micro-polygons of one surface patch. In contrast to [HMF07], this can be done without an explicit input hierarchy and in a non-recursive manner, thus featuring a better memory access pattern.

Rays are then intersected with the acceleration structure using single ray traversal. Improved memory access by tracing ray packets did not pay off at this stage, as even these pre-sorted rays for one patch are very incoherent with regard to traversing the bottom-level hierarchy in the case of path tracing. Similar to the discrete geometry approach followed in Section 5.3, we tessellate down to sub-pixel size and use the boxes of the leaf nodes directly as geometry, as this accuracy is sufficient [DK06].

In order to assess the quality of the implicit BVH construction for patches, we tested two very simple scenes, to avoid the effect of a complicated top-level hierarchy in the timing figures. In Figure 6.3, timings are plotted for a scene containing only one patch, and the displaced teapot scene (see Figure 6.2). The bottom-level ray tracing time can be slightly improved when using a general surface area heuristic (SAH) [GS87] when constructing the acceleration structure for the teapot (0.119 seconds SAH vs. 0.121 seconds implicit). For the simple one-patch scene, our implicit tree can even be ray traced faster (0.041 seconds SAH vs. 0.028 seconds implicit). This might also be due to the fact that our bottom-level QBVH traversal implementation exploits the special memory layout of the implicit BVH, using skip lists. It also explains the difference to the ray tracing time of the midsplit tree (0.048 seconds), which results in quite similar topology.

6.1.2. Crack-Free Level of Detail Geometry Approximation

While it is common understanding that the availability of different levels of detail can vastly enhance rendering efficiency, care needs to be taken in order to avoid rendering artifacts due to the approximative nature of simplifications.

The level of detail is selected by choosing the resolution parameter m (see Figure 6.4) from Section 6.1 as the smallest m such that $4^m \geq R/\text{spp}$, where R is the number of rays that intersect the axis-aligned bounding box of the patch under consideration and spp is the number of samples per pixel. In order to ameliorate the self-intersection problem, secondary rays are offset by $\varepsilon = l/2^m$ along normal direction, where l is the length of the longest side of the bounding box of the

Figure 6.4.: Illustration of the surface approximation by selecting the level of detail $m = 1, 2, \ldots, 8$ (from left to right).

actually intersected patch.

The intuition behind the selection heuristic is simple: Regions with high ray density (for example regions traversed by a bundle of specularly reflected rays) require a finer level of detail as compared to regions with less rays (as for example after diffuse reflection).

The selection heuristic does not guarantee that a ray intersects adjacent patches at the same level of detail. We therefore require all bounding boxes to be conservative. In our case this is guaranteed by the min-max MIP maps from Section 6.1.1, the use of interval arithmetic on the noise function, and the convex hull property of the control polygon of the Bézier patches. If now adjacent patches share an identical boundary, bounding boxes of adjacent patches at different levels of detail are guaranteed to touch at least and overlap most of the time. As a consequence, using intersections with the bounding boxes of the bottom level hierarchy instead of intersections with micro-polygons guarantees hole free rendering.

While this method is simpler than other state of the art techniques like for example stitching together adjacent geometry [CFLB06, Sec. 6.6] or DiagSplit [FFB+09], it requires to select a sufficiently fine level of detail such that the resulting hole free approximation of the surfaces by boxes remains invisible [DK06].

For directly visible geometry the selection heuristic results in marginally smaller than pixel-sized boxes, which reliably avoids level of detail popping artifacts during animation. However, if for example a patch only partially overlaps the viewport, then the visible part of the patch will have a much higher ray density as compared to what is determined by the selection heuristic. Note that due to the X-ray characteristic of the top-level intersection candidate selection, our heuristic does not fail in the case of partial occlusion: the rays see through the occluder as far as LOD selection is concerned. In a similar way, shading differences due to changing level of detail may become visible for secondary effects as for example self-shadowing.

While the selection heuristic rarely does not determine a sufficiently fine level of detail, ray differentials [Ige99] provide a widely used alternative and are easy to approximate in our system as all rays of a generation are traced at once (see Section 6.2.2). This allows for selecting the level of detail depending on the smallest distance between individual rays and complementing the coarse levels with directional opacity information as in [LBBS08], or using frequency domain filtering [HSRG07].

Because the bottom-level acceleration structure is a complete quad-tree, the

upper levels always represent the coarser levels of detail. Shading information such as color from texture, uv coordinates, and normals from vertex differences can be filtered on demand and rays can be terminated at individual levels of detail. Note that this will result in a slightly more complicated memory access pattern.

6.2. Reordering Rays

Our rendering system follows the two-level modeling approach commonly applied by artists who create coarse geometry using free form surfaces and then refine it by adding geometric detail and complex shaders.

6.2.1. Top-Level Hierarchy

In order to minimize the number of dicing operations we introduce an active ray buffer. Directly after traversing the top-level hierarchy, all potential intersections (maximum N per ray per iteration) are sorted by patch ID. Then the geometry and the shaders of each patch are prepared only once for this iteration. Lastly, all rays associated with the patch are now processed in one block of computation. As new rays might be generated due to recursive ray tracing the loop of top-level traversal, sorting, tessellation and bottom-level processing is iterated as needed. Finally, the result of the first hit point is added to the accumulation buffer.

Partitioning the computation this way greatly facilitates parallelization in each of the four processing steps. Even for global illumination computations where rays are typically incoherent after the first bounce, the explicit sorting step maximizes coherence for further processing.

The top-level hierarchy is represented by a Quad-BVH (QBVH or mBVH) [DHK08, EG08], whose leaves are the conservative bounding boxes of single patches. Ray tracing starts by generating rays and storing them in an active ray buffer. The traversal of the top-level hierarchy can then be executed in parallel by partitioning the ray buffer.

Rather than computing the first intersection directly, we gather N intersection candidates per ray. Each potential intersection with a leaf bounding box is recorded in pairs of the form (`rayid`, `leafid`). In an optimal case all possible intersections would fit into this buffer. Given a number R of rays and a memory size M, N is proportional to M/R. The intersection candidates are then sorted by `leafid` in order to group all rays intersecting the same leaf, thereby reducing multiple accesses to the same leaf node. This approach may resemble [NFL07], however, specific details are not disclosed in their work and only simulated memory traffic statistics are provided.

For each `leafid` in the array, the leaf object is tessellated and the rays corresponding to the `leafid` are traced through the leaf object (see Section 6.1). In a parallel implementation each thread picks the next `leafid` as a task. Writing back intersection results to rays is either serialized by implementing a few locks

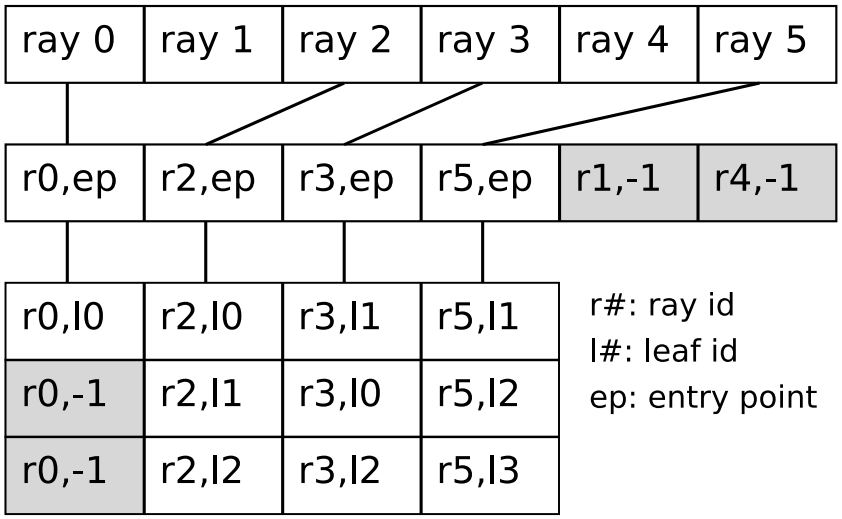


Figure 6.5.: The buffers used to sort the rays. Top: main buffer holding the actual ray structs, containing information such as hit distance, normal, ray origin, reciprocal direction. This buffer is not sorted and can be used to derive pixel indices. After one iteration of QBVH intersection, the second buffer is filled in parallel with entry points and all inactive rays are removed. Finally, all patches with a possible intersection on the way are stored in the third buffer. In this example, if another ray terminates, enough memory will become available for each of the three remaining rays to store one more intersection in order to tackle a larger depth complexity (four in this illustration).

for larger blocks of rays or, more efficiently, by writing the ray intersections to small buffers for each thread, which are synchronized at the end.

We conservatively determine for which rays the closest intersection has already been found by comparison against the following patch bounding box. These rays are terminated.

Once all rays are intersected, the top-level traversal is continued for all non-terminated rays using the last `leafid` along the ray direction as an entry point. These entry points have been stored explicitly per ray in an additional buffer (see Figure 6.5), because the original order of the (`rayid`, `leafid`) array is destroyed during reordering. Since traversal is ordered by ray direction, it is always clear which children to process next after the entry point, when stepping up in the hierarchy.

As the resulting number R' of non-terminated rays is typically significantly smaller than R, the next iteration can handle more potential intersections $N' \sim M/R'$, fully reusing the allocated buffers. This way, the process does not have to be repeated often, as the depth complexity of most scenes (the forest scene in Figure 6.1 has an overdraw of about 200) is reached quickly.

This scheme enables two more optimizations. First, in the presence of shaders, which require to access large memory blocks (such as measured BRDF data), many rays intersecting the respective surface will have an early out event at the same time and thus the memory does not have to be accessed several times. Second, to further reduce the need for repeated dicing over generations of rays, the early termination event can be used to shade a terminated block of rays, and spawn new ray directions, which can directly be intersected with the already diced originating patch and then be re-injected into the top-level traversal.

6.2.2. Tracing Rays in Groups and by Generation

Physically-based rendering requires a lot of rays to be traced. This number is typically too large to fit the required ray buffer into main memory. Also, at the beginning, not all rays are known. Some effects (such as soft shadows, ambient occlusion, reflections and so on) require several passes to be rendered, i.e. another generation, or wave, of rays to be shot.

There are several choices, which balance depth complexity, re-dicing, and memory requirements:

1. Re-inject rays as needed after an early termination event. This is done by replacing the terminated ray by a newly spawned one, instead of removing it from the buffer. This will always utilize the ray buffer well and use the (`rayid`, `leafid`) buffer for new rays rather than to tackle depth complexity.

2. Group rays by generation. This fixes the memory requirements for this wave of rays, but suffers from re-dicing for each pass.

3. Tile the screen. This can exploit some locality for first generation lens connection rays, but as rays quickly become divergent, re-dicing is as bad as in the previous variant.

Our current implementation uses the second approach. In general it is most efficient to trace as many rays as possible (i.e. fit into main memory) at a time.

For a simple path tracer, it is sufficient to update a single (spectral or RGB) path contribution value in the ray at each bounce. In the presence of complex reflection shaders with splitting into S sub-paths, each new ray needs to be assigned the correct weight $1/S$, but great care has to be taken not to exceed the buffer limits. A similar approach could be taken to implement ambient occlusion.

Bidirectional path tracing can be done by first tracing a wave of S paths from the sensor and T paths from the lights at the same time (resulting in $S+T$ rays at a time). After that, $S \cdot T$ connection rays have to be spawned with the respective weights, for example calculated using multiple importance sampling [VG95]. To bring the number of connection rays down to $S+T$ as well, Russian roulette based on these weights can be used.

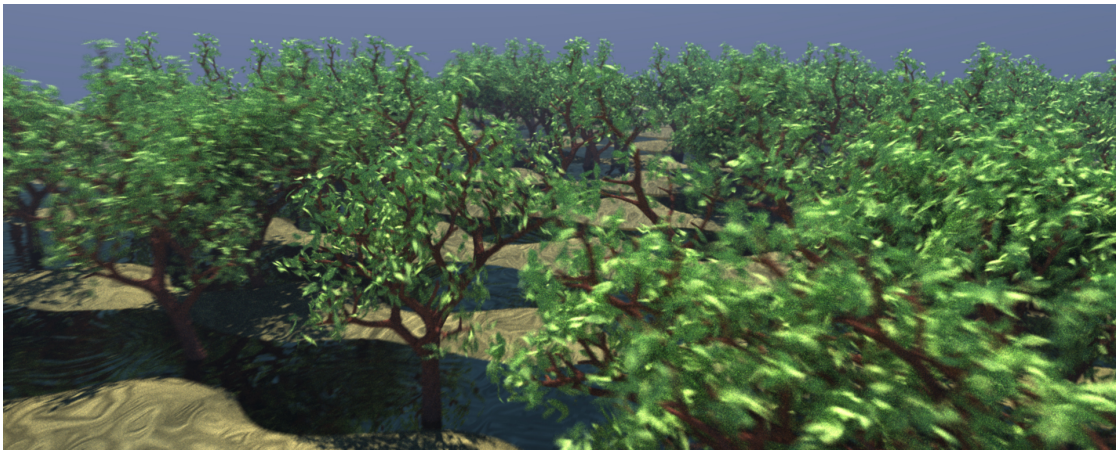


Figure 6.6.: The tree scene with exaggerated motion blur, which was used for the timings in Table 6.1

	dice [s]	bottom [s]	top [s]	shade [s]	sort [s]	#diced	total [s]
mb	211.06	86.90	27.70	25.20	34.84	1497633	179.0
no mb	104.10	56.50	24.76	25.36	28.82	1313136	133.0

Table 6.1.: Timings for the forest scene with exaggerated motion blur (seen in Figure 6.6) on a Core 2 Quad. Motion blur results in a slowdown of a factor of two for the dicing stage, and micro-polygon intersection is slightly slower. As more patches have to be diced in the presence of motion blur, also the sorting time is increased. All times are total times, except dice and bottom-level times, which are accumulated over all four cores.

6.3. Accelerating Motion Blur by Hierarchies Sharing Topology

Motion approximated by linear splines is standard in production (see e.g. [CFLB06, Sec.6.3] or [Grü08, Sec.2.4]). Given the instants $t_0 < t_1 < \cdots < t_n$ defining the time intervals $[t_i, t_{i+1})$, tracing a ray at time $t \in [t_i, t_{i+1})$ is accomplished by instancing two micro-polygon buffers, one at time t_i and one at time t_{i+1}. The actual bounding boxes and micro-polygons used during ray traversal then are determined by linear interpolation. We use this method for the bottom-level hierarchy.

Concerning the top-level hierarchy, the same principles can be applied. However, due to the cost to construct the hierarchy, we chose to use only one hierarchy based on bounding boxes conservatively covering the whole time interval $[t_0, t_n)$. See Table 6.1 for a comparison of render times with and without motion blur.

dicing	1000 trees	100 trees
cache 10	1,897,385	1,161,468
cache 100	943,669	772,491
cache 1000	825,808	606,371
reordering	482,405	354,534

Table 6.2.: This table shows how often patches have to be diced using a cache with 10, 100 and 1000 patches and our reordering method. Numbers are acquired using top-level traversal (i.e. independent of LOD) for the two forest scenes with motion blur at $1920 \times 768 \times 64$ rays. The reordering method only requires to store one diced patch per thread.

eye		bounce 1		bounce 2		bounce 3	
R	N	R	N	R	N	R	N
23592960	8	21589447	8	15585868	12	10052438	18
19009592	9	16419230	11	10886646	17	6707465	28
5840016	32	4549352	41	1529033	100	298166	100
1115814	100	406945	100	14093	100	82	100
1062	100	17	100	-	-	-	-
23592960	8	20674988	9	11985885	15	7311443	25
11479199	16	11145131	16	4617303	40	1775062	100
2110951	89	1241408	100	59364	100	204	100
7509	100	333	100	-	-	-	-

Table 6.3.: Tackling depth complexity: when rays are terminated early due to sorted BVH traversal, the memory can be used to store more patch intersections N for the remaining rays R. This is an example for the forest in Figure 6.8 (top table) and Figure 6.1 (bottom table) for the four waves of path tracing with next event estimation. N is clipped to 100 to avoid excessive fragmentation of memory for simple cases.

6.4. Results

We implemented a Monte Carlo global illumination renderer on top of our ray tracing architecture. We chose a simple path tracer with next event estimation, i.e. paths are traced from the eye and with a depth of three, additionally sampling the direct light contribution at each interaction point. Russian roulette is used to decide whether to sample the hemisphere or the light sources. This way, a maximum number of *width* $\times$ *height* $\times$ *samples per pixel* $\times$ *4* rays is traced per frame. While uncommon in movie production, this is a good demonstration of the generality of our method.

To achieve equivalent detail in a regular mesh-based renderer, the micro-polygons would have to be dumped to a triangle soup which would exceed the

scene	maximum	accessed
100 trees (Figure 6.1)	$108 \cdot 10^9$	$170 \cdot 10^6$
1000 trees (Figure 6.8)	$1,058 \cdot 10^9$	$317 \cdot 10^6$
dinosaur (Figure 6.9, left)	$59 \cdot 10^9$	$11 \cdot 10^6$
displaced dinosaur (Figure 6.9, right)	$59 \cdot 10^9$	$34 \cdot 10^6$

Table 6.4.: Impact of LOD and early ray termination due to occlusion: ratio of micro-polygons actually created to a constant LOD of $m = 10$ while path tracing.

capacities of these rendering systems. We therefore do not show comparisons with these. We tested the system on a variety of scenes ranging from trivial (Figure 6.2 left, equivalent to 260k triangles) over simple (Figure 6.2 right, equivalent to 16 million triangles), moderately complex (Figures 1,6.7 and 6.9) to massive scenes with detail equivalent to a mesh with over 1050 billion triangles (Figure 6.8), and scenes using reflection shaders accessing very large measured BRDF data sets (Figure 6.12). The trees are procedurally generated using L-systems with procedural displacement textures for the patches. The rest of the scenes is modeled in Bézier patches.

As all available rays are intersected with the scene at once before shading is started, complex reflection shaders benefit from this deferred shading architecture. If a second sorting step is inserted after all ray intersections have been found, even one single data load operation per bounce can be guaranteed. This is necessary, if not all used BRDF data sets fit into main memory at the time. In the case of our test scene (Figure 6.12), this was not necessary, but the time spent to sort the ray buffer can almost be neglected compared to the time spent in shading (less than 10 seconds compared to 263 seconds for shading $960 \times 640 \times 64$ rays).

We compare our ray reordering to a caching based system, similar to [PH96]. Our results in Table 6.2 indicate that for highly complex scenes caches need to be very large to be efficient. With our reordering method, they are not necessary, and the implementation becomes thus simpler than [PKGH97].

The table indicates that significantly more than 1000 cache lines, each representing one tessellated patch, are necessary in order to reduce the number of dicing steps to those achieved with our reordering. Besides the top-level hierarchy, our scheme requires a fixed memory footprint independent of geometry complexity. For path tracing at 1920x768x64 rays, our implementation needs 6120 MB for representing the ray buffers (68 bytes per ray: $3\times$ float position, $3\times$ float direction, $3\times$ float normal, $1\times$ float hit distance, $3\times$ float color, $3\times$ float path tracing weight RGB, $1\times$ int16 shader index and $1\times$ int16 additional flags to mark e.g. shadow rays). To represent the full tree of a single diced patch at LOD $m = 10$ (1024×1024 micro-quads with bounding boxes, color, texture coordinates and normals at two time instances and every level of detail), 120 MB are required. Our approach

requires a single buffer (corresponding to a single cache line), while the cache will grow linearly with the number of cache lines.

For an example how depth complexity is handled by the limited (`rayid,leafid`) buffer (see Section 6.2), see Table 6.3. It can be seen that even for scenes with very large overdraw, only few iterations are required. As some rays terminate, the buffer is quickly available for the remaining rays to store many more intersection candidates in the next iteration.

To get an impression of the impact of LOD, see Figure 6.11. The rendering times are dominated by dicing, so the graph shows only timings for top- and bottom-level traversal and shading. As expected, dicing and bottom-level tree construction seems to be linear and tracing bottom-level rays is about logarithmic. Top-level traversal does not change in this graph, as the top-level hierarchy is not affected by the LOD changes. Obviously a lot of time can be saved by reducing the number of micro-polygons per patch.

As illustrated in Table 6.4, our system is very efficient due to choosing the appropriate LOD and avoiding dicing for occluded patches altogether. The comparison here is carried out between actually created micro-polygons and the number of polygons in a triangle mesh with equivalent detail (maximum LOD $m = 10$). Note that this number is not overly large, this LOD is also chosen for some patches by the algorithm and becomes especially necessary for heavily displaced patches. The figures show that our system can robustly handle a vast amount of geometry, which surpasses the complexity demonstrated by the Razor system [SMD+06]. Also, we do not need to keep any diced micro-polygons, which avoids the problem of flushing on-demand geometry.

GPU Implementation and Frustum Tracing. In our setting, a GPU implementation did not increase performance compared to the quad core CPU version, because due to the different work package sizes the algorithm did not scale to hundreds or thousands of threads. Also the memory limitations on GPU hardware play a role here.

To cut down memory requirements for the ray buffer, we packed the rays into small frustums on the fly, following [Res07] similar to [GL10]. This removes a lot of memory accesses, since the frustum data is very compact as compared to the full ray buffer. Also intersection calculation can be performed very quickly, and frustums created on the fly, even for secondary rays. For simplistic scenes and very coherent rays, this works very well (see Figure 6.13, left). Unfortunately even for moderately complex geometry (see Figure 6.13, right), this approach is unable to cull the rays tightly. That is, the whole bundle of rays needs to be intersected with a lot of geometry because the frustum does not cut off space around the rays effectively enough. This is especially apparent in the presence of high depth complexity, and already shows up for primary rays.

6.5. Conclusion

We presented a ray tracing method, which is able to efficiently handle large amounts of data resulting from free form surface patches, details added by micro-polygon tessellation, and data intensive shaders. Expensive geometry and shaders (in terms of computation or memory access) are handled well due to reordering of computations, which results in great data locality. Parallelization is simple as all rays traverse one phase before the next one is started. We introduced only one additional sorting step on the ray buffer, which has negligible impact on rendering time. The core of our contribution is the two-level ray tracing system with reordering, which can be easily augmented by other advanced rendering techniques, as we have demonstrated for simple LOD selection and path tracing. The same two-level approach with reordering can be combined with general global illumination algorithms or even accelerate out-of-core rendering.

There are no restrictions imposed on ray tracing. However, there are some limitations when using the method in a rendering system. First, the shading language needs a mechanism to dispatch a ray and correctly account for its contribution by the time it finishes (see Section 6.2.2). If rendering is based on BRDFs, this is straightforward.

Second, the presented LOD assumes good importance sampling. That is, it assumes that if rays are diverging, the contributing radiance is low frequency. It will thus result in blocky shadows if sampling a small direct light source over the hemisphere instead of the geometry. If this is recognized by the rendering algorithm, it can be used as a feature to speed up low-frequency indirect illumination.

In the straight forward implementation, our bottom-level tessellation is bound to 2^m steps along one edge for an integral m. While the voxels always remain sub-pixel-sized for eye rays, popping artifacts for wildly displaced patches could be ameliorated by smaller steps between levels of detail. This can be achieved by using unbalanced quad-trees in our framework.

For artist-driven displacement shaders, the evaluation of interval arithmetic of the noise can be a restriction. By enforcing bandlimited noise functions, this calculus could be replaced by slightly larger worst-case bounding boxes.

In future work, the method can be complemented by specialized rendering algorithms which better exploit its strengths by reflecting the two-level nature, such as local high-frequency ambient occlusion inside a diced patch together with a far-field approximation for global illumination, similar to [KK04, AFO05]. Also stereo renderings could profit from coherence between rays for the two view positions. With the algorithm presented in this chapter, robust rendering of large, out-of-core or procedural micro-polygon data is possible with respect to rendering time.

Figure 6.7.: A tree rendered using our architecture. If it was dumped to a triangle mesh, it would consist of around 8.4 billion triangles (micro-polygons from 12k displaced Bézier patches). Due to the on-demand procedural geometry generation and the level of detail system, our system does not even create all these, and is able to render this scene with global illumination in a few minutes.

Figure 6.8.: Stress test: this forest consists of two million patches, which would need a triangle mesh equivalent of more than 1050 billion triangles if fully tessellated. Fully path traced with motion blur, the image renders at $1920 \times 768 \times 16$ samples in 2:24 minutes on a four core machine.

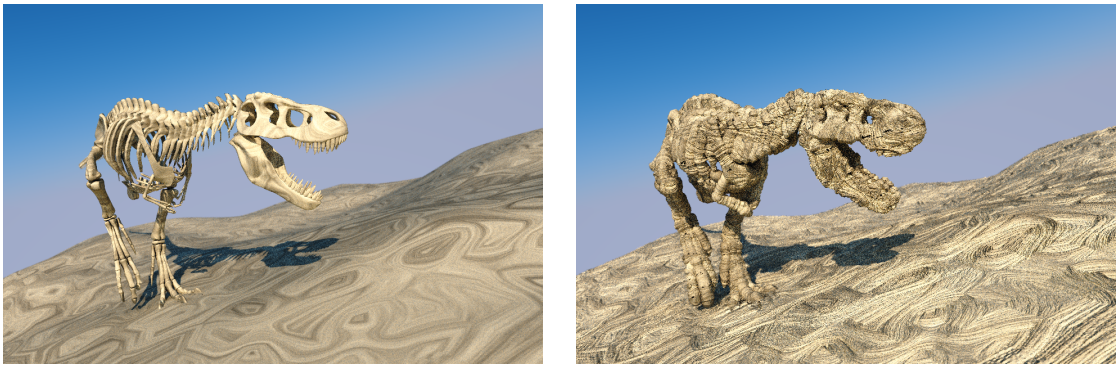


Figure 6.9.: A dinosaur (taken out of the natural history museum from the lighting challenge site and converted to Bézier patches using vertex normals), with 56k patches. On the left, around 11 million micro-polygons out of 59 billion potential have been created and rendered using path tracing in 18 seconds on a Core 2 Quad. On the right, a noisy displacement has been applied, resulting in about 34 million created micro-polygons and an increased render time of 37 seconds. Both images are rendered at $960 \times 640 \times 16$ samples. Some of the time is spent in the (intentionally) expensive procedural displacement texture.

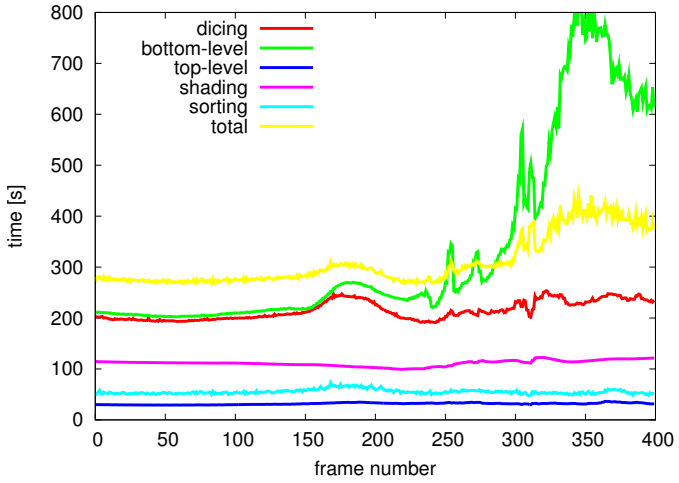


Figure 6.10.: Statistics for the video where Figure 6.1 are still frames from. These numbers have been generated on a Core 2 Quad, using 32 samples per pixel in 1920×768 resolution, path traced up to path depth of three (plus evaluation of direct light at each bounce). Again all times are total times, except dice and bottom-level times, which are accumulated over all four cores. Near the end, the camera closes up to a branch, so one patch has to be tessellated very finely (the maximum $m = 10$ is reached).

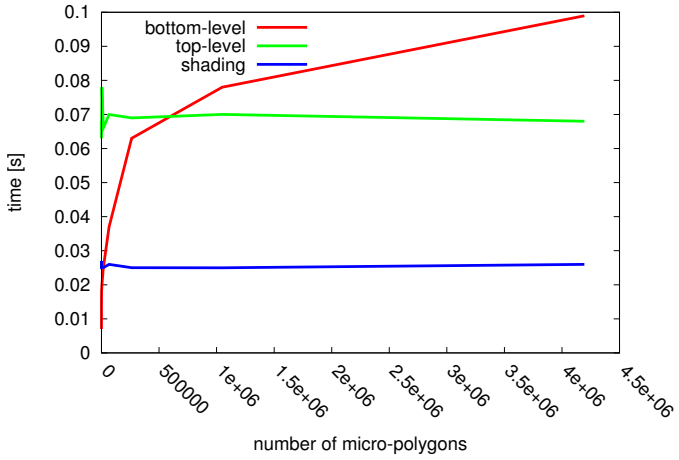


Figure 6.11.: Rendering time is determined by the programmable parts of the system, i.e. shading and surface patch tessellation (tessellation time is linear and takes over 100 seconds for $4 \cdot 10^6$ micro-polygons and is therefore omitted in the plot). Timings are obtained for the displaced teapot example (Figure 6.2).

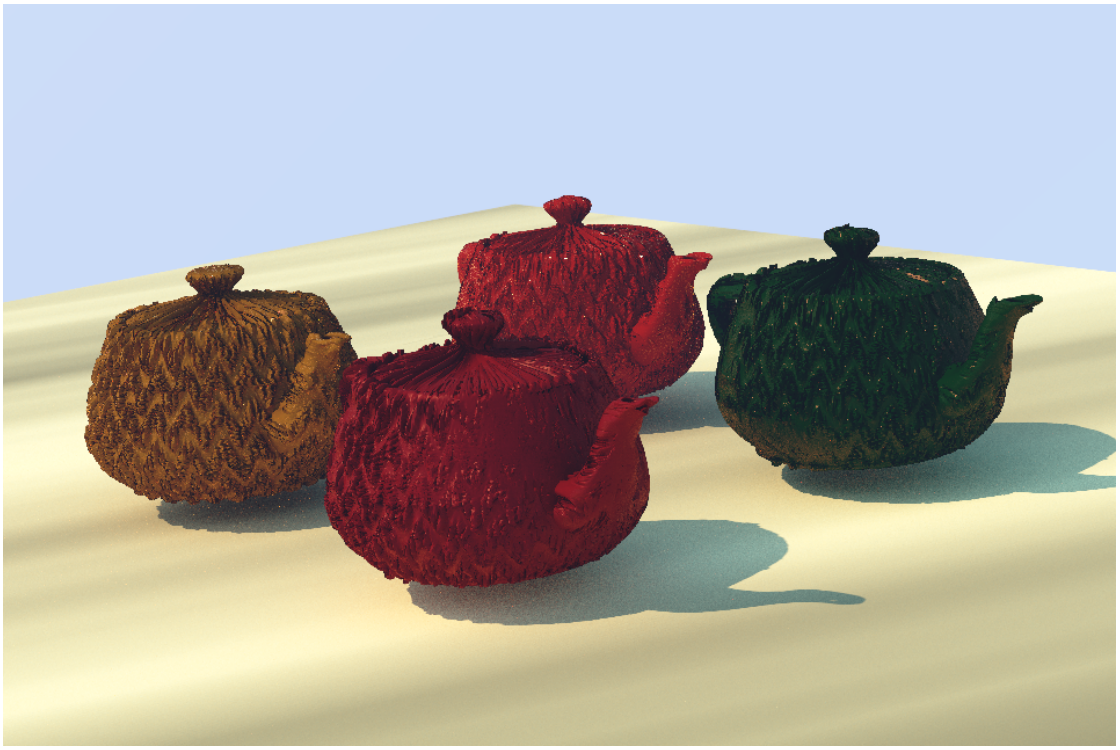


Figure 6.12.: Four teapots rendered with measured BRDF data. One BRDF data set alone is over 300 MB large, and each teapot consists of over 16 million triangles. Thanks to reordering of shading computations, it can be guaranteed that each BRDF data set is loaded only once per bounce.

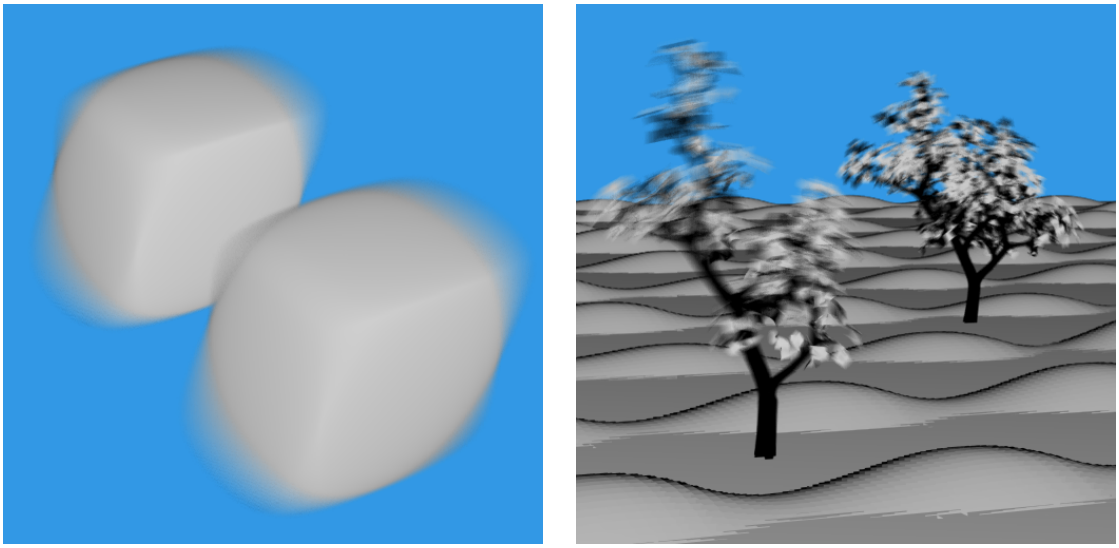


Figure 6.13.: Real-time Rayes test scenes. Left: two motion blurred cubes, right: two motion-blurred trees. For close to non-trivial geometry as on the right, the performance of the shaft culling method suffers severely, and does thus not provide a real improvement over the mono-ray approach of the original Rayes. The left image was ray traced with about 4 mrays/sec on a quad core computer, whereas for the trees the performance drops to only a few hundred thousand rays per second, because the coarse shafts cannot be terminated early.

"I know I know, Otto. You've been a very good assistant. And you brought in some good parts. That Manus was perfect. But what we really need now is the perfect. . . "

Nasum

7 Summary

In this work, we assembled tools and extended algorithms to implement a robust spectral rendering system. Building on top of this, the special demands of various fields have been considered, to make rendering more robust for arbitrary input data. In the next few sections, a brief summary of the most important results is given.

Spectral Rendering. We collocated a lot of well known facts and techniques and complemented them by some new bits, to create a color managed, spectral rendering system. Chapter 2 can be seen as detailed instruction on how to implement such a rendering system that just works.

Spectral Reflection. Faithful color rendition using spectrally measured BRDFs can be achieved using a color managed rendering pipeline. Chapter 3 shows this, and presents experiments with a novel BRDF lobe function, the Möbius lobe, which has some advantages over existing functions because it is defined exactly on the upper hemisphere.

Fluorescence. Spectral light transport can also be used to simulate complex light interaction such as fluorescence, as shown in Chapter 4. Fluorescence can also be rendered in a combined way with reflection, by measuring BRDFs in the

angular domain and the bispectral domain at the same time. We showed how to do this efficiently by upsampling sparse data to dense data sets using principal component analysis. The bispectral light transport simulation described here was also used to to support optimization of solar cells.

Precise Intersections. In Chapter 5, we investigated precise ray/triangle intersection tests targeting custom hardware, which produced more accurate results than the floating point versions. These algorithms have the potential to be implemented in less die area than full floating point computation. While precision is most interesting for scientific simulation, the associated area savings can be beneficial for graphics applications, too, and render the algorithm interesting for hardware producers.

Micro-Polygon Ray Tracing. To make massive, possibly out-of-core, micro-polygon ray tracing efficient, we investigated reordering of computations to make good use of memory once it is loaded, and construct specialized acceleration hierarchies in linear time (see Chapter 6).

7.1. Future Work

While we showed how to solve these problems individually, it remains to integrate all the different demands into a unified system. On this way, there are some challenging tasks, such as to implement a precise ray/triangle intersection test in off-the-shelf hardware, after optimizing it for the target architecture, die area, clock speed, and SIMD processing.

This would enable us to ray trace Reyes-style geometry with precisely intersected micro-triangles, which can replace the voxel rendering. Since these C_1 surfaces are somewhat more explored than voxels, it would be straightforward to fix the cracks between tessellated patches, include more sophisticated level of detail, adaptive tessellation, and more input primitives.

To find compact representations for data on the hemisphere or for scattered data interpolation it can be useful to have radial basis functions restricted to the hemisphere. This can be achieved using a weighted sum of multiple Möbius lobes.

Even though ray tracing is widely used throughout all rendering applications, finally and possibly most challenging, it remains to make accurate light simulation practical for all fields of rendering, by trading error for speed to create plausible images quickly enough for time-critical applications.

A
Source Code

A.1. Chirkov-Style Integer Ray/Triangle Intersection Test

This is the C source code which has been used to verify the theoretical precision analysis given in Section 5.2.6 in software. Note that temporary values have to be stored in `int64_t`, and the unneeded bits can only be stripped after the calculation. These wide values can be avoided in specialized hardware, which can be built to only calculate the necessary bits.

```
static inline void
triangle_shiftcross
(const int64_t *a, const int64_t *b, int *n, int *shift_n)
{
  // 23 bits without sign bit
  int64_t longn[3];
  crossproduct(a, b, longn);
  uint64_t dreggn = llabs(longn[0]) | llabs(longn[1]) | llabs(longn[2]);
  const int lz = __builtin_clzll(dreggn);
  *shift_n = 64 - 23 - (lz > (64-23) ? 64-23 : lz);
```

```
  for(int k=0;k<3;k++) n[k] = longn[k] >> (*shift_n);
}

static inline void
triangle_intersect_chirkov_int
(const triangle_int_t *t, const ray_t *ray, rayhit_t *hit,
 const uint32_t i)
{
  // (assume 23-bit fixed point, do precise 64 bit calculations)
  int64_t ac[3], ab[3], bc[3], a0[3], b0[3], c0[3];
  for(int k=0;k<3;k++)
  {
    ab[k] = (int64_t)t->v[1][k] - (int64_t)t->v[0][k];
    ac[k] = (int64_t)t->v[2][k] - (int64_t)t->v[0][k];
    bc[k] = (int64_t)t->v[2][k] - (int64_t)t->v[1][k];
    a0[k] = (int64_t)ray->ipos[k] - (int64_t)t->v[0][k];
    b0[k] = (int64_t)ray->ipos[k] - (int64_t)t->v[1][k];
    c0[k] = (int64_t)ray->ipos[k] - (int64_t)t->v[2][k];
  }
  // out facing normals for ccw tris:
  int a[3], b[3], c[3];
  int shift_a, shift_b, shift_c;
  triangle_shiftcross(ab, a0, a, &shift_a);
  triangle_shiftcross(bc, b0, b, &shift_b);
  triangle_shiftcross(c0, ac, c, &shift_c);

  const int64_t dotra = ((dotproductl(a, ray->idir)>>15));//14)+1)>>1;
  const int64_t dotrb = ((dotproductl(b, ray->idir)>>15));//14)+1)>>1;
  const int64_t dotrc = ((dotproductl(c, ray->idir)>>15));//14)+1)>>1;

  if(((dotra >= 0LL) && (dotrb >= 0LL) && (dotrc >= 0LL)) ||
     ((dotra <= 0LL) && (dotrb <= 0LL) && (dotrc <= 0LL)))
  {
    int n[3], shift_n;
    triangle_shiftcross(ab, ac, n, &shift_n);
    // d : 15 + 2 + 23 = 40 ( >> shift_n, << 15) (dir: 1 ^= 1<<15)
    const int64_t d = dotproduct(ray->idir, n);
    if(d == 0) return;
    // 23 + 23 + 2 + 15 = 63
    const int64_t dist = -(dotproduct(a0, n)<<15)/d;
    if(dist > 0 && dist <= hit->idist)
    {
      hit->idist = dist;
      hit->dist = tomant(dist);
      hit->tri = i;
      int64_t u = (dotproduct(a, ray->idir)<<(23 + shift_a - shift_n))/d;
      if(u < 0) u = 0;
      if(u > 0x7fffff) u = 0x7fffff;
      hit->u = tomant(u);
```

```
      int64_t v = (dotproduct(c, ray->idir)<<(23 + shift_c - shift_n))/d;
      if(v < 0) v = 0;
      if(v > 0x7fffff) v = 0x7fffff;
      hit->v = tomant(v);
    }
  }
}
```

A.2. Chirkov-Style Fixed Point Ray/Triangle Intersection Test

This section lists the VHDL source code of the Chirkov-style fixed point ray/triangle intersection test. It synthesized on a Virtex5 FPGA using the Xilinx toolchain at 390 MHz using 17 bits for the bounding box quantization, to exploit the 18×18 multiply element available on this piece of hardware. The pipeline is 8 clocks deep, but only tests for intersection and does not compute the distance. Optimized, precise division was given in [Han07].

```
------------------------------------------------------------
-- converts a wide temporary value to mini float format
-- only works for 2*18 bit in, 32 bit lzd
-- copyright (c) 2007 johannes hanika, hanatos@gmail.com
------------------------------------------------------------

library ieee;

use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.intrt_p.all;

package minifloat_p is

  component minifloat3 is
  generic (win : integer := 32; wshift : integer := 4);
  port
  (
    clk : in  std_logic;
    d0  : in  std_logic_vector(win-1 downto 0);
    d1  : in  std_logic_vector(win-1 downto 0);
    d2  : in  std_logic_vector(win-1 downto 0);
    s   : out std_logic_vector(wshift-1 downto 0);
    q0  : out std_logic_vector(frac_width-1 downto 0);
    q1  : out std_logic_vector(frac_width-1 downto 0);
    q2  : out std_logic_vector(frac_width-1 downto 0)
  );
  end component;

end package;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

-- detect leading zeroes
entity lzd16 is
port
(
  d : in  std_logic_vector(15 downto 0);
  q : out std_logic_vector(2 downto 0)
);
end entity;

architecture rtl of lzd16 is
  signal l2 : std_logic_vector(7 downto 0);
begin
main : process(d, l2)
  variable p : std_logic_vector(2 downto 0);
begin
  for i in 1 to 7 loop
    l2(i) <= d(2*i) or d(2*i+1);
  end loop;
  p(2) := l2(7) or l2(6) or l2(5) or l2(4);
  p(1) := ((l2(7) or l2(6)) and p(2)) or ((l2(3) or l2(2))
          and not p(2));
  case p(2 downto 1) is
  when "11" => p(0) := l2(7);
  when "10" => p(0) := l2(5);
  when "01" => p(0) := l2(3);
  when others => p(0) := l2(1);
  end case;
  q <= p;
end process;
end architecture;


library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use work.intrt_p.all;

entity shift18 is
port
(
  d : in  std_logic_vector(2*18-1 downto 0);
  s : in  std_logic_vector(2 downto 0);--(3 downto 0);
  q : out std_logic_vector(frac_width-1 downto 0)
);
end entity;
architecture rtl of shift18 is
begin
process(d, s)
begin
  case s is

  when "000"
    => q <= d(2*18-1) & d(21 downto (21+2)-frac_width);
  when "001"
    => q <= d(2*18-1) & d(23 downto (23+2)-frac_width);
  when "010"
    => q <= d(2*18-1) & d(25 downto (25+2)-frac_width);
  when "011"
    => q <= d(2*18-1) & d(27 downto (27+2)-frac_width);
  when "100"
    => q <= d(2*18-1) & d(29 downto (29+2)-frac_width);
  when "101"
    => q <= d(2*18-1) & d(31 downto (31+2)-frac_width);
  when "110"
```

```
    => q <= d(2*18-1) & d(33 downto (33+2)-frac_width);
  when others
    => q <= d(2*18-1) & d(35 downto (35+2)-frac_width);

  end case;
end process;
end architecture;


library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use work.intrt_p.all;
use work.minifloat_p.all;

entity minifloat3 is
generic (win : integer := 32; wshift : integer := 4);
port
(
  clk : in  std_logic;
  d0  : in  std_logic_vector(win-1 downto 0);
  d1  : in  std_logic_vector(win-1 downto 0);
  d2  : in  std_logic_vector(win-1 downto 0);
  s   : out std_logic_vector(wshift-1 downto 0);
  q0  : out std_logic_vector(frac_width-1 downto 0);
  q1  : out std_logic_vector(frac_width-1 downto 0);
  q2  : out std_logic_vector(frac_width-1 downto 0)
);
end entity;

architecture rtl of minifloat3 is

  component lzd16 is
  port
  (
    d : in  std_logic_vector(15 downto 0);
    q : out std_logic_vector(2 downto 0)
  );
  end component;
  component shift18 is
  port
  (
    d : in  std_logic_vector(2*18-1 downto 0);
    s : in  std_logic_vector(2 downto 0);
    q : out std_logic_vector(frac_width-1 downto 0)
  );
  end component;

  type reg_t is record
    d0, d01 : std_logic_vector(2*18-1 downto 0);
    d1, d11 : std_logic_vector(2*18-1 downto 0);
    d2, d21 : std_logic_vector(2*18-1 downto 0);
    s       : std_logic_vector(2 downto 0);
    lzd_in  : std_logic_vector(15 downto 0);
  end record;

  signal lzd_in  : std_logic_vector(15 downto 0);
  signal lzd_out : std_logic_vector(2 downto 0);
  signal dreggn  : std_logic_vector(win-1 downto 0);
  signal r, rin  : reg_t;

begin

  comb: process (d0, d1, d2, r)
  variable x, res : std_logic_vector(15 downto 0);
  variable v : reg_t;
  begin
    v := r;
    v.d0  := d0;
    v.d1  := d1;
    v.d2  := d2;
    v.d01 := r.d0;
    v.d11 := r.d1;
    v.d21 := r.d2;

    x := (others => d0(win-1));
    res := x xor d0(win-1 downto win-16);

    x := (others => d1(win-1));
    res := res or (x xor d1(win-1 downto win-16));

    x := (others => d2(win-1));
    res := res or (x xor d2(win-1 downto win-16));

    v.lzd_in := res;

    v.s := lzd_out;
    rin <= v;
  end process;

  lzd0   : lzd16   port map(r.lzd_in, lzd_out);
  shift0 : shift18 port map(r.d0, lzd_out, q0);
  shift1 : shift18 port map(r.d1, lzd_out, q1);
  shift2 : shift18 port map(r.d2, lzd_out, q2);
  s <= lzd_out;

  regs: process(clk)
  begin
    if rising_edge(clk) then
      r <= rin;
    end if;
  end process;

end architecture rtl;


------------------------------------------------------------
-- tests a ray against a triangle
-- unprojected chirkov-style, algebraically equivalent to
-- moeller trumbore/3d barycentric
-- takes 8 clock cycles @ 390 MHz
-- unfortunately also without calculating the distance.
-- copyright (c) 2007 johannes hanika, hanatos@gmail.com
------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
use work.intrt_p.all;

package tritest_chirkov_p is
  type tritest_in_t is record
    ray    : ray_t;
    tri    : tri_t;
    en     : std_logic;   -- enable
    trinum : std_logic_vector(15 downto 0);
    newray : std_logic;
  end record;

  type tritest_out_t is record
    far_x : std_logic_vector(aabb_width-1 downto 0); -- tfar
    far_y : std_logic_vector(aabb_width-1 downto 0);
    far_z : std_logic_vector(aabb_width-1 downto 0);

    hit    : std_logic;                      -- hit or not
    rdy    : std_logic;                      -- output valid
    trinum : std_logic_vector(15 downto 0);

    u      : std_logic_vector(aabb_width-1 downto 0);
    v      : std_logic_vector(aabb_width-1 downto 0);
    abco   : std_logic_vector(aabb_width-1 downto 0);
    d      : std_logic_vector(frac_width-1 downto 0);
    newray : std_logic;
  end record;

  component tritest is
  port
  (
    clk : in  std_logic;
    d   : in  tritest_in_t;
    q   : out tritest_out_t
  );
  end component;
end package;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
use work.intrt_p.all;
use work.tritest_chirkov_p.all;
use work.minifloat_p.all;

entity tritest is
port
(
```

```
  clk : in  std_logic;
  d   : in  tritest_in_t;
  q   : out tritest_out_t
);
end entity;

architecture rtl of tritest is
  -- need 7 regs:
  type reg0_t is record  -- sub
    ab_x, ab_y, ab_z, ac_x, ac_y, ac_z, bc_x, bc_y, bc_z,
    aO_x, aO_y, aO_z, bO_x, bO_y, bO_z, cO_x, cO_y, cO_z
      : std_logic_vector(aabb_width downto 0);
    omega_x : std_logic_vector(frac_width-1 downto 0);
    omega_y : std_logic_vector(frac_width-1 downto 0);
    omega_z : std_logic_vector(frac_width-1 downto 0);
    far_x   : std_logic_vector(aabb_width-1 downto 0);
    far_y   : std_logic_vector(aabb_width-1 downto 0);
    far_z   : std_logic_vector(aabb_width-1 downto 0);
    en      : std_logic;
    trinum  : std_logic_vector(15 downto 0);
    newray  : std_logic;
  end record;

  type reg1_t is record -- 2x mul
    lu_x_0, lu_x_1, lu_y_0, lu_y_1, lu_z_0, lu_z_1,
    lv_x_0, lv_x_1, lv_y_0, lv_y_1, lv_z_0, lv_z_1,
    lw_x_0, lw_x_1, lw_y_0, lw_y_1, lw_z_0, lw_z_1
      : std_logic_vector(2*aabb_width+1 downto 0);
    omega_x : std_logic_vector(frac_width-1 downto 0);
    omega_y : std_logic_vector(frac_width-1 downto 0);
    omega_z : std_logic_vector(frac_width-1 downto 0);
    far_x   : std_logic_vector(aabb_width-1 downto 0);
    far_y   : std_logic_vector(aabb_width-1 downto 0);
    far_z   : std_logic_vector(aabb_width-1 downto 0);
    en      : std_logic;
    trinum  : std_logic_vector(15 downto 0);
    newray  : std_logic;
  end record;

  type reg2_t is record -- 2x shift
    omega_x : std_logic_vector(frac_width-1 downto 0);
    omega_y : std_logic_vector(frac_width-1 downto 0);
    omega_z : std_logic_vector(frac_width-1 downto 0);
    far_x   : std_logic_vector(aabb_width-1 downto 0);
    far_y   : std_logic_vector(aabb_width-1 downto 0);
    far_z   : std_logic_vector(aabb_width-1 downto 0);
    en      : std_logic;
    trinum  : std_logic_vector(15 downto 0);
    newray  : std_logic;
  end record;

  type reg3_t is record -- 2x mul
    u_0, u_1, u_2, v_0, v_1, v_2, w_0, w_1, w_2
      : std_logic_vector(2*frac_width-1 downto 0);
    far_x   : std_logic_vector(aabb_width-1 downto 0);
    far_y   : std_logic_vector(aabb_width-1 downto 0);
    far_z   : std_logic_vector(aabb_width-1 downto 0);
    en      : std_logic;
    trinum  : std_logic_vector(15 downto 0);
    newray  : std_logic;
  end record;

  type reg4_t is record -- add
    u, v, w : std_logic_vector(2*frac_width-1 downto 0);
    far_x   : std_logic_vector(aabb_width-1 downto 0);
    far_y   : std_logic_vector(aabb_width-1 downto 0);
    far_z   : std_logic_vector(aabb_width-1 downto 0);
    en      : std_logic;
    trinum  : std_logic_vector(15 downto 0);
    newray  : std_logic;
  end record;

  type reg_t is record
    r0 : reg0_t;
    r1 : reg1_t;
    r2 : reg1_t;
    r3 : reg2_t;
    -- r4 : reg2_t;
    r5 : reg3_t;
    r6 : reg3_t;
    r7 : reg4_t;
  end record;

  -- this line needs more manual '0's to compile.
  signal r, rin : reg_t := (others=>'0');

  signal lu_x, lu_y, lu_z, lv_x, lv_y, lv_z, lw_x, lw_y,
         lw_z : std_logic_vector(2*aabb_width+1 downto 0);
  signal u_x, u_y, u_z, v_x, v_y, v_z, w_x, w_y, w_z
    : std_logic_vector(frac_width-1 downto 0);
  signal shift_u, shift_v, shift_w
    : std_logic_vector(2 downto 0);

begin

mf0 : minifloat3 generic map(2*aabb_width+2, 3)
  port map(clk, lu_x, lu_y, lu_z, shift_u, u_x, u_y, u_z);
mf1 : minifloat3 generic map(2*aabb_width+2, 3)
  port map(clk, lv_x, lv_y, lv_z, shift_v, v_x, v_y, v_z);
mf2 : minifloat3 generic map(2*aabb_width+2, 3)
  port map(clk, lw_x, lw_y, lw_z, shift_w, w_x, w_y, w_z);

process(r, d, u_x, u_y, u_z, v_x, v_y, v_z, w_x, w_y, w_z,
    shift_u, shift_v, shift_w)
  variable v : reg_t;
begin
  v := r;

  ------------------------------------------------------------
  -- reg0: sub, far, copy omega
  ------------------------------------------------------------

  -- sub
  v.r0.ab_x := ('0'&d.tri.bx) - ('0'&d.tri.ax);
  v.r0.ab_y := ('0'&d.tri.by) - ('0'&d.tri.ay);
  v.r0.ab_z := ('0'&d.tri.bz) - ('0'&d.tri.az);
  v.r0.ac_x := ('0'&d.tri.cx) - ('0'&d.tri.ax);
  v.r0.ac_y := ('0'&d.tri.cy) - ('0'&d.tri.ay);
  v.r0.ac_z := ('0'&d.tri.cz) - ('0'&d.tri.az);
  v.r0.bc_x := ('0'&d.tri.cx) - ('0'&d.tri.bx);
  v.r0.bc_y := ('0'&d.tri.cy) - ('0'&d.tri.by);
  v.r0.bc_z := ('0'&d.tri.cz) - ('0'&d.tri.bz);
  v.r0.aO_x := ('0'&d.ray.O_x) - ('0'&d.tri.ax);
  v.r0.aO_y := ('0'&d.ray.O_y) - ('0'&d.tri.ay);
  v.r0.aO_z := ('0'&d.ray.O_z) - ('0'&d.tri.az);
  v.r0.bO_x := ('0'&d.ray.O_x) - ('0'&d.tri.bx);
  v.r0.bO_y := ('0'&d.ray.O_y) - ('0'&d.tri.by);
  v.r0.bO_z := ('0'&d.ray.O_z) - ('0'&d.tri.bz);
  v.r0.cO_x := ('0'&d.ray.O_x) - ('0'&d.tri.cx);
  v.r0.cO_y := ('0'&d.ray.O_y) - ('0'&d.tri.cy);
  v.r0.cO_z := ('0'&d.ray.O_z) - ('0'&d.tri.cz);
  -- copy omega
  v.r0.omega_x := d.ray.omega_x;
  v.r0.omega_y := d.ray.omega_y;
  v.r0.omega_z := d.ray.omega_z;
  v.r0.en := d.en;
  v.r0.trinum := d.trinum;
  v.r0.newray := d.newray;
  -- find far values
  if d.tri.ax > d.tri.bx then
    if d.tri.ax > d.tri.cx then
      v.r0.far_x := d.tri.ax;
    else
      v.r0.far_x := d.tri.cx;
    end if;
  else
    if d.tri.bx > d.tri.cx then
      v.r0.far_x := d.tri.bx;
    else
      v.r0.far_x := d.tri.cx;
    end if;
  end if;
  if d.tri.ay > d.tri.by then
    if d.tri.ay > d.tri.cy then
      v.r0.far_y := d.tri.ay;
    else
      v.r0.far_y := d.tri.cy;
    end if;
  else
    if d.tri.by > d.tri.cy then
      v.r0.far_y := d.tri.by;
    else
      v.r0.far_y := d.tri.cy;
    end if;
  end if;
  if d.tri.az > d.tri.bz then
```

```
  if d.tri.az > d.tri.cz then
    v.r0.far_z := d.tri.az;
  else
    v.r0.far_z := d.tri.cz;
  end if;
else
  if d.tri.bz > d.tri.cz then
    v.r0.far_z := d.tri.bz;
  else
    v.r0.far_z := d.tri.cz;
  end if;
end if;

------------------------------------------------------------
-- reg1: pipelined mul stage1, copy far, copy omega
------------------------------------------------------------

v.r1.lu_x_0 := r.r0.ab_y * r.r0.aO_z;
v.r1.lu_x_1 := r.r0.ab_z * r.r0.aO_y;
v.r1.lu_y_0 := r.r0.ab_z * r.r0.aO_x;
v.r1.lu_y_1 := r.r0.ab_x * r.r0.aO_z;
v.r1.lu_z_0 := r.r0.ab_x * r.r0.aO_y;
v.r1.lu_z_1 := r.r0.ab_y * r.r0.aO_x;

v.r1.lv_x_0 := r.r0.bc_y * r.r0.bO_z;
v.r1.lv_x_1 := r.r0.bc_z * r.r0.bO_y;
v.r1.lv_y_0 := r.r0.bc_z * r.r0.bO_x;
v.r1.lv_y_1 := r.r0.bc_x * r.r0.bO_z;
v.r1.lv_z_0 := r.r0.bc_x * r.r0.bO_y;
v.r1.lv_z_1 := r.r0.bc_y * r.r0.bO_x;

v.r1.lw_x_0 := r.r0.cO_y * r.r0.ac_z;
v.r1.lw_x_1 := r.r0.cO_z * r.r0.ac_y;
v.r1.lw_y_0 := r.r0.cO_z * r.r0.ac_x;
v.r1.lw_y_1 := r.r0.cO_x * r.r0.ac_z;
v.r1.lw_z_0 := r.r0.cO_x * r.r0.ac_y;
v.r1.lw_z_1 := r.r0.cO_y * r.r0.ac_x;

-- copy omega
v.r1.omega_x := r.r0.omega_x;
v.r1.omega_y := r.r0.omega_y;
v.r1.omega_z := r.r0.omega_z;
-- pass on max for early out in nbvh
v.r1.far_x := r.r0.far_x;
v.r1.far_y := r.r0.far_y;
v.r1.far_z := r.r0.far_z;
v.r1.en := r.r0.en;
v.r1.trinum := r.r0.trinum;
v.r1.newray := r.r0.newray;

------------------------------------------------------------
-- reg2: pipelined mul stage2, copy far, copy omega
------------------------------------------------------------

v.r2.lu_x_0 := r.r1.lu_x_0;
v.r2.lu_x_1 := r.r1.lu_x_1;
v.r2.lu_y_0 := r.r1.lu_y_0;
v.r2.lu_y_1 := r.r1.lu_y_1;
v.r2.lu_z_0 := r.r1.lu_z_0;
v.r2.lu_z_1 := r.r1.lu_z_1;

v.r2.lv_x_0 := r.r1.lv_x_0;
v.r2.lv_x_1 := r.r1.lv_x_1;
v.r2.lv_y_0 := r.r1.lv_y_0;
v.r2.lv_y_1 := r.r1.lv_y_1;
v.r2.lv_z_0 := r.r1.lv_z_0;
v.r2.lv_z_1 := r.r1.lv_z_1;

v.r2.lw_x_0 := r.r1.lw_x_0;
v.r2.lw_x_1 := r.r1.lw_x_1;
v.r2.lw_y_0 := r.r1.lw_y_0;
v.r2.lw_y_1 := r.r1.lw_y_1;
v.r2.lw_z_0 := r.r1.lw_z_0;
v.r2.lw_z_1 := r.r1.lw_z_1;

-- copy omega
v.r2.omega_x := r.r1.omega_x;
v.r2.omega_y := r.r1.omega_y;
v.r2.omega_z := r.r1.omega_z;
-- pass on max for early out in nbvh
v.r2.far_x := r.r1.far_x;
v.r2.far_y := r.r1.far_y;
v.r2.far_z := r.r1.far_z;
v.r2.en := r.r1.en;
v.r2.trinum := r.r1.trinum;
v.r2.newray := r.r1.newray;

-- 1x reg parallel to r2 are in minifloat
lu_x <= r.r1.lu_x_0 - r.r1.lu_x_1;
lu_y <= r.r1.lu_y_0 - r.r1.lu_y_1;
lu_z <= r.r1.lu_z_0 - r.r1.lu_z_1;
lv_x <= r.r1.lv_x_0 - r.r1.lv_x_1;
lv_y <= r.r1.lv_y_0 - r.r1.lv_y_1;
lv_z <= r.r1.lv_z_0 - r.r1.lv_z_1;
lw_x <= r.r1.lw_x_0 - r.r1.lw_x_1;
lw_y <= r.r1.lw_y_0 - r.r1.lw_y_1;
lw_z <= r.r1.lw_z_0 - r.r1.lw_z_1;

------------------------------------------------------------
-- reg3: shift mul result (done in mf0-2), copy far, omega
------------------------------------------------------------
-- v.r3.omega_x := r.r2.omega_x;
-- v.r3.omega_y := r.r2.omega_y;
-- v.r3.omega_z := r.r2.omega_z;
-- v.r3.far_x   := r.r2.far_x;
-- v.r3.far_y   := r.r2.far_y;
-- v.r3.far_z   := r.r2.far_z;
-- v.r3.en      := r.r2.en;
-- v.r3.trinum  := r.r2.trinum;

------------------------------------------------------------
-- reg4: shift mul result (done in mf0-2), copy far, omega
------------------------------------------------------------
-- v.r4.omega_x := r.r3.omega_x;
-- v.r4.omega_y := r.r3.omega_y;
-- v.r4.omega_z := r.r3.omega_z;
-- v.r4.far_x   := r.r3.far_x;
-- v.r4.far_y   := r.r3.far_y;
-- v.r4.far_z   := r.r3.far_z;
-- v.r4.en      := r.r3.en;
-- v.r4.trinum  := r.r3.trinum;

------------------------------------------------------------
-- reg5: eval dot products, copy far
------------------------------------------------------------
v.r5.far_x := r.r2.far_x;
v.r5.far_y := r.r2.far_y;
v.r5.far_z := r.r2.far_z;
v.r5.u_0 := u_x * r.r2.omega_x;
v.r5.u_1 := u_y * r.r2.omega_y;
v.r5.u_2 := u_z * r.r2.omega_z;
v.r5.v_0 := v_x * r.r2.omega_x;
v.r5.v_1 := v_y * r.r2.omega_y;
v.r5.v_2 := v_z * r.r2.omega_z;
v.r5.w_0 := w_x * r.r2.omega_x;
v.r5.w_1 := w_y * r.r2.omega_y;
v.r5.w_2 := w_z * r.r2.omega_z;
v.r5.en  := r.r2.en;
v.r5.trinum := r.r2.trinum;
v.r5.newray := r.r2.newray;

------------------------------------------------------------
-- reg6: pipelined dot products, copy far
------------------------------------------------------------
v.r6.far_x := r.r5.far_x;
v.r6.far_y := r.r5.far_y;
v.r6.far_z := r.r5.far_z;
v.r6.u_0 := r.r5.u_0;
v.r6.u_1 := r.r5.u_1;
v.r6.u_2 := r.r5.u_2;
v.r6.v_0 := r.r5.v_0;
v.r6.v_1 := r.r5.v_1;
v.r6.v_2 := r.r5.v_2;
v.r6.w_0 := r.r5.w_0;
v.r6.w_1 := r.r5.w_1;
v.r6.w_2 := r.r5.w_2;
v.r6.en  := r.r5.en;
v.r6.trinum := r.r5.trinum;
v.r6.newray := r.r5.newray;

------------------------------------------------------------
-- reg7: pipelined dot products, copy far
------------------------------------------------------------
v.r7.far_x := r.r6.far_x;
v.r7.far_y := r.r6.far_y;
```

```
v.r7.far_z := r.r6.far_z;
v.r7.u := r.r6.u_0 + r.r6.u_1 + r.r6.u_2;
v.r7.v := r.r6.v_0 + r.r6.v_1 + r.r6.v_2;
v.r7.w := r.r6.w_0 + r.r6.w_1 + r.r6.w_2;
v.r7.en := r.r6.en;
v.r7.trinum := r.r6.trinum;
v.r7.newray := r.r6.newray;

------------------------------------------------------------
-- output
------------------------------------------------------------

q.far_x <= r.r7.far_x;
q.far_y <= r.r7.far_y;
q.far_z <= r.r7.far_z;

q.rdy    <= r.r7.en;
q.trinum <= r.r7.trinum;
q.newray <= r.r7.newray;

rin <= v;
end process;

process(clk)
begin
  if rising_edge(clk) then
    r <= rin;
  end if;
end process;

end architecture rtl;
```

Bibliography

[AFO05] Okan Arikan, David Forsyth, and James O'Brien. Fast and detailed approximate global illumination by irradiance decomposition. *ACM Transactions on Graphics (Proc. SIGGRAPH 2005)*, pages 1108–1114, 2005. 120

[AK10] Timo Aila and Tero Karras. Architecture considerations for tracing incoherent rays. In *Proc. High-Performance Graphics 2010*, pages 113–122, 2010. 33

[AL09] Timo Aila and Samuli Laine. Understanding the efficiency of ray traversal on GPUs. In *Proc. High-Performance Graphics 2009*, pages 145–149, 2009. 33

[AM01] Elli Angelopoulou and Rana Molana. Multispectral skin color modeling. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 635–642, 2001. 59

[Ame] American Society for Testing and Materials. Reference solar spectral irradiance: Air mass 1.5. http://rredc.nrel.gov/solar/spectra/am1.5/. 18, 49, 62

[AMH02] Tomas Akenine-Möller and Eric Haines. *Real-Time Rendering, 2nd ed.* A.K. Peters Ltd., 2002. 92

[AS00] Michael Ashikhmin and Peter Shirley. An anisotropic Phong BRDF model. *Journal of Graphics Tools*, 5(2):25–32, 2000. 36, 37, 39

[Bad90] Didier Badouel. An efficient ray-polygon intersection. In A. S. Glassner, editor, *Graphics Gems*, pages 390–393. Academic Press Professional, 1990. 89, 91

[BBdF96] A. Baccini, Ph. Besse, and A. de Falguerolles. An L1-norm PCA and a heuristic approach. In *Ordinal and Symbolic Data Analysis*, pages 359–368. Springer, 1996. 75

[BBLW07] Carsten Benthin, Solomon Boulos, Dylan Lacewell, and Ingo Wald. Packet-based ray tracing of Catmull-Clark subdivision surfaces. Technical Report UUSCI-2007-011, SCI Institute, University of Utah, 2007. 108

[BBS+09] Brian Budge, Tony Bernardin, Jeff Stuart, Shubhabrata Sengupta, Kenneth Joy, and John Owens. Out-of-core data management for path tracing on hybrid resources. In *Computer Graphics Forum (Proc. of Eurographics 2009)*, pages 385–396, 2009. 107

[Ben04] Carsten Benthin. *Realtime Ray Tracing on Current CPU Architectures*. PhD thesis, Saarland University, 2004. 33, 89

[BHD+08] Marion Bendig, Johannes Hanika, Holger Dammertz, Jan Christoph Goldschmidt, Marius Peters, and Michael Weber. Simulation of fluorescent concentrators. In *Proc. 2008 IEEE/EG Symposium on Interactive Ray Tracing*, pages 93–98, 2008. 14, 60

[Bie00] Ludwig Bieberbach. *Conformal Mapping*. AMS Chelsea Publishing, 2000. 40

[BMW+09] Jiri Bittner, Oliver Mattausch, Peter Wonka, Vlastimil Havran, and Michael Wimmer. Adaptive global visibility sampling. *ACM Transactions on Graphics (Proc. SIGGRAPH 2009)*, pages 94:1–94:10, 2009. 84

[BSBvR06] Antonius Burgers, Lenneke Slooff, Andreas Buchtemann, and John van Roosmalen. Performance of Single Layer Luminescent Concentrators with Multiple Dyes. In *Conference Record of the 2006 IEEE 4th World Conference on Photovoltaic Energy Conversion*, pages 198–201, 2006. 58

[BSKvR05] A.R. Burgers, L.H. Slooff, R. Kinderman, and J.A.M. van Roosmalen. Modelling of Luminescent Concentrators by Ray-Tracing. In *Proceedings of the 20th European Photovoltaic Solar Energy Conference and Exhibition*, 2005. 58

[BWSF06] Carsten Benthin, Ingo Wald, Michael Scherbaum, and Heiko Friedrich. Ray Tracing on the CELL Processor. In *Proc. 2006 IEEE Symposium on Interactive Ray Tracing*, pages 25–23, 2006. 33

[CAU83] M. Carrascosa, F. Agullo-Lopez, and S. Unamuno. Monte Carlo simulation of the performance of PMMA luminescent solar collectors. *Applied Optics*, 22:3236–3241, 1983. 58

[CC78] Edwin Catmull and Jim Clark. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer Aided Design*, 10(6):350–355, 1978. 85

[CCC87] Robert Cook, Loren Carpenter, and Edwin Catmull. The Reyes image rendering architecture. *Computer Graphics (Proc. SIGGRAPH '87)*, pages 95–102, 1987. 84, 105, 106

[CE05] David Cline and Parris Egbert. A practical introduction to Metropolis light transport. Technical report, Brigham Young University, 2005. 29

[CFLB06] Per Christensen, Julian Fong, David Laur, and Dana Batali. Ray tracing for the movie 'Cars'. In *Proc. 2006 IEEE Symposium on Interactive Ray Tracing*, pages 73–78, 2006. 106, 112, 116

[CHCH06] Nathan Carr, Jared Hoberock, Keenan Crane, and John Hart. Fast GPU ray tracing of dynamic meshes using geometry images. In *GI '06: Proceedings of the 2006 conference on Graphics interface*, pages 203–209, 2006. 108, 110

[Chi05] Nick Chirkov. Fast 3d line segment–triangle intersection test. *journal of graphics, gpu, and game tools*, 10(3):13–18, 2005. 89, 91, 92

[CHPR07] Robert Cook, John Halstead, Maxwell Planck, and David Ryu. Stochastic simplification of aggregate detail. *ACM Transactions on Graphics*, 26(3):79, 2007. 109

[CJAMJ05] Petrik Clarberg, Wojciech Jarosz, Tomas Akenine-Möller, and Henrik Wann Jensen. Wavelet importance sampling: efficiently evaluating products of complex functions. *ACM Transactions on Graphics*, 24(3):1166–1175, 2005. 33

[CMP96] Wonjoon Cho, Takashi Maekawa, and Nicholas Patrikalakis. Topologically reliable approximation of composite Bézier curves. *Computer Aided Geometric Design*, 13(6):497–520, 1996. 94

[CS08] John Carmack and Ryan Shrout. John Carmack on id Tech 6, Ray Tracing, Consoles, Physics and more. PC Perspective interview March 12, 2008. 96

[CT81] Robert Cook and Kenneth Torrance. A reflectance model for computer graphics. *Computer Graphics (Proc. SIGGRAPH '81)*, pages 307–316, 1981. 36, 58

[CUR96] CUReT. Columbia Utrecht Texture Database. Web-Page. http://www1.cs.columbia.edu/CAVE/software/curet/index.php, 1996. 58

[DBB06] Philip Dutré, Kavita Bala, and Philippe Bekaert. *Advanced Global Illumination*. AK Peters, Ltd., 2006. 26

[DCWP02] Kate Devlin, Alan Chalmers, Alexander Wilkie, and Werner Purgathofer. Star: Tone reproduction and physically based spectral rendering. In *State of the Art Reports, Eurographics 2002*, pages 101–123, 2002. 21

[DDK08] Sabrina Dammertz, Holger Dammertz, and Alexander Keller. Efficient search for low-dimensional rank-1 lattices with applications in graphics. In *Proc. Monte Carlo and Quasi-Monte Carlo Methods 2006*, pages 217–236. Springer, 2008. 33

[Deb98] Paul Debevec. Rendering synthetic objects into real scenes: bridging traditional and image-based graphics with global illumination and high dynamic range photography. *Proc. of SIGGRAPH '98*, pages 189–198, 1998. 18

[dFS04] Luiz de Figueiredo and Jorge Stolfi. Affine arithmetic: Concepts and applications. *Numerical Algorithms*, 37(1–4):147–158, 2004. 87

[DH09] Holger Dammertz and Johannes Hanika. Plane sampling for light paths from the environment map. *journal of graphics, gpu and game tools*, 14(2):25–31, 2009. 13, 33

[DHK08] Holger Dammertz, Johannes Hanika, and Alexander Keller. Shallow bounding volume hierarchies for fast SIMD ray tracing of incoherent rays. In *Computer Graphics Forum (Proc. 19th Eurographics Symposium on Rendering)*, pages 1225–1234, 2008. 14, 27, 32, 104, 106, 113

[DHKL09] Holger Dammertz, Johannes Hanika, Alexander Keller, and Hendrik Lensch. A hierarchical automatic stopping condition for Monte Carlo global illumination. In *Proc. of the WSCG 2009*, pages 159–164, 2009. 13, 33

[DK06] Holger Dammertz and Alexander Keller. Improving ray tracing precision by world space intersection computation. In *Proc. 2006 IEEE Symposium on Interactive Ray Tracing*, pages 25–32, 2006. 94, 111, 112

[DK08] Holger Dammertz and Alexander Keller. Edge volume heuristic - robust triangle subdivision for improved BVH performance. In *Proc. 2008 IEEE/EG Symposium on Interactive Ray Tracing*, pages 155–158, 2008. 33

[Don54] R. Donaldson. Spectrophotometry of fluorescent pigments. *British Journal of Applied Physics*, 5(6):210–214, 1954. 58, 66

[Dre07] Ulrich Drepper. What every programmer should know about memory. LWN.net, 2007. 84

[DSHL10] Holger Dammertz, Daniel Sewtz, Johannes Hanika, and Hendrik Lensch. Edge-avoiding à-trous wavelet transform for fast global illumination filtering. In *Proc. High Performance Graphics 2010*, pages 67–75, 2010. 13, 33

[Dut03] Philip Dutré. Global illumination compendium – the concise guide to global illumination algorithms. `http://people.cs.kuleuven.be/~philip.dutre/GI/`, 2003. 18, 19, 21

[DWT+10] Yue Dong, Jiaping Wang, Xin Tong, John Snyder, Yanxiang Lan, Moshe Ben-Ezra, and Baining Guo. Manifold bootstrapping for SVBRDF capture. *ACM Transactions on Graphics (Proc. SIGGRAPH 2010)*, pages 1–10, 2010. 37

[EBJ+06] Dave Edwards, Solomon Boulos, Jared Johnson, Peter Shirley, Michael Ashikhmin, Michael Stark, and Chris Wyman. The halfway vector disk for BRDF modeling. *ACM Transactions on Graphics*, 25(1):1–18, 2006. 36, 37, 40

[EG07] Manfred Ernst and Gunther Greiner. Early split clipping for bounding volume hierarchies. In *Proc. 2007 IEEE/EG Symposium on Interactive Ray Tracing*, pages 73–78, 2007. 33

[EG08] Manfred Ernst and Gunther Greiner. Multi bounding volume hierarchies. In *Proc. 2008 IEEE/EG Symposium on Interactive Ray Tracing*, pages 35–40, 2008. 33, 104, 113

[Ein05] Albert Einstein. Über einen die Erzeugung und Verwandlung des Lichtes betreffenden heuristischen Gesichtspunkt. *Annalen der Physik*, 17:132–148, 1905. 19

[EL10] Christian Eisenacher and Charles Loop. Data-parallel micropolygon rasterization. In *Eurographics 2010 short papers*, pages 53–56, 2010. 108

[Eri05] Christer Ericson. *Real-Time Collision Detection*. Morgan Kaufmann, 2005. 99

[Eri07] Christer Ericson. Plücker coordinates considered harmful! Real-Time Collision Detection blog, 2007. 89, 91

[Erm75] Sergej Mikhailovich Ermakow. *Die Monte-Carlo-Methode und verwandte Fragen*. VEB Deutscher Verlag der Wissenschaften, 1975. 23

[FBH98] Hugh Fairman, Michael Brill, and Henry Hemmendinger. How the CIE 1931 color-matching functions were derived from wright-guild data. *Color Research and Application*, 22(1):11–23, 1998. 20

[FFB+09] Matthew Fisher, Kayvon Fatahalian, Solomon Boulos, Kurt Akeley, William Mark, and Pat Hanrahan. DiagSplit: Parallel, crack-free, adaptive tessellation for micropolygon rendering. *SIGGRAPH Asia '09: ACM SIGGRAPH Asia 2009 papers*, pages 1–10, 2009. 112

[GD03] Philippe Guigue and Olivier Devillers. Fast and robust triangle-triangle overlap test using orientation predicates. *journal of graphics, gpu, and game tools*, 8(1):25–42, 2003. 89, 91

[GGGW06] Jan Christoph Goldschmidt, Stefan Glunz, Andreas Gombert, and Gerhard Willeke. Advanced fluorescent concentrators. In *Proceedings of the 21st European Photovoltaic Solar Energy Conference*, 2006. 60

[GHSK08] Leonhard Grünschloß, Johannes Hanika, Ronnie Schwede, and Alexander Keller. (t, m, s)-nets with maximized minimum distance. In *Proc. Monte Carlo and Quasi-Monte Carlo Methods 2006*, pages 397–412. Springer, 2008. 14, 33

[GL10] Kirill Garanzha and Charles Loop. Fast ray sorting and breadth-first packet traversal for GPU ray tracing. In *Computer Graphics Forum (Proc. of Eurographics 2010)*, pages 289–298, 2010. 119

[Gla89] Andrew Glassner. *An Introduction to Ray Tracing*. Academic Press, 1989. 27, 84

[Gla94] Andrew Glassner. A model for fluorescence and phosphorescence. In *Proceedings of the 5th Eurographics Workshop on Rendering*, pages 57–68, 1994. 58, 66

[GMD10] David Geisler-Moroder and Arne Dür. A new Ward BRDF model with bounded albedo. In *Proceedings of the Eurographics Symposium on Rendering*, pages 1391–1398, 2010. 36

[Gol91] David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Comput. Surv.*, 23(1):5–48, 1991. 86

[Gri09] Larry Gritz. Production perspectives on high performance graphics. Keynote Talk at the High Performance Graphics conference, 2009. 105

[Grü08] Leonhard Grünschloß. Motion blur. Master's thesis, Ulm University, 2008. 116

[GS87] Jeffrey Goldsmith and John Salmon. Automatic creation of object hierarchies for ray tracing. *IEEE Computer Graphics & Applications*, 7(5):14–20, 1987. 111

[GT94] Dietrich Gundlach and Heinz Terstiege. Problems in measurement of fluorescent materials. *Color Research & Application*, 19(6):427–436, 1994. 59

[Had00] Mirza Hadzic. Letters: Graphics algorithms. *Dr. Dobb's Journal*, 25(11), November 2000. 95

[Han03] Johannes Hanika. Real-Time Raytracing Voxel Engine, 2003. `http://rearview.sourceforge.net/`. 96

[Han07] Johannes Hanika. Fixed point hardware ray tracing. Master's thesis, Ulm University, 2007. 88, 131

[Han09] Johannes Hanika. Digital Photography Workflow Software, 2009. `http://darktable.sourceforge.net/`. 21

[Has70] W. Keith Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, 1970. 28, 46

[Hav01] Vlastimil Havran. *Heuristic Ray Shooting Algorithms.* PhD thesis, Czech Technical University, 2001. 84

[HDC07] Roger Hersch, Philipp Donzé, and Sylvain Chosson. Color images visible under UV light. *ACM Transactions on Graphics (Proc. SIGGRAPH 2007)*, page 75, 2007. 59

[Hec90] Paul Heckbert. Adaptive radiosity textures for bidirectional ray tracing. *Computer Graphics (Proc. SIGGRAPH '90)*, pages 145–154, 1990. 27, 28

[Hei82] K. Heidler. *Wirkungsgraduntersuchung zur Solarenergiekonversion mit Fluoreszenzkollektoren*. PhD thesis, Albert-Ludwigs-Universität Freiburg, 1982. 58

[HHA+10] Matthias Hullin, Johannes Hanika, Boris Ajdin, Jan Kautz, Hans-Peter Seidel, and Hendrik Lensch. Acquisition and analysis of bispectral bidirectional reflectance and reradiation distribution functions. *ACM Transactions on Graphics (Proc. SIGGRAPH 2010)*, pages 1–7, 2010. 13, 19, 65

[HJ09] Toshiya Hachisuka and Henrik Wann Jensen. Stochastic progressive photon mapping. *ACM Transactions on Graphics*, 28(5):1–8, 2009. 26

[HK07] Johannes Hanika and Alexander Keller. Towards hardware ray tracing using fixed point arithmetic. In *Proc. 2007 IEEE/EG Symposium on Interactive Ray Tracing*, pages 119–128, 2007. 14, 86, 91

[HKL09] Johannes Hanika, Alexander Keller, and Hendrik Lensch. Two-level ray tracing with reordering for highly complex scenes. Technical Report 2009–11, Ulm University, 2009. 105

[HKL10] Johannes Hanika, Alexander Keller, and Hendrik Lensch. Two-level ray tracing with reordering for highly complex scenes. In *Proc. of Graphics Interface 2010*, pages 145–152, 2010. 13, 105

[HL90] Pat Hanrahan and Jim Lawson. A language for shading and lighting calculations. *Computer Graphics (Proc. SIGGRAPH '90)*, pages 289–298, 1990. 106

[HM00] Eric Haines and Tomas Möller. Triangle intersection tests. *Dr. Dobb's Journal*, August 2000. 89, 92

[HMF07] Warren Hunt, William Mark, and Don Fussell. Fast and lazy build of acceleration structures from scene hierarchies. In *Proc. 2007 IEEE/EG Symposium on Interactive Ray Tracing*, pages 47–54, 2007. 108, 111

[HMI08] Silja Holopainen, Farshid Manoocheri, and Erkii Ikonen. Goniofluorometer for characterization of fluorescent materials. *Applied Optics*, 47(6):835–842, 2008. 59

[HOJ08] Toshiya Hachisuka, Shinji Ogaki, and Henrik Wann Jensen. Progressive photon mapping. *SIGGRAPH Asia '08: ACM SIGGRAPH Asia 2008 papers*, pages 1–8, 2008. 26

[HQL+10] Qiming Hou, Hao Qin, Wenyao Li, Baining Guo, and Kun Zhou. Micropolygon ray tracing with defocus and motion blur. *ACM Transactions on Graphics (Proc. SIGGRAPH 2010)*, pages 1–10, 2010. 108

[HRB+09] Jared Heinly, Shawn Recker, Kevin Bensema, Jesse Porch, and Christiaan Gribble. Integer ray tracing. *journal of graphics, gpu, and game tools*, 14(4):31–56, 2009. 86

[HSRG07] Charles Han, Bo Sun, Ravi Ramamoorthi, and Eitan Grinspun. Frequency domain normal map filtering. *ACM Transactions on Graphics (Proc. SIGGRAPH 2007)*, page 28, 2007. 112

[Ige99] Homan Igehy. Tracing ray differentials. *Proc. of SIGGRAPH '99*, pages 179–186, 1999. 109, 112

[Inc04] Adobe Systems Inc. Digital negative specification. `http://www.adobe.com/products/dng/pdfs/dng_spec.pdf`, accessed 27/08/2010, 2004. 21

[IWRP06] Thiago Ize, Ingo Wald, Chelsea Robertson, and Steven Parker. An Evaluation of Parallel Grid Construction for Ray Tracing Dynamic Scenes. In *Proc. 2006 IEEE Symposium on Interactive Ray Tracing*, pages 27–55, 2006. 33

[Jak10] Wenzel Jakob. Mitsuba renderer. `http://www.mitsuba-renderer.org/`, 2010. 26

[Jef05] Alan Jeffrey. *Complex Analysis and Applications*. Chapman and Hall/CRC; 2nd edition, 2005. 40

[Jen95] Henrik Wann Jensen. Importance driven path tracing using the photon map. In *Rendering Techniques '95 (Proc. of the Sixth Eurographics Workshop on Rendering)*, pages 326–335, 1995. 28

[Jen96] Henrik Wann Jensen. Global illumination using photon maps. In *Rendering Techniques '96 (Proc. of the Seventh Eurographics Workshop on Rendering)*, pages 21–30, 1996. 26

[JMLH01] Henrik Wann Jensen, Stephen Marschner, Marc Levoy, and Pat Hanrahan. A practical model for subsurface light transport. *Proc. of ACM SIGGRAPH 2001*, pages 511–518, 2001. 37

[Jon00] Ray Jones. Intersecting a ray and a triangle with Plücker coordinates. *Ray Tracing News*, 13(1), 2000. 89, 91

[Kaj86] Jim Kajiya. The Rendering Equation. *Computer Graphics (Proc. SIGGRAPH '86)*, pages 143–150, 1986. 22, 26

[Kea96] Baker Kearfott. Interval computations: Introduction, uses, and resources. *Euromath Bulletin*, 2(1):95–112, 1996. 87

[Kel97] Alexander Keller. Instant radiosity. *Proc. of SIGGRAPH '97*, pages 49–56, 1997. 26

[Kel98] Alexander Keller. *Quasi-Monte Carlo Methods for Photorealistic Image Synthesis*. PhD thesis, University of Kaiserslautern, 1998. 24

[Kem09] Christian Kempter. Estimating brdfs from height fields and scanning electron microscope photographs. Master's thesis, Ulm University, 2009. 36

[KGV83] Scott Kirkpatrick, Daniel Gelatt, and Mario Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983. 46

[KK03] Qifa Ke and Takeo Kanade. Robust subspace computation using L1 norm. Technical Report CMU-CS-03-172, Carnegie Mellon, 2003. 75

[KK04] Thomas Kollig and Alexander Keller. Illumination in the presence of weak singularities. In *Proc. Monte Carlo and Quasi-Monte Carlo Methods 2002*, pages 245–257. Springer, 2004. 120

[Knu81] Donald Knuth. *The Art of Computer Programming, Volume II: Seminumerical Algorithms, 2nd Edition*. Addison-Wesley, 1981. 86, 88

[KS02] Toshi Kato and Jun Saito. Kilauea parallel global illumination renderer. In *Fourth Eurographics Workshop on Parallel Graphics and Visualization*, pages 7–13, 2002. 108

[KS06] Andrew Kensler and Peter Shirley. Optimizing ray-triangle intersection via automated search. In *Proc. 2006 IEEE Symposium on Interactive Ray Tracing*, pages 33–38, 2006. 89

[KSKAC02] Csaba Kelemen, László Szirmay-Kalos, György Antal, and Ferenc Csonka. A simple and robust mutation strategy for the Metropolis light transport algorithm. In *Computer Graphics Forum (Proc. Eurographics 2002)*, pages 531–540, 2002. 29, 30

[KSKK10] Murat Kurt, László Szirmay-Kalos, and Jaroslav Křivánek. An anisotropic BRDF model for fitting and Monte Carlo rendering. *ACM Transactions on Graphics (Proc. SIGGRAPH 2010)*, pages 1–15, 2010. 36

[KSL05] Jan Kautz, Peter-Pike Sloan, and Jaakko Lehtinen. Precomputed radiance transfer: Theory and practice. SIGGRAPH 2005 Courses, 2005. 84

[Lai10] Samuli Laine. Restart trail for stackless BVH traversal. In *Proc. High-Performance Graphics 2010*, pages 107–111, 2010. 33

[LBBS08] Dylan Lacewell, Brent Burley, Solomon Boulos, and Peter Shirley. Raytracing prefiltered occlusion for aggregate geometry. In *IEEE Symposium on Interactive Raytracing 2008*, pages 19–26, 2008. 112

[LJA97] James Leland, Norbert Johnson, and Angelo Arecchi. Principles of bispectral fluorescence colorimetry. *Photometric Engineering of Sources and Systems*, 3140(1):76–87, 1997. 58

[LMW90] Bernd Lamparter, Heinrich Müller, and Jörg Winckler. The ray-z-buffer—an approach for ray tracing arbitrarily large scenes. Technical report, Albert-Ludwigs University at Freiburg, 1990. 107

[LS08] Charles Loop and Scott Schaefer. Approximating Catmull-Clark subdivision surfaces with bicubic patches. *ACM Transactions on Graphics*, 27(1):1–11, 2008. 85

[LV00] Tom Lokovic and Eric Veach. Deep shadow maps. *Proc. of ACM SIGGRAPH 2000*, pages 385–392, 2000. 106

[LW93] Eric Lafortune and Yves Willems. Bi-directional path tracing. In *proceedings of third international conference on computational graphics and visualization techniques (compugraphics âĂŹ93)*, pages 145–153, 1993. 27

[LW95] Eric Lafortune and Yves Willems. A 5d tree to reduce the variance of Monte Carlo ray tracing. In *Rendering Techniques 1995 (Proc. of the Sixth Eurographics Workshop on Rendering)*, pages 11–20, 1995. 26, 28

[LYM07] Christian Lauterbach, Sung-Eui Yoon, and Dinesh Manocha. Ray-strips: A compact mesh representation for interactive ray tracing. In *Proc. 2007 IEEE/EG Symposium on Interactive Ray Tracing*, pages 19–26, 2007. 106

[LYTM08] Christian Lauterbach, Sung-Eui Yoon, Ming Tang, and Dinesh Manocha. ReduceM: Interactive and memory efficient ray tracing of large models. *Computer Graphics Forum*, 27(4):1313–1321, 2008. 106

[Mar98] Marti Maria. Little color management system. `www.littlecms.com`, 1998. 21

[MDH+10] Stefan Menz, Holger Dammertz, Johannes Hanika, Hendrik Lensch, and Michael Weber. Graphical interface models for procedural mesh growing. In *Proc. Vison, Modeling and Visualization*, pages 17–24, 2010. 13

[Moo66] Ramon Moore. *Interval Analysis*. Prentice-Hall, 1966. 87

[Mor66] G.M. Morton. A computer oriented geodetic data base and a new technique in file sequencing. Technical report, IBM Ltd., 1966. 98

[MPBM03a] Wojciech Matusik, Hanspeter Pfister, Matt Brand, and Leonard McMillan. A data-driven reflectance model. *ACM Transactions on Graphics (Proc. SIGGRAPH 2003)*, 22(3):759–769, 2003. 37, 47, 58, 67, 69

[MPBM03b] Wojciech Matusik, Hanspeter Pfister, Matthew Brand, and Leonard McMillan. Efficient isotropic BRDF measurement. In *Proceedings of*

the Eurographics Symposium on Rendering, pages 241–248, 2003. 71

[MRR+53] Nicolas Metropolis, Arianna Rosenbluth, Marshall Rosenbluth, Augusta Teller, and Edward Teller. Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, 21(6):1087–1092, 1953. 23, 28, 46

[MT97] Tomas Möller and Ben Trumbore. Fast, minimum storage ray/triangle intersection. *Journal of Graphics Tools*, 2(1):21–28, 1997. 89, 91

[MWLT00] Stephen Marschner, Stephen Westin, Eric Lafortune, and Kenneth Torrance. Image-based bidirectional reflectance distribution function measurement. *Applied Optics*, 39(16):2592–2600, 2000. 58

[Nas83] Kurt Nassau. *The Physics and Chemistry of Color: The Fifteen Causes of Color*. John Wiley and Sons, 1983. 18

[NDM05] Addy Ngan, Frédo Durand, and Wojciech Matusik. Experimental analysis of BRDF models. In *Rendering Techniques 2005 (Proc. 16th Eurographics Symposium on Rendering)*, pages 117–226, 2005. 51, 58

[NFL07] Paul Navrátil, Donald Fussell, and Calvin Lin. Dynamic ray scheduling to improve ray coherence and bandwidth utilization. In *Proc. 2007 IEEE/EG Symposium on Interactive Ray Tracing*, pages 95–104, 2007. 107, 113

[NNSK99] László Neumann, Attila Neumann, and László Szirmay-Kalos. Compact metallic reflectance models. In *Computer Graphics Forum (Proc. of Eurographics 1999)*, pages 161–172, 1999. 37

[NRH+77] F. Nicodemus, J. Richmond, J. Hsia, I. Ginsberg, and T. Limperis. Geometrical considerations and nomenclature for reflectance. *Final Report, National Bureau of Standards, Washington, DC. Inst. for Basic Standards*, 1977. 22, 66, 106

[NVI09] NVIDIA. CUDA best practices guide 2.3, 2009. 85, 87

[Pat93] Nicholas Patrikalakis. Surface-to-surface intersections. *IEEE Computer Graphics and Applications*, 13(1):89–95, 1993. 94

[PB96] James Proctor and Yvonne Barnes. NIST high accuracy reference reflectometer-spectrophotometer. *Journal of Research of the Nat. Institute of Standards and Technology*, 101(5):619–626, 1996. 59

[PFA+10] Jacopo Pantaleoni, Luca Fascione, Timo Aila, Martin Hill, Sebastian Sylwan, and David Luebke. PantaRay: Directional occlusion for fast cinematic lighting of massive scenes. SIGGRAPH 2010 talks, 2010. 84

[PFHA10] Jacopo Pantaleoni, Luca Fascione, Martin Hall, and Timo Aila. PantaRay: Fast ray-traced occlusion caching of massive scenes. *ACM Transactions on Graphics (Proc. SIGGRAPH 2010)*, pages 1–10, 2010. 108

[PGL+07] Marius Peters, Jan Christoph Goldschmidt, Philipp Loeper, Andreas Gombert, and Gerhard Willeke. Application of photonic structures on fluorescent concentrators. In *Proceedings of the 22nd European Photovoltaic Solar Energy Conference*, 2007. 60

[PH96] Matt Pharr and Pat Hanrahan. Geometry caching for ray-tracing displacement maps. In *Eurographics Rendering Workshop 1996*, pages 31–40, 1996. 108, 118

[PH04] Matt Pharr and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann Publishers Inc., 2004. 26

[Pha05] Matt Pharr. Extended photon map implementation for PBRT. `www.pbrt.org/plugins/exphotonmap.pdf`, 2005. 28, 42

[PKGH97] Matt Pharr, Craig Kolb, Reid Gershbein, and Pat Hanrahan. Rendering complex scenes with memory-coherent ray tracing. *Proc. of SIGGRAPH '97*, pages 101–108, 1997. 107, 108, 118

[PL10] Jacopo Pantaleoni and David Luebke. HLBVH: Hierarchical LBVH construction for real-time ray tracing of dynamic geometry. In *Proc. High-Performance Graphics 2010*, pages 87–95, 2010. 33

[PO08] Anjul Patney and John Owens. Real-time Reyes-style adaptive surface subdivision. *SIGGRAPH Asia '08: ACM SIGGRAPH Asia 2008 papers*, pages 143:1–143:8, 2008. 108

[PT95] Les Piegl and Wayne Tiller. *The NURBS Book*. Springer, 1995. 85

[Qui03] Michael Quinn. *Parallel Programming in C with MPI and OpenMP*. McGraw-Hill Education Group, 2003. 33

[Ren50] William Rense. Polarization studies of light diffusely reflected from ground and etched glass surfaces. *Journal of the optical society of America*, 40(1):55–59, 1950. 39

[Res07] Alexander Reshetov. Faster ray packets – triangle intersection through vertex culling. In *Proc. 2007 IEEE/EG Symposium on Interactive Ray Tracing*, pages 105–112, 2007. 33, 119

[RHF+07] Matthias Raab, Johannes Hanika, Manuel Finkch, Leonhard Grünschloß and Alexander Keller. Benchmarking ray tracing for realistic light transport algorithms, 2007. `http://bwfirt.sourceforge.net/`. 14, 33

[RL00] Szymon Rusinkiewicz and Marc Levoy. QSplat: A multiresolution point rendering system for large meshes. *Proc. of ACM SIGGRAPH 2000*, pages 343–352, 2000. 98

[RUL00] J. Revelles, Carlos Ureña, and Miguel Lastra. An efficient parametric algorithm for octree traversal. In *Proc. of the WSCG 2000*, pages 212–219, 2000. 98

[SB55] W. Stiles and J. Burch. Interim report to the commission internationale de l'Éclairage, Zürich, 1955, on the national physical laboratory's investigation of colour-matching. *Optica Acta*, 2:168–181, 1955. 20

[SB59] W. Stiles and J. Burch. N.P.L. colour-matching investigation: final report. *Optica Acta*, 6:1–26, 1959. 21

[SB87] John Snyder and Alan Barr. Ray tracing complex models containing surface tessellations. *Computer Graphics (Proc. SIGGRAPH '87)*, pages 119–128, 1987. 108

[SBB+06] Abraham Stephens, Solomon Boulos, James Bigler, Ingo Wald, and Steven Parker. An application of scalable massive model interaction using shared memory systems. In *Proceedings of the 2006 Eurographics Symposium on Parallel Graphics and Visualization*, pages 19–26, 2006. 106

[Sch94] Christophe Schlick. An inexpensive BRDF model for physically-based rendering. *Computer Graphics Forum*, 13(3):233–246, 1994. 37

[SFD09] Martin Stich, Heiko Friedrich, and Andreas Dietrich. Spatial splits in bounding volume hierarchies. In *Proc. High Performance Graphics 2009*, pages 7–13, 2009. 33

[SHDL10] Christoph Schied, Johannes Hanika, Holger Dammertz, and Hendrik Lensch. High performance iterated function systems. In *GPU Computing Gems*, page to appear. Morgan Kaufmann, 2010. 13

[Shi00] Peter Shirley. *Realistic Ray Tracing*. AK Peters, Ltd., 2000. 27, 84

[Shi02] Peter Shirley. *Fundamentals of Computer Graphics*. A.K. Peters Ltd., 2002. 17, 18, 21

[SKG+07] A. Schüler, A. Kostro, C. Galande, M. Valle del Olmo, E. de Chambrier, and B.Huriet. Principles of Monte-Carlo Ray-Tracing Simulations of Quantum Dot Solar Concentrators. In *Proceedings of the ISES solar world congress 2007*, 2007. 58

[SM08] Mutsuo Saito and Makoto Matsumoto. SIMD-oriented fast Mersenne twister: A 128-bit pseudorandom number generator. In *Proc. Monte Carlo and Quasi-Monte Carlo Methods 2006*, pages 607–622. Springer, 2008. 26

[SMD+06] Gordon Stoll, William Mark, Peter Djeu, Rui Wang, and Ikrima Elhassan. Razor: An architecture for dynamic multiresolution ray tracing. Technical report 06-21, Department of Computer Science, University of Texas at Austin, 2006. 108, 109, 119

[Smi99] Brian Smits. An RGB-to-spectrum conversion for reflectances. *Journal of Graphics Tools*, 4(4):11–22, 1999. 19

[SN07] P. Susila and J. Naus. A Monte Carlo study of the chlorophyll fluorescence emission and its effects on the leaf spectral reflectance and transmittance under various conditions. *Photochemical & Photobiological Sciences*, 6:894–902, 2007. 58

[Sob94] Ilya Sobol'. *A Primer for the Monte Carlo Method*. CRC Press, 1994. 23, 65

[SSHL97] Peter Shirley, Brian Smits, Helen Hu, and Eric Lafortune. A practitioners' assessment of light reflection models. In *Proceedings of the 5th Pacific Conference on Computer Graphics and Applications*, pages 40–49, 1997. 37

[SSS00] Brian Smits, Peter Shirley, and Michael Stark. Direct ray tracing of displacement mapped triangles. In *Proc. Eurographics Workshop on Rendering Techniques 2000*, pages 307–318, 2000. 108

[Tat05] Natalya Tatarchuk. Practical dynamic parallax occlusion mapping. *ACM SIGGRAPH 2005 Sketches*, page 106, 2005. 85, 108

[TWW88] Joseph Traub, Grzegorz Wasilkowski, and Henryk Woźniakowski. *Information Based Complexity*. Academic Press, 1988. 24

[Vea97] Eric Veach. *Robust Monte Carlo Methods for Light Transport Simulation*. PhD thesis, Stanford University, 1997. 26, 96

[VG94] Eric Veach and Leonidas Guibas. Bidirectional estimators for light transport. In *Rendering Techniques '94 (Proc. of the Fifth Eurographics Workshop on Rendering)*, pages 147 – 161, 1994. 27

[VG95] Eric Veach and Leonidas Guibas. Optimally combining sampling techniques for Monte Carlo rendering. *Proc. of SIGGRAPH '95*, pages 419–428, 1995. 27, 115

[VG97] Eric Veach and Leonidas Guibas. Metropolis light transport. *Proc. of SIGGRAPH '97*, pages 65–76, 1997. 28

[VGmm98] Terrence Vergauwen, Jean-Philippe Grimaldi, and many more. LuxRender: GPL physically based renderer. `http://www.luxrender.net/`, 1998. 26

[Wäc08] Carsten Wächter. *Quasi-Monte Carlo Light Transport Simulation by Efficient Ray Tracing*. PhD thesis, Universität Ulm, 2008. 83, 86, 111

[Wal04] Ingo Wald. *Realtime Ray Tracing and Interactive Global Illumination*. PhD thesis, Saarland University, 2004. 33, 91, 92

[Wal07] Ingo Wald. On fast construction of SAH based bounding volume hierarchies. In *Proc. 2007 IEEE/EG Symposium on Interactive Ray Tracing*, pages 33–40, 2007. 33, 111

[War92] Gregory Ward. Measuring and modeling anisotropic reflection. *Computer Graphics (Proc. SIGGRAPH '92)*, 26(2):265–272, 1992. 58

[War02] Henry Warren. *Hacker's Delight*. Addison-Wesley Longman Publishing Co., Inc., 2002. 88

[WBB08] Ingo Wald, Carsten Benthin, and Solomon Boulos. Getting rid of packets - efficient SIMD single-ray traversal using multi-branching BVHs. In *Proc. 2008 IEEE/EG Symposium on Interactive Ray Tracing*, pages 49–57, 2008. 33

[WBWS01] Ingo Wald, Carsten Benthin, M. Wagner, and Philipp Slusallek. Interactive rendering with coherent ray tracing. In *Computer Graphics Forum (Proc. Eurographics 2001)*, pages 153–164, 2001. 106

[WGRK+97] A.J. Welch, C. Gardner, R. Richards-Kortum, E. Chan, G. Criswell, J. Pfefer, and S. Warren. Propagation of fluorescent light. *Lasers in Surgery and Medicine*, 21:166–178, 1997. 58

[WH06] Ingo Wald and Vlastimil Havran. On building fast *k*d-trees for ray tracing, and on doing that in $\mathcal{O}(n \log n)$. In *Proc. 2006 IEEE Symposium on Interactive Ray Tracing*, pages 18–20, 2006. 33, 111

[Whi80] Turner Whitted. An improved illumination model for shaded display. *Communications of the ACM*, 23(6):343–349, 1980. 27, 84

[WMG+07] Ingo Wald, William Mark, Johannes Günther, Solomon Boulos, Thiago Ize, Warren Hunt, Steven Parker, and Peter Shirley. State of the art in ray tracing animated scenes. In *State of the Art Reports, Eurographics 2007*, pages 1691–1722, 2007. 84

[Woo06] Sven Woop. *DRPU: A Programmable Hardware Architecture for Real-time Ray Tracing of Coherent Dynamic Scenes*. PhD thesis, Saarland University, 2006. 89, 92

[WPO96] Andrew Woo, Andrew Pearce, and Marc Ouellette. It's really not a rendering bug, you see... *IEEE Computer Graphics & Applications*, 16(5):21–25, 1996. 83, 96

[WRC88] Greg Ward, Francis Rubinstein, and Robert Clear. A ray tracing solution for diffuse interreflection. *Computer Graphics (Proc. SIGGRAPH '88)*, pages 85 – 90, 1988. 26

[WTP01] Alexander Wilkie, R.F. Tobler, and W. Purgathofer. Combined rendering of polarization and fluorescence effects. In *Proceedings of the 12th Eurographics Workshop on Rendering*, 2001. 58

[WWLP06] Alexander Wilkie, Andrea Weidlich, C. Larboulette, and W. Purgathofer. A reflectance model for diffuse fluorescent surfaces. In *Proceedings of the 4th International Conference on Computer Graphics and Interactive Techniques in Australasia and the Southeast Asia*, 2006. 58

[XR10] X-Rite. Ma98 specifications: Portable multi-angle spectrophotometer. `http://www.xrite.com/documents/literature/en/L10-372_MA98_en.pdf`, 2010. 44

[YM06] Sung-Eui Yoon and Dinesh Manocha. R-LODs: fast LOD-based ray tracing of massive models. *SIGGRAPH '06: ACM SIGGRAPH 2006 Sketches*, page 67, 2006. 109

[Zas81] Armin Zastrow. *Physikalische Analyse der Energieverlustmechanismen im Fluoreszenzkollektor*. PhD thesis, Albert-Ludwigs-Universität Freiburg, 1981. 60