Karlsruhe Institute of Technology

Irradiance Importance Sampling

Bachelor Thesis of

Alisa Jung

At the Department of Informatics
Institute for Visualization and Data Analysis (IVD)

Reviewer:	Prof. Dr. Carsten Dachsbacher
Second reviewer:	Prof. Dr. Hartmut Prautzsch
Advisor:	Dipl-Inf. Florian Simon

Duration: 02.10.2014 – 02.02.2015

Abstract

For many years path tracing has been a popular choice when it came to rendering photorealistic images. By combining different sampling techniques, the algorithm can be adapted to the complexity and challenges of individual scenes. That way effects like indirect lighting and caustics can be rendered realistically.
This work will introduce a new approach to include irradiance importance sampling to a regular path tracing algorithm. Before an image is rendered, the incident radiance (direct and indirect) across a scene is approximated and cached. The information from these caches can be used to generate paths with a potentially high contribution to the rendering equation.
Caching and importance sampling the incident radiance enables us to compensate for some of the drawbacks of sampling paths solely according to local surface properties. This approach may also be combined with known sampling techniques, such as next event estimation or generating samples from a BSDF (bidirectional scattering distribution function), using multiple importance sampling.

Zusammenfassung

Seit einiger Zeit zählt Path Tracing zu den beliebtesten Verfahren bei der Erzeugung fotorealistischer Bilder. Der Algorithmus kann durch Einsatz verschiedener Sampling-Techniken variiert und an die Komplexität der darzustellenden Szene angepasst werden. So sind etwa indirekte Beleuchtung und Kaustiken realistisch darstellbar.
Diese Arbeit stellt den Einsatz von Caches zur Approximation der einfallenden Beleuchtung aus verschiedenen Richtungen vor. Vor der Berechnung eines Bildes wird die einfallende Beleuchtung (direkt und indirekt) überall in der Szene approximiert und in Caches gespeichert. Diese können beim Path Tracing genutzt werden, um Pfade mit potenziell hohem Beitrag zur Rendergleichung zu erzeugen.
Dadurch lassen sich manche Schwächen ausgleichen, die auftreten, wenn diese Pfade nur nach Oberflächeneigenschaften erzeugt werden. Das Verfahren lässt sich mit bekannten Sampling-Techniken wie Next Event Estimation oder klassischem Sampling nach der BSDF (bidirectional scattering distribution function) durch den Einsatz von Multiple Importance Sampling verbinden.

Contents

1. Introduction

The concept of path tracing was initially introduced by Kajiya in 1986 ([Kaj86]), along with the rendering equation. Basically, the idea was to create paths starting from the camera and to extend these path at each surface intersection according to some sampling strategy (e.g. according to the local BSDF).
Over the years, several improvements to this approach were developed. Examples include next event estimation (creating direct connections from a surface point to a light source) or bidirectional path tracing ([LW93], [Laf96]), where paths are created not only from camera to light source, but also from light to camera.

We will introduce a new sampling strategy for extending paths that can be combined with known sampling techniques, and evaluate if and when this strategy can improve the quality of the resulting images. Our strategy is based on caching the incident radiance, i.e. we will place caches across the scene and each of these caches will store the incident radiance for different directions, thus allowing us to sample directions from irradiance.

Chapter 2 will outline some concepts of probability theory and numerical integration, and introduce the problem of physically based light transport. In chapter 3 we will introduce path tracing, a common algorithm to render photorealistic images. Chapter 4 covers our new approach: Creating caches that can later be used for irradiance importance sampling. In chapter 5 we show how we combined these caches with our basic path tracing algorithm and evaluate the results. Finally, chapter 6 references some related concepts.

2. Basics

This chapter covers all basic concepts and definitions used in the main part of this thesis. First we will cover some probability theory and Monte Carlo integration. Furthermore we present multiple importance sampling, a common variance reduction technique to improve Monte Carlo integration. After that we will introduce physically based rendering and the rendering equation.

2.1 Mathematical Basics

2.1.1 Some Probability Theory

For all probability related concepts we use the same definitions and terminology as in [Vea98]. This section will only give a short overview on those equations that we will really need later on. For simplicity we will skip most of the derivations and equations that are not important to understand the main part of this thesis. See [Vea98, chapter 2.3] for additional background information on probability theory.

The Probability Density Function

We will start with probability density functions (**pdf**). In general, a pdf is a continuous function $\Omega \to \mathbb{R}_0^+$ with the additional condition

$$\int_\Omega p(x)dx = 1. \tag{2.1}$$

For one-dimensional $\Omega \subseteq \mathbb{R}$, we can use p to compute the probability $\mathcal{P}$ of a random variable X with that density lying in an Interval $[a, b] \subseteq \Omega$ as follows:

$$\mathcal{P}\{a \leq X \leq b\} = \int_a^b p(x)dx.$$

We can also extend this principle to n-dimensional $\Omega \subseteq \mathbb{R}^n$ and random variables $X = (X^1, \ldots, X^n) \in \Omega$. At first we need the *joint cumulative distribution function* P:

$$P(X) = P(x^1, \ldots, x^n) := \mathcal{P}\{X^i \leq x^i \ \forall i = 1, \ldots, n\}.$$

The *joint density function* p for multidimensional Ω can be obtained from

$$p(x^1, \ldots, x^n) = \frac{\partial^n P}{\partial x^1 \cdots \partial x^n}(x^1, \ldots, x^n). \tag{2.2}$$

Note that this definition also works for $n = 1$ and agrees with the conditions from above (equation 2.1).

The joint density function p can be used to compute the probability of X being part of some Lebesgue measurable subset $D \subseteq \Omega$:

$$\mathcal{P}\{X \in D\} = \int_D p(X)d(X) = \int_D p(x^1, \ldots, x^n)dx^1 \cdots dx^n.$$

Measures

Throughout this thesis, we are going to use different criteria (e.g. surface area, solid angle or projected solid angle) to measure the same domain. This also affects the form of the applied pdfs. In the previous section we always integrated directly using dx or dx^i. We will now introduce a more general approach that allows us to transform between different measures.

First of all, we define

$$P(D) := \mathcal{P}\{X \in D\}.$$

P is a measure function called *probability measure* for a random variable $X \in \Omega$. Just as in equation 2.1, we demand $P(\Omega) = 1$. Similar to equation 2.2, we obtain a corresponding pdf p depending on a measure μ as follows:

$$p(x) = \frac{dP}{d\mu}(x). \tag{2.3}$$

There are different ways to pick D and μ and therefore different possible pdfs. All of them satisfy

$$P(X) = \int_D p(x)d\mu(x)$$

with their very own measure μ that was used to define them in 2.3.

Expectation and variance

The *expectation* or *expected value* E of a random variable X with density p and measure μ represents the expected average value of a huge number of random variables with density p. It is defined as

$$E(X) = \int_\Omega x \cdot p(x)d\mu(x).$$

This equation also works if we consider some function f of X instead of X itself:

$$E(f(X)) = \int_\Omega f(x) \cdot p(x) d\mu(x).$$

The *variance* indicates how much the random variables vary from the expected value. A variance of 0 means that all random variables equal the expectation. The variance is always greater than or equal to zero and given by

$$Var(X) = E[(X - E(X))^2].$$

Another measure for the variation of random variables is the *standard deviation*:

$$\sigma(X) = \sqrt{Var(X)}.$$

2.1.2 Monte Carlo Integration

Monte Carlo integration is a non-deterministic unbiased numerical method for estimating integrals and is especially useful whenever those integrals are high dimensional. As explained later on, solving the light transport problem requires computing such an integral over a theoretically infinite number of dimensions. We will now present the Monte Carlo estimator, show some of its advantages and introduce importance sampling to improve it.

Say we want to integrate a function $f : \Omega \to \mathbb{R}$ over Ω, where Ω has an arbitrary dimensionality and μ is some measure function on Ω:

$$I = \int_\Omega f(x) d\mu(x). \tag{2.4}$$

Let $p(x)$ be a probability density function on Ω with

$$\forall x \in \Omega : f(x) \neq 0 \Rightarrow p(x) > 0$$

and let X be a random variable with density p. Then we can also express I as follows:

$$I = \int_\Omega f(x) d\mu(x) = \int_\Omega \frac{f(X)}{p(X)} \cdot p(X) d\mu(X) = E\left(\frac{f(X)}{p(X)}\right).$$

Thus we can interpret I as the expectation of $f(X)/p(X)$. This expected value can be approximated using the Monte Carlo estimator F_N:

$$I = E\left(\frac{f(X)}{p(X)}\right) \approx \frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i)}{p(X_i)} =: F_N. \tag{2.5}$$

We use the index N to indicate that the estimator uses N samples.

Error and Bias

Given an estimator F_N and the quantity I we want to estimate, $F_N - I$ is called the *error* of the estimator. The *bias* β of an estimator is the expected value of its error:

$$\beta[F_N] = E(F_N - I).$$

If $\beta[F_N] = 0$ independent of the number of samples N, the estimator is called *unbiased*. This is a desirable property for any estimator, because it guarantees that it will yield the correct value on average.

Advantages of Monte Carlo integration

The Monte Carlo estimator is unbiased, since

$$\beta[F_N] = E\left(\frac{1}{N}\sum_{i=1}^{N}\frac{f(X_i)}{p(X_i)} - I\right) = E\left(\frac{1}{N}\sum_{i=1}^{N}\frac{f(X_i)}{p(X_i)}\right) - E(I) = E\left(\frac{f(X)}{p(X)}\right) - I = I - I = 0.$$

When dealing with one or two dimensions, other techniques such as quadrature rules perform way better than F_N. However, Monte Carlo integration is practically the only integration method hat can handle infinite-dimensional integrals at all.
It also has some desirable properties when dealing with high-dimensional integrals. First of all, its quality does not depend on the dimensionality. With deterministic approaches, the number of samples needed for the same standard deviation explodes exponentially with each additional dimension. In contrast, the standard deviation of F_N is always proportional to $1/\sqrt{N}$, regardless of the dimensionality. More precisely,

$$\sigma(F_N) = \frac{1}{\sqrt{N}}\sigma\left(\frac{f(X)}{p(X)}\right), \tag{2.6}$$

as elaborated in [Vea98]. The quality of the estimator solely depends on the number of samples and on the quality of p (i.e. how proportional p and f are). The estimator also converges at a rate of $\mathcal{O}(1/\sqrt{N})$.

Another advantage of Monte Carlo integration is its ability to handle singularities, i.e. functions with discontinuities. In computer graphics, this is the case with ideal specular surfaces (e.g. mirrors), where light arriving from one direction is only scattered along exactly one outgoing direction; in other words, only light from one single direction contributes to the color we see on that surface.

See [Vea98, chapter 2] for additional information on Monte Carlo integration.

Importance Sampling

The difficulty of Monte Carlo integration lies in finding a good pdf. It can be shown that the optimal choice is

$$p^*(x) = \frac{f(x)}{I}$$

with a derivation of 0 (see equation 2.6 and [Vea98, chapter 2.5.2]) - one sample generated by p^* would suffice to compute the exact value of I. Unfortunately, since I is the value we want to approximate, we can't know the exact value of I in advance (if we did, numerically approximating I would be pointless).
However, there are ways to quickly construct a pdf that is at least roughly proportional to f. Generating Samples from a probability density function that is supposedly proportional to f is called *Importance Sampling*. Section 3.3 shows some Importance Sampling strategies specifically designed for approximating the rendering equation.

Importance Sampling is one of several so-called *variance reduction methods*. In addition to Importance Sampling we only used Russian Roulette, which is explained in section 3.6. See [Vea98, chapters 2.5 - 2.8] for an introduction to other variance reduction methods.

2.1.3 Multiple Importance Sampling

It can be hard to construct one general pdf that performs well in any case. Most times, it is easier to construct different pdfs that perform well under different circumstances, and weight them together. This can greatly improve Importance Sampling and is called *Multiple Importance Sampling (MIS).*

As we will see in section 3.3, there are different strategies to create pdfs for physically based rendering, but all of them show major weaknesses under different conditions. We will now show how any sampling strategies can be combined in a way that, given a pdf p (p') that performs well (badly) in case A and badly (well) in case B, whenever we have case A p is chosen more often or weighted more heavily than p'. Thus we can reduce variance and make Monte Carlo integration more robust.

There are two main categories for MIS: The multi-sample model, where many samples are generated and weighted together for one step, and the one-sample model, where only one sample is generated per step, which means we have to pick and weight one pdf for every step.

Given a domain Ω, a function $f : \Omega \to \mathbb{R}$ and a measure μ, both models use a set of n sampling strategies with pdfs $p_1, \ldots, p_n$ to approximate equation 2.4.

In order to apply the models, $f(x)$ and $p_i(x)$ must be defined for any $x \in \Omega, i \in 1, \ldots, n$, and we have to be able to generate samples from $p_1, \ldots, p_n$.

The Multi-Sample Model

The multi-sample model can be applied whenever we are able to generate one or more samples from each sampling strategy (= pdf) to approximate an integral.

First, a fix number $n_i \geq 1$ of samples is generated from each sampling strategy i according to its pdf p_i. We will denote each sample $X_{i,j}$ with $i = 1, \ldots, n$ and $j = 1, \ldots, n_i$ referring to the sampling strategy and its number of samples respectively.
All samples generated from the same strategy i share one weighting function w_i. This allows us to assign different weights to samples from the same strategy. Finally we can compute the *multi-sample estimator* F to approximate 2.4:

$$F = \sum_{i=1}^{n} \frac{1}{n_i} \left(\sum_{j=1}^{n_i} w_i(X_{i,j}) \frac{f(X_{i,j})}{p_i(X_{i,j})} \right).$$

If we want F to be unbiased ([Vea98, Lemma 9.1]), there are two conditions for our weighting functions:

$$f(x) \neq 0 \Rightarrow \sum_{i=1}^{n} w_i(x) = 1 \text{ and} \tag{2.7}$$

$$\forall i = 1, \ldots, n : p_i(x) = 0 \Rightarrow w_i(x) = 0. \tag{2.8}$$

Note that for $n = 1, n_1 = N$ and $w_1(x) = 1/N$ (one pdf with N uniformly weighted samples) the resulting estimator F equals F_N from equation 2.5.

Path tracing usually uses the multi-sample model to combine BSDF sampling and next event estimation, a short overview on that case is given in section 3.5.

The Balance Heuristic

One possible way to define the weighting functions is

$$w_i(x) = \frac{n_i p_i(x)}{\sum_k n_k p_k(x)}. \tag{2.9}$$

This combining strategy is called *balance heuristic*. It shows a minimal variance compared to any other combination strategy - [Vea98, 9.2.2, page 264] shows that "no other combination strategy is much better".

The Power Heuristic

The power heuristic is a generalization of the balance heuristic. It uses an exponent β for all weights:

$$w_i(x) = \frac{(n_i p_i(x))^\beta}{\sum_k (n_k p_k(x))^\beta}. \tag{2.10}$$

As Veach showed in [Vea98, chapter 9.3.1], the power heuristic for $\beta = 2$ can lead to better results for very rough or barely rough surfaces.

Other possible weighting functions include discrete $w_i(x) \in \{0, 1\}$ or uniform functions $w_i(x) = 1/n$, the cutoff heuristic, which ignores samples with small probabilities, or the maximum heuristic (the power heuristic for $\beta \to \infty$).

The One-Sample Model

There are times when we cannot or do not want to generate multiple samples from multiple pdfs, but still have several sampling strategies to choose from. In that case, we use a set of fix probabilities $c_1, \dots c_n$ with $\sum c_i = 1$ to pick one of our n sampling strategies and take one sample X_j from the chosen pdf p_j. The *one-sample estimator* to approximate 2.4 is given as

$$F = \frac{w_j(X_j) f(X_j)}{c_j p_j(X_j)}. \tag{2.11}$$

For this model the balance heuristic is such a good choice for the weighting function that we don't have to use any improvements like the power heuristic ([Vea98, 9.2.4]). Plus, as long as we consider conditions 2.7 and 2.8, equation 2.11 also gives an unbiased estimator.

In section 5.2.1, we will use the one-sample model to combine two different sampling strategies for extending a path. We will also decide on c_1 and c_2 depending on the current surface probabilities instead of using the same c_1 and c_2 every time.

2.2 Physically based light transport

2.2.1 Assumptions

Before we introduce the equations we used to model light transport, we are going to make a few assumptions. These constraints won't produce any significant difference compared to a physically accurate model, as long as we only render "ordinary" scenes that don't rely heavily on one of the excluded phenomena. However, it is always important to decide if the current model can produce valid results for a certain scene, or if some of those assumptions have to be discarded.

Basically we assume a geometric optics model, where light only travels in straight lines and only interacts (gets emitted, scattered or absorbed) at surfaces. We also ignore all behaviour that arises from the wave or quantum characteristics of light.
Thus the proposed model can't accurately render phenomena like participating media (e.g. fog or smoke), subsurface scattering (e.g. wax) or relativistic effects, diffraction or fluorescence. It also ignores varying refraction indices, be it dispersion at a surface, caused by a different index for different wave lengths, or media with a continuously varying index. With this model, light of any wavelength behaves the same, and no energy is lost while traveling between surfaces.

There are some additional choices caused by the Mitsuba Renderer [Jak10] which we used as an implementation framework. Mitsuba supports a wide range of surfaces scattering models (see [Jak10, Mitsuba Documentation chapter 8.2]) and allows for transmitting and ideal specular surfaces. However, in our implementation and test scenes, we ignored the more complex BSDFs for simplicity. We also ignore point light sources and terminate any path as soon as it hits a light source.

2.2.2 Notations

From now on, we will use the following notations (depicted in figure 2.1):

- $\mathcal{S}^2$ will be the set of all directions.
- $\omega \in \mathbb{R}^3$, $||\omega|| = 1$ will describe any direction on the unit sphere.
- x will refer to a surface point and $N(x)$ to the surface's normal at x.
- σ is the surface area measure on $\mathcal{S}^2$.
  $\sigma(\omega)$ represents the solid angle covered by ω.
- $\sigma_x^\perp(\omega) := |\omega \cdot N(x)| \cdot \sigma(\omega)$ is the projected solid angle at x.
  Sometimes we might just use $\sigma^\perp(\omega)$ to refer to projected solid angles in general.
  Note that the solid angle and projected solid angle are unitless.
- $dA(x)$ will refer to an infinitesimal surface area around x.
- $\mathcal{H}_+^2(x) = \{\omega \in \mathcal{S}^2 | \omega \cdot N(x) > 0\}$ is the upper hemisphere and
  $\mathcal{H}_-^2(x) = \{\omega \in \mathcal{S}^2 | \omega \cdot N(x) < 0\}$ is the lower hemisphere at x.
- $L_o(x, \omega)$ represents the radiance leaving x into direction ω.
- $L_i(x, \omega)$ represents the incoming radiance at x from direction ω.
- $ray(x, \omega)$ refers to a ray with origin x and direction ω.
- $x_{\mathcal{M}}(x, \omega)$ is the *ray casting function*. It returns the first surface point that is hit by $ray(x, \omega)$.

In free space (where x is not a surface point) $L_i(x, \omega) = L_o(x, -\omega)$. At surfaces the relation between L_i and L_o is more complex, see section 2.17.

Also note that for $L_i(x, \omega_i)$ the photons are actually moving in direction $-\omega_i$, as the $\omega \in \mathbb{R}^3$ always point from x towards the hemisphere while the (incident) photons move through the hemisphere towards x.

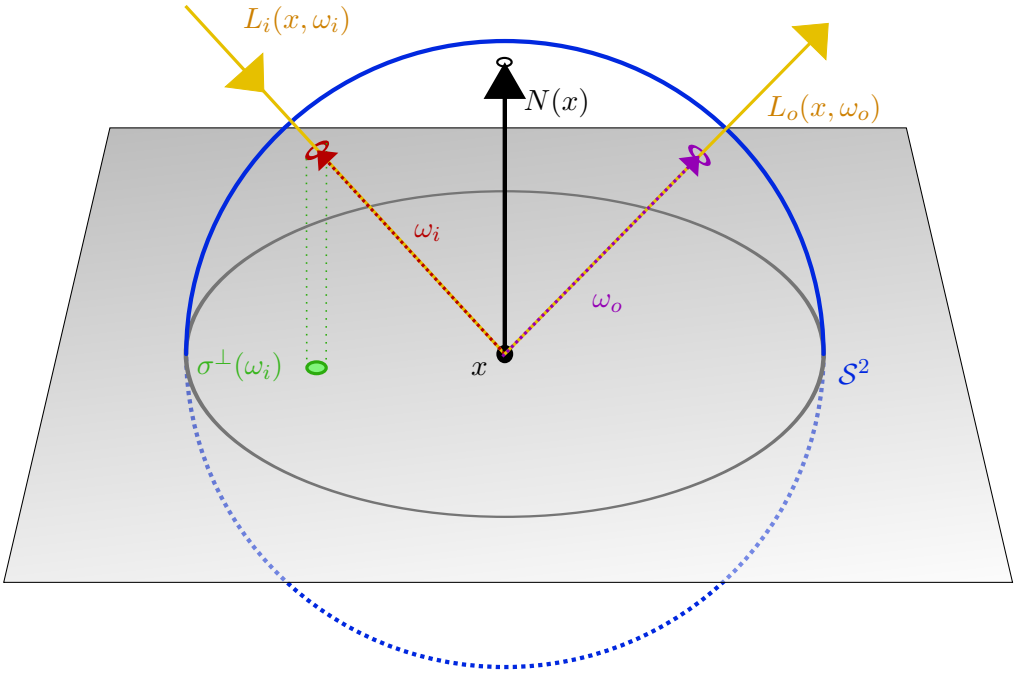

Figure 2.1: The upper blue line represents $\mathcal{H}_+^2$, the dotted blue line represents $\mathcal{H}_-^2$.

2.2.3 Radiometry

This section will briefly cover the physical background of radiance and irradiance.
Physically speaking, the *irradiance* is defined as power Φ (in Watt) per unit surface area A (in m^2) at some point $x \in \mathbb{R}^3$:

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

The term irradiance and E are used for the total *incident* radiation on *one* hemisphere only. Radiance *leaving* the surface is referred to as *radiant exitance* and can happen on both hemispheres.

The *radiance* itself can be expressed as

$$L(x, \omega) = \frac{d^2\Phi(x, \omega)}{dA(x)d\sigma_x^{\perp}(\omega)}.$$

It measures the radiance at x from an infinitesimal cone around direction ω. The definition of L can also be extended to include different wavelengths; we will ignore that.
We will use ω_i and $L_i(x, \omega_i)$ to refer to radiance arriving at a point x from direction ω_i (photons moving along $-\omega_i$), and $L_o(x, \omega_o)$ when we talk about the radiance leaving x into direction ω_o.

We can also express irradiance as an integral of incoming light as follows:

$$E(x) = \int_{\mathcal{H}^2_+(x)} L(x, \omega) d\sigma^{\perp}(\omega)$$

or

$$dE(\omega_i) = L_i(\omega_i) d\sigma^{\perp}(\omega_i). \tag{2.12}$$

Experiments have shown that irradiance and outgoing radiance are proportional, i.e.

$$dL_o(\omega_o) \propto dE(\omega_i).$$

We will not explain this topic any further, see [Vea98, chapter 9.3] for a more detailed introduction on radiometric quantities.

2.2.4 The bidirectional scattering distribution function (BSDF)

The *bidirectional scattering distribution function* (**BSDF**) is used to describe how surfaces scatter light. It indicates how much of the energy arriving at a point x from an incoming direction ω_i is scattered into any outgoing direction ω_o.

Usually, the BSDF is divided into a *bidirectional reflectance distribution function* (BRDF) and a *bidirectional transmittance distribution function* (BTDF), that cover the reflectance and transmittance at a surface respectively. The BRDF only uses incoming and outgoing directions on the same hemisphere, whereas the BTDF only uses directions on opposite hemispheres. Since the BSDF is simply the union of two BRDFs and two BTDF (one for each side of the surface), we will only use the BSDF later on. We will denote BRDFs with f_r, BTDFs with f_t and BSDFs with f_s.

Parameterization

The BSDF f_s at some surface point x is defined as

$$f_s(\omega_i, x, \omega_o) = \frac{dL_{o,s}(\omega_o)}{dE(\omega_i)}.$$

Using equation 2.12 we can also write f_s as

$$f_s(\omega_i, x, \omega_o) = \frac{dL_{o,s}(\omega_o)}{L_i(\omega_i)d\sigma^{\perp}(\omega_i)}. \tag{2.13}$$

It describes the relation of radiance arriving from direction ω_i to $L_{o,s}(x, \omega_o)$ (radiance scattered along ω_o). We add the index s to L_o to indicate that we are only considering outgoing light that was scattered, and ignore outgoing light that was emitted at x.

With physically based rendering, we can always assume that the incoming radiance originated at some point x_0 (either it was emitted at x_0 along $-\omega_i$ and hit x directly, or it was emitted somewhere else, scattered across the scene and finally scattered from x_0 to x). Theoretically a ray with origin x and direction ω_o may never intersect any geometry again, which means that light scattered along ω_o leaves the scene, but those cases are not interesting for us. So, assuming that the light scattered at x along ω_o hits the scene again at some point x_2, we can also use the following notation for f_s:

$$f_s(x_0 \to x \to x_2) = f_s(\omega_i, x, \omega_o). \tag{2.14}$$

Properties

We can make two assumptions about physically correct BRDFs that do not necessarily apply to BTDFs and BSDFs. First of all, physically correct BRDFs are always symmetric:

$$\forall x, \omega_i, \omega_o : \; f_r(\omega_i, x, \omega_o) = f_r(\omega_o, x, \omega_i).$$

Plus, we can assume that the scattered energy never exceeds the incoming energy, this property is called *energy conservation*:

$$\forall \omega_i \in \mathcal{H}_i^2 : \; \int_{\mathcal{H}_o^2} f_r(\omega_i, x, \omega_o) d\sigma^{\top}(\omega_o) \leq 1.$$

With some surfaces these conditions do not hold for the BSDF or BTDF. To include such surfaces to a bidirectional path tracing algorithm or path tracing combined with photon mapping, we also have to use the *adjoint BSDF*, which is basically the BSDF with switched arguments ($f_s(\omega_o, x, \omega_i)$ instead of $f_s(\omega_i, x, \omega_o)$, see [Vea98, chapter 3.7.3 - 3.7.6]).
The adjoint BSDF is used to sample paths that carry importance and for scattering photons, while we can use the original BSDF for paths that carry radiance. Later on, we will use the adjoint BSDF in our photon mapping step, as explained in section 4.2.1.

Delta-BSDFs

Ideal specular surfaces have the unique property that for every outgoing direction ω_o there is exactly one incoming direction ω_i, so that all of the radiance arriving at ω_i is scattered along ω_o. For a given x and ω_o (or x and ω_i), the BSDF describing this surface is always 0 except for one ω_i (or ω_o). We call such a BSDF a *delta-BSDF*. All other BSDFs contain a diffuse component, we call those *smooth* BSDFs.

2.2.5 The Rendering Equation

The goal of our algorithm is to render an image, which means we have to compute the color of each individual pixel. In order to do that we model each pixel j as a hypothetical sensor with area $\mathcal{M}$. The so-called *sensor sensitivity function* $W_e^j(x,\omega)$ describes how this sensor responds to light arriving at $x \in \mathcal{M}$ from direction ω. The color of the pixel is given as the total sensor response, called the *measurement equation*:

$$I^j = \int_{\mathcal{M}\times\mathcal{S}^2} W_e^j(x,\omega) L_i(x,\omega) dA(x) d\sigma_x^\perp(\omega). \tag{2.15}$$

Sometimes the scenes we want to render are "open" - for some directions ω' and starting points x', $ray(x',\omega)$ may never intersect any geometry. This means that no light L_i can arrive at x' along ω' and the integrand will be 0 for ω'. For all other directions there has to be one point $x_{\mathcal{M}}(x,\omega)$ in the scene that was hit by a ray from x along ω.

Since our model only allows light to change at surfaces, we can express the incident radiance as

$$L_i(x,\omega) = L_o(x_{\mathcal{M}}(x,\omega), -\omega).$$

The outgoing light can be divided into scattered and emitted light:

$$L_o(x,\omega) = L_e(x,\omega) + L_{o,s}(x,\omega).$$

The emitted light $L_e(x,\omega)$ is given by the scene description. We can obtain the scattered light when we convert equation 2.13 to the *(surface) scattering equation*:

$$L_{o,s}(x,\omega) = \int_{\mathcal{S}^2} L_i(x,\omega_i) \cdot f_s(\omega_i, x, \omega_o) d\sigma^\perp(\omega_i). \tag{2.16}$$

Combining these equations leads to the *rendering equation*

$$L_o(x,\omega) = L_e(x,\omega) + \int_{\mathcal{S}^2} L_i(x,\omega) \cdot f_s(\omega_i, x, \omega_o) d\sigma^\perp(\omega_i). \tag{2.17}$$

Veach [Vea98, chapter 3.7.2] uses the term *light transport equation* for a slightly different form of the same equation:

$$L_o(x,\omega_o) = L_e(x,\omega_o) + \int_{\mathcal{S}^2} L_o(x_{\mathcal{M}}(x,\omega_i), -\omega_i) \cdot f_s(\omega_i, x, \omega_o) d\sigma_x^\perp(\omega_i).$$

In theory the rendering equation (or the light transport equation) could now be solved recursively. In praxis they are usually solved with Monte Carlo integration.
Both equations represent the outgoing light at x into ω_o as the sum of the radiance emitted along ω_o and the integral of all scattered incident radiance.

2.2.6 Three-point form of the rendering equation

In this section we will show how to express the rendering equation 2.17 over surface area by using surface points instead of directions. This makes the arguments more relatable to an actual path from surface point to surface point and leads us to a non-recursive solution for the rendering equation which is the basis of path tracing.

As we want to use surface points instead of direction vectors, we need a way to convert between the projected solid angle measure used in 2.17 and the surface area measure. Figure 2.2 depicts the relations between the (projected) solid angle at some surface point x and the surface area at another point x'.

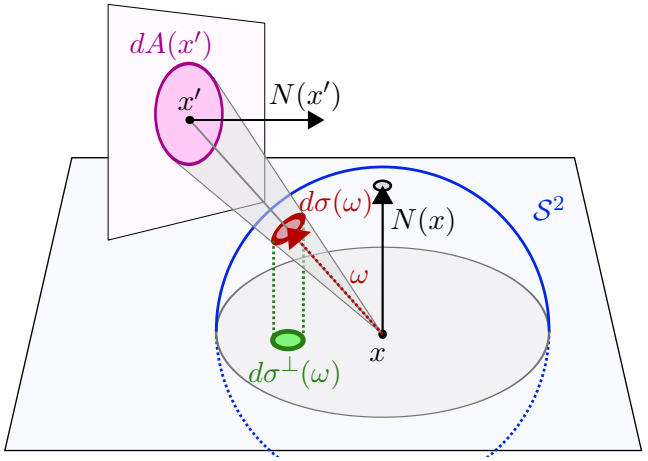

Figure 2.2: Connection between different measures

First we define the *geometry term*

$$G(x \leftrightarrow x') = V(x \leftrightarrow x') \frac{|\omega \cdot N(x)||\omega \cdot N(x')|}{|x - x'|^2}$$

where $V(x \leftrightarrow x')$ is 1 whenever x and x' are visible from each other and 0 otherwise. V is also called *visibility factor*.
We can use G to switch between the measures as follows:

$$d\sigma^{\perp}(\omega) = |\omega \cdot N(x)| d\sigma(\omega) = G(x \leftrightarrow x') dA(x'). \tag{2.18}$$

Sometimes we will write $cos(x \to x')$ instead of $\omega \cdot N(x)$.

Given a surface point x and direction ω such that $x' = x_{\mathcal{M}}(x, \omega)$, we also define

$$L(x \to x') := L(x, \omega).$$

With equation 2.18 and the notation from 2.14 we can now express the rendering equation over the union of all scene surfaces $\mathcal{M}$:

$$L(x' \to x'') = L_e(x' \to x'') + \int_{\mathcal{M}} L(x \to x') f_s(x \to x' \to x'') G(x \leftrightarrow x') . dA(x) \tag{2.19}$$

This is also called the *three-point form* of the light transport equation.

3. Path Tracing

In this chapter we show how to express the rendering equation over path space and represent the light transport problem without any recursions. This formulation lets us design a basic path tracing algorithm, where we create random paths to approximate the rendering equation. We will outline some sampling strategies that are commonly used for path tracing and shortly introduce our new approach for sampling incident radiance. These strategies have different effects on the quality of the resulting images and can be combined using Multiple Importance Sampling. Last we will introduce Russian Roulette as a method to reduce computation time.

3.1 Path Space

First, we need to define paths and path space:

- $x = x_0 x_1 \dots x_k$ is a path with length $1 \leq k < \infty$ defined by surface points x_i
- Ω_k is the set of all paths of length k
- $\mu_k(D) = \int_D dA(x_0) \cdots dA(x_k)$ is the *area-product measure* on Ω_k, with $D \subseteq \Omega_k$
- $\Omega = \bigcup_{k=1}^{\infty} \Omega_k$ is the *path space*. It contains all paths with a finite length.
- $\mu(D) = \sum_{k=1}^{\infty} \mu_k(D \cap \Omega_k)$ is the area-product measure for Ω

3.2 Rendering Equation over Path Space

We will now expand the three-point form of the rendering equation one step. Since the terms will get very long, we will abbreviate $L(a \to b)$, $f_s(a \to b \to c)$ and $G(a \leftrightarrow b)$ with $L(a, b)$, $f_s(a, b, c)$ and $G(a, b)$.

$$
\begin{aligned}
L(x_1 \to x_0) &= L_e(x_1, x_0) + \int_{\mathcal{M}} L(x_2, x_1) f_s(x_2, x_1, x_0) G(x_2, x_1) dA(x_2) \\
&= L_e(x_2, x_1) + \int_{\mathcal{M}} \left(L_e(x_2, x_1) + \int_{\mathcal{M}} L(x_3, x_2) f_s(x_3, x_2, x_1) G(x_3, x_2) dA(x_3) \right) \\
&\qquad \cdot f_s(x_2, x_1, x_0) G(x_2, x_1) dA(x_2) \\
&= L_e(x_1, x_0) + \int_{\mathcal{M}} L_e(x_2, x_1) f_s(x_2, x_1, x_0) G(x_2, x_1) dA(x_2) + \\
&\quad \int_{\mathcal{M}^2} L(x_3, x_2) f_s(x_3, x_2, x_1) G(x_3, x_2) \\
&\qquad \cdot f_s(x_2, x_1, x_0) G(x_2, x_1) dA(x_2) dA(x_3).
\end{aligned}
$$

If we continue to do this and merge the result with the measurement equation 2.15, we can rewrite I^j with x_0 on pixel j as

$$
\begin{aligned}
I^j = \sum_{k=1}^{\infty} \int_{\mathcal{M}^{k+1}} & L_e(x_k \to x_{k-1}) G(x_k \leftrightarrow x_{k-1}) \\
& \cdot \left(\prod_{i=1}^{k-1} f_s(x_{i+1} \to x_i \to x_{i-1}) G(x_i \leftrightarrow x_{i-1}) \right) \\
& \cdot W_e^j(x_1 \to x_0) dA(x_0) \cdots dA(x_k).
\end{aligned}
$$

Since we have removed any L_i and L_o from the equation, we can now solve I^j without any recursion by evaluating all the integrals. The integrand is defined for each path length and gives us the *measurement contribution function*

$$
\begin{aligned}
f_j(x) = & L_e(x_k \to x_{k-1}) G(x_k \leftrightarrow x_{k-1}) \cdot \\
& \left(\prod_{i=1}^{k-1} f_s(x_{i+1} \to x_i \to x_{i-1}) G(x_i \leftrightarrow x_{i-1}) \right) W_e^j(x_1 \to x_0).
\end{aligned} \tag{3.1}
$$

Figure 3.1 shows an example for a transport path $x_0 \dots x_4$ with all relevant factors.

We can now compute I^j with the *path integral* as follows:

$$
I^j = \int_{\Omega} f_j(x) d\mu(x). \tag{3.2}
$$

This formulation allows us to evaluate the color of a pixel without any recursions by integrating over path space. However, as the path space has a theoretically infinite number of dimensions, we apply Monte Carlo integration to solve the path integral.

$$
\int_{\Omega} f_j(x) d\Omega(x) = E\left(\frac{f_j(X)}{p(X)} \right) \approx \frac{1}{N} \sum_{i=1}^{N} \frac{f_j(X_i)}{p(X_i)} \tag{3.3}
$$

where $p(X)$ is a probability density function for sampling a certain path measured over surface areas, and X was sampled according to p.

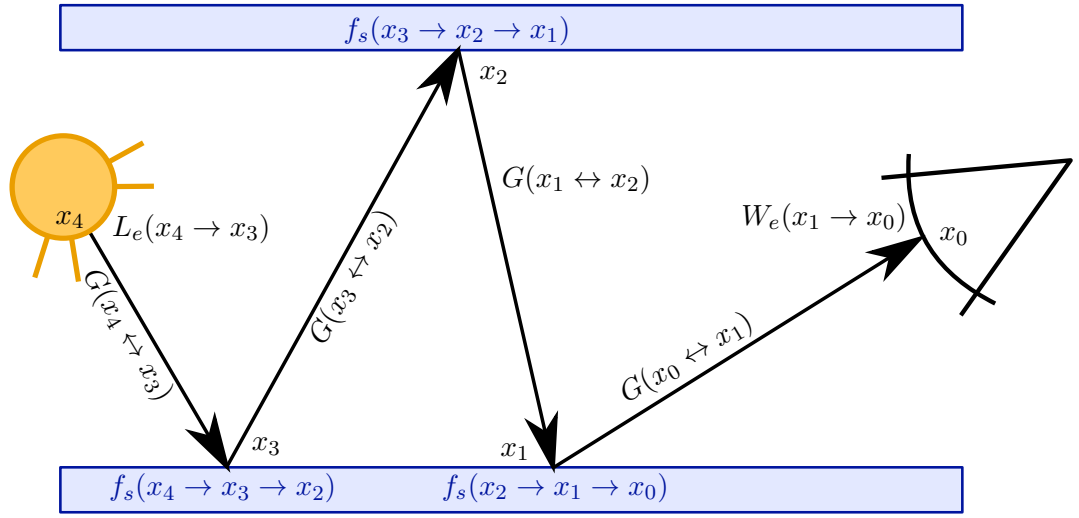

Figure 3.1: A possible light transport path $x_0x_1x_2x_3x_4$ with all factors for the measurement contribution function.

What we have to do next is create N paths of any length, compute their probability and their contribution and weight them together. The next part will cover how to find a good pdf in order to decrease the variance of our estimate. After that we show the basic structure of a path tracing algorithm.

3.3 Importance Sampling Options

Since we build paths by separately creating one point after another, starting from the camera, we can only affect three factors of the total path integral at each point x_i where we have to extend the path. We can directly influence the values of $f_s(x_{i+1} \to x_i \to x_{i-1})$ and the $cos(x_i \to x_{i+1})$ part of $G(x_i \leftrightarrow x_{i+1})$ (at least as long as we don't include the cosine at the next potential path point x_{i+1}) when we sample x_{i+1}. But we can also try to choose path directions that lead us to a a high L_e fast, so that we lose as little energy as possible while traversing the scene.

So instead of one sampling strategy that yields one pdf for the whole path integral, we will create a new pdf at each point x_i locally and only use it once to sample the next point x_{i+1} until we hit a light source or sample a direction that leaves the scene. It is therefore sufficient to simply consider the scattering equation for the current path point x when we want to pick a pdf.
There are several options for choosing pdfs that are roughly proportional to the integrand in the scattering equation 2.16

$$\begin{aligned} L_{o,s}(x,\omega) &= \int_{\mathcal{S}^2} L_i(x,\omega_i) \cdot f_s(\omega_i,x,\omega_o) d\sigma^{\perp}(\omega_i) \\ &= \int_{\mathcal{S}^2} L_i(x,\omega_i) \cdot f_s(\omega_i,x,\omega_o) \cdot cos(w_i) d\sigma(w_i). \end{aligned} \tag{3.4}$$

Obviously, the integrand will get bigger for bigger f_s, L_i and cosine values, although a big cosine can always be cancelled by a small BSDF value or relatively little irradiance (and the other way around). Generally most pdfs try to approximate only one of the factors f_s, L_i or $cos(\omega_i)$.

The next few segments will outline four pdf choices and address their strengths and weaknesses. All options offer certain benefits under certain circumstances, but also show substantial flaws under others. Therefore it is common to combine different strategies (usually BSDF sampling and next event estimation) in order to perform well under any condition.

3.3.1 Cosine to normal

The plainest option is to generate samples proportional to their cosine with the surface normal. Many samples will be roughly perpendicular to the surface and less will follow flat directions.
Unfortunately, as the pdf is basically just a scaled cosine function, it doesn't necessarily approximate the full integrand very well. For example, the more specular the BSDF, the more the value of the integrand depends on ω_o and f_s. Another potential problem is strong but flat radiance with almost no radiance from above, as most samples will be far away from flat directions.
Theoretically, this pdf will produce correct results in any case if enough samples are used, but this may take way longer than the other options proposed below.

3.3.2 BSDF sampling

Next we discuss a pdf that is proportional to the BSDF and will lead to more samples wherever f_s is big. If the probabilities are expressed over solid angle we can apply formulation 3.4 of the scattering equation. This enables us to use Monte Carlo integration to approximate the scattering integral with a pdf $p \propto f_s$ as follows:

$$\int_{\mathcal{S}^2} L_i(x,\omega_i) \cdot f_s(\omega_i, x, \omega_o) \cdot cos(\omega_i) d\sigma(\omega_i) \approx \frac{L_i(x,\omega_i') \cdot f_s(\omega_i', x, \omega_o) \cdot cos(\omega_i')}{p(\omega_i')}, \tag{3.5}$$

where ω_i' is a random variable with density p measured over solid angle.

This strategy can be repeated to approximate $L_i(x,\omega_i')$ in turn until a light source is hit. If BSDF sampling is the only strategy used to create a path, we can get the total energy arriving at the camera along that path like this:

Algorithm 1 basic path tracing with BSDF only

$j \leftarrow 1$
create Camera ray with direction $-\omega_o^j$
$x_j \leftarrow$ first intersection of ray with scene
$pathEnergyWeight \leftarrow W_e^j(x_1 \rightarrow x_0) \cdot G(x_1 \leftrightarrow x_0)/p_A(x_1)$
while x_j not on light source **do**
    $\omega_i^j \leftarrow$ new sample from surface BSDF at x_j
    $pathEnergyWeight \leftarrow pathEnergyWeight \cdot \frac{f_s(\omega_i^j, x_j, \omega_o^j) \cdot cos(\omega_i^j)}{p(\omega_i^j)}$
    $j \leftarrow j + 1$
    $x_j \leftarrow$ first intersection of ray along ω_i^j with scene
    if no intersection exists **then**
        return
    end if
    $\omega_o^j \leftarrow -\omega_i^j$
end while
$k \leftarrow j$
$totalEnergy \leftarrow pathEnergyWeight \cdot L_e(x_k, \omega_o^k)$

Note that we terminate any path in line 11 if a ray along the sampled direction doesn't intersect the scene, as well as we stop each path the first time it hits a light source. Plus, we have to revert the direction in line 13 so we can insert it to f_s when we get to line 7 again.

We will now recalculate what happens in algorithm 1. In order to abbreviate the next equation, we define $G_{0,1} := G(x_0 \leftrightarrow x_1)$ and $W := W_e^j(x_1 \to x_0)$.
Assuming we don't leave the scene and end up with a path $(x_0, \ldots, x_k)$ with x_0 on the camera and x_k on a light source, the total energy arriving at the camera along that path will be

$$\begin{aligned} totalEnergy &= L_e(x_k, \omega_o^k)\frac{G_{0,1}W}{p_A(x_1)} \cdot \prod_{j=1}^{k-1} \frac{f_s(\omega_i^j, x_j, \omega_o^j) \cdot cos_{x_j}(\omega_i^j)}{p(\omega_i^j)} \\ &= L_e(x_k \to x_{k-1})\frac{G_{0,1}W}{p_A(x_1)} \cdot \prod_{i=1}^{k-1} \frac{f_s(x_{i+1} \to x_i \to x_{i-1}) \cdot cos(x_i \to x_{i+1})}{p(x_i \to x_{i+1})} \\ &= L_e(x_k \to x_{k-1})\frac{G_{0,1}W}{p_A(x_1)} \cdot \prod_{i=1}^{k-1} \frac{f_s(x_{i+1} \to x_i \to x_{i-1}) \cdot G(x_i \leftrightarrow x_{i+1})}{p_A(x_{i+1})}. \end{aligned} \tag{3.6}$$

$cos(x \to y)$ represents the cosine of the surface normal at x with the direction of a ray from x through y. p_A is p converted to the surface area measure (see equation 2.18).
The last line is equivalent to the measurement contribution function 3.1 divided by the probability of sampling $x_0 \ldots x_k$. This result can now be added to our estimator 3.3.

Strengths and Weaknesses

Samplig the BSDF works especially well for highly specular surfaces, but shows weaknesses when dealing with small light sources.
Whenever a surface is highly specular, the BSDF strongly determines where the integrand can even reach big values, regardless of the actual irradiance. On the other hand, if a surface is ideally diffuse, the BSDF has no influence whatsoever on the integrand's magnitude. In that case, sampling the BSDF would be no better than generating directions completely at random. Figure 3.2 shows how the roughness of a surface and the size of a light source determine the quality of BSDF sampling.

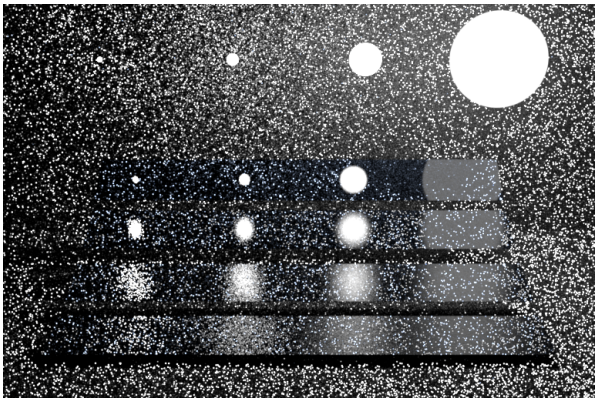

Figure 3.2: On the upper right, where the surface is highly specular and the light sources are large, BSDF sampling performs rather well.
On the lower left as well as the floor and wall, where the surface is highly diffuse and the light sources are small, BSDF sampling produces a lot of noise.

A problem with generating samples solely from the BSDF is that the path is not very likely to hit a light source. This leads to more longer paths and, when the scene is not closed, to paths completely leaving the geometry. The smaller the area covered by light sources the more paths will be leaving the scene before hitting a light source. Figure 3.3 shows how BSDF sampling behaves for different numbers of samples per pixel.

Additional properties of BSDF sampling (as well as next event estimation) can be found in [Vea98].

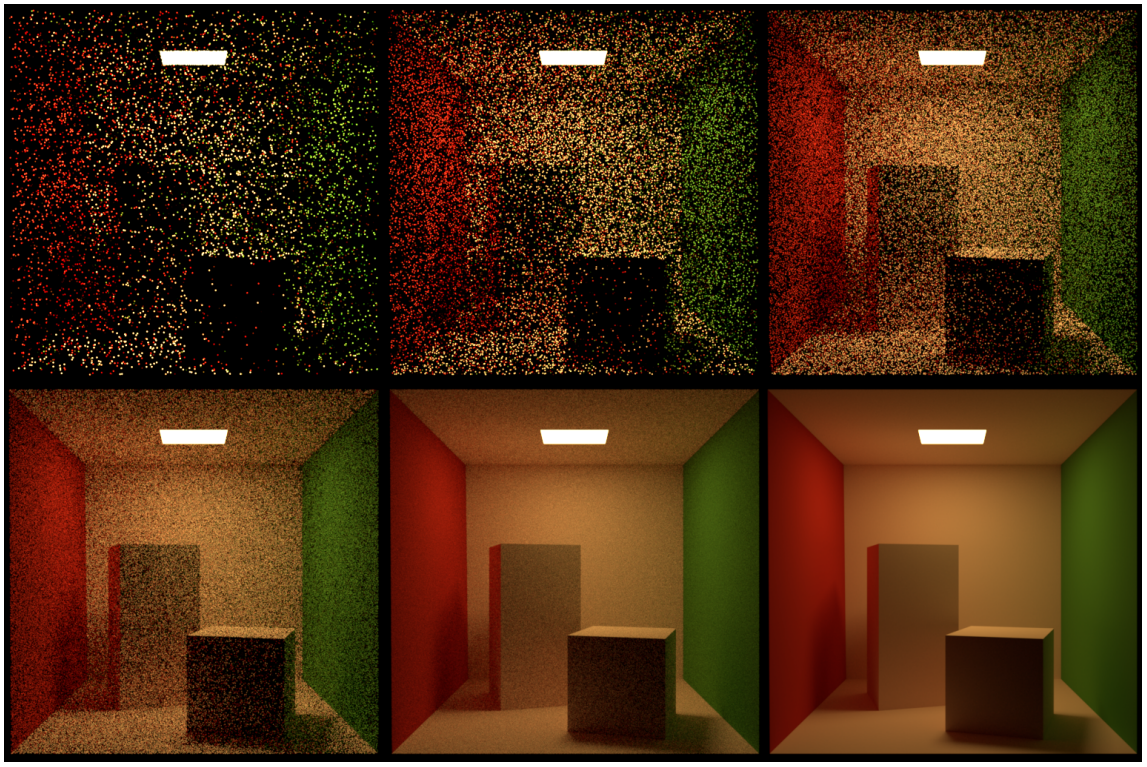

Figure 3.3: BSDF sampling with 1, 4, 16, 64 and 1024 samples per pixel, and a perfect image on the bottom right (created with another algorithm).
Note how even 1024 samples per pixel still produce some noise compared to the exact solution.

3.3.3 Next Event Estimation

As seen above, sampling the BSDF shows some flaws when dealing with diffuse surfaces and small light sources. Next Event Estimation (NEE) is a sampling strategy that performs by far better in these cases, but exhibits some weaknesses for specular surfaces and large light sources.
With NEE, samples are not generated as outgoing directions at the surface point in order to create paths of arbitrary length, but as points on any light source instead: We take an incomplete path $(x_0, \ldots, x_k)$, sample a point x' on a light source and check if x_k and x' are visible to each other. If that's the case we have created a complete path from the camera (x_0) to a light source (x'). Since this strategy always terminates the given path, using it exclusively would result in creating mainly short paths. As a consequence NEE is usually combined with BSDF sampling.

The goal of this strategy is to use a pdf that approximates $L_i(x, \omega_i)$. Since most surface interactions attenuate the energy of incoming light, L_i will presumably be a lot bigger if ω_i points directly to a light source.

In order to better understand how a pdf for NEE can be created, we look at the scattering integral over surface area. Remember the three-point form of the rendering equation 2.19, where we included a geometry term G in order to compensate for the conversion from solid angle to surface area:

$$L_o(x' \to x'') = \int_{\mathcal{M}} L(x \to x') f_s(x \to x' \to x'') G(x \leftrightarrow x') dA(x).$$

Now, intuitively $L(x \to x')$ will be a lot bigger whenever x is actually emitting light instead of just redirecting and thereby attenuating it. As we have no analytical information on L (as opposed to sampling the BSDF or cosine), we use the alternative approximation of giving a high probability to those x on light sources. When combining NEE and BSDF, the multi-sample model even allows for this pdf to be 0 for all points that are not part of a light source.

Usually a light source is picked according to its area and intensity, and a point on that light source is sampled uniformly. These probabilities are initially measured over surface area and have to be converted to solid angle, if we want to combine NEE and BSDF sampling.

Strengths and Weaknesses

As indicated above, NEE performs well for small light sources as well as highly diffuse surfaces. Veach illustrates the different aspects of NEE versus BSDF sampling with the glossy highlights problem in [Vea98, chapter 9.3.1]. We used a replica of his scene from the Mitsuba download website [Jak10]. The results are shown in figure 3.4 for BSDF sampling, NEE and the combination of both using the multi-sample model.
The biggest problem with NEE is that it completely ignores strong indirect lighting (e.g. from mirrors or glass surfaces) and can therefore not render any caustics (see figure 3.5).

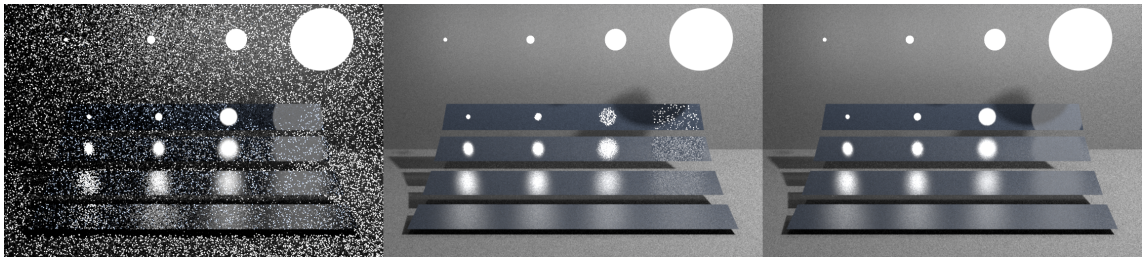

Figure 3.4: Left: BSDF sampling (good with specular surfaces and large light sources)
Center: NEE (good with small light sources or diffuse surfaces)
Right: MIS of BSDF sampling and NEE

3.3.4 Irradiance Sampling

The main part of this thesis will cover caches for incident radiance as an alternative way to create a pdf proportional to L_i. While NEE only considers direct illumination and can be computed as needed, our approach to irradiance sampling requires a preprocessing step. In that step, the incident radiance - direct and indirect - is approximated and cached. Each cache is placed at a surface point x and provides a pdf that is proportional to the incident radiance arriving at x from different directions. The pdf values are measured over solid angle. As shown later, sampling an outgoing direction from such a cache is fast and simple, and - given certain conditions - irradiance importance sampling can offer advantages over NEE or BSDF sampling.

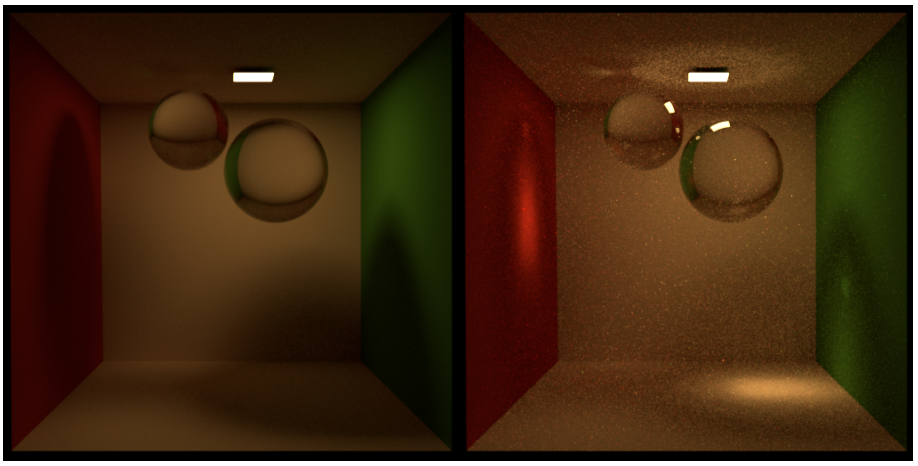

Figure 3.5: Left: NEE only, the caustics are missing. Right: Reference with caustics.

In chapter 4 we show how we used caches to approximate incident radiance, and chapter 5 will explain how to use these caches for irradiance importance sampling.

3.4 Path contribution

Assuming that we have created a path $x = x_0 x_1 \dots x_k$ according to some strategy, we can now compute its contribution to our estimator from equation 3.3 as

$$contribution(x) = \frac{f_j(x)}{p(x)}.$$

The total probability of sampling the whole path is given as the product of all probabilities used to sample a new point of the path. We only use forward sampling, meaning that we only sample new points on a path from camera to light source. Note that bidirectional techniques use both forward and backward sampling - subpaths are sampled starting both from a light source and from the camera, which affects the way their probabilities have to be computed. See [Vea98, chapter 10] for further information.

Note that equation 3.2 is measured by the area-product measure. So theoretically we have to measure our probabilities over surface area. Let $p_A(x_{i+1})$ be the probability of sampling (the direction to) x_{i+1} when we are at point x_i, measured over surface area, and let $p_\sigma(x_i \to x_{i+1})$ be the probability of sampling x_{i+1} from x_i measured over solid angle. We can convert between p_A and p_σ similar to equation 2.18 as follows:

$$p_A(x_{i+1}) \cdot cos(x_i \to x_{i+1}) = G(x_i \leftrightarrow x_{i+1}) p_\sigma(x_i \to x_{i+1}), \tag{3.7}$$

In the next equation we will replace $f_s(a \to b \to c)$ with $f_s(a, b, c)$ and $G(a \leftrightarrow b)$ with $G(a, b)$, and we will use

$$W := \frac{W_e^j(x_1 \to x_0) G(x_0, x_1)}{p_A(x_1)}.$$

The total contribution of a path $x = x_0 x_1 \dots x_k$ from camera to light can be computed by combining p and f_j.

$$\begin{aligned} contribution(x) &= L_e(x_k \to x_{k-1}) \cdot W \cdot \prod_{i=1}^{k-1} \frac{f_s(x_{i+1}, x_i, x_{i-1}) G(x_i, x_{i+1})}{p_A(x_{i+1})} \\ &= L_e(x_k \to x_{k-1}) \cdot W \cdot \prod_{i=1}^{k-1} \frac{f_s(x_{i+1}, x_i, x_{i-1}) cos(x_i \to x_{i+1})}{p_\sigma(x_i \to x_{i+1})}. \end{aligned} \tag{3.8}$$

Equation 3.8 shows that the geometry term disappears almost completely if we measure our pdfs over solid angle. Fortunately this is also way more convenient for BSDF sampling and our caches, as these strategies only provide directions and don't make use of any information on the other surfaces in the scene - the next path point is only retrieved by casting a ray from the current path point along the sampled direction.

This also means that we can assemble the path's contribution gradually while we create it, as depicted in algorithm 2: At each point x_i we evaluate the local BSDF for the previous and sampled direction and the cosine factor for the sampled direction. We multiply these values together, and as soon as we hit a light source we multiply L_e and are done.

Algorithm 2 Assemble a path's energy

$weight \leftarrow W$
while x_i not on light source **do**
  Sample x_{i+1}
  $weight \leftarrow weight \cdot \frac{f_s(x_{i+1}, x_i, x_{i-1}) cos(x_i \to x_{i+1})}{p_\sigma(x_i \to x_{i+1})}$
end while
return $L_e(x_k \to x_{k-1}) \cdot weight$

Note that for NEE, where we sample a point instead of a direction, a pdf measured over surface area is more convenient. So if we hit the light source deliberately by using NEE, we need to replace the last $\frac{cos(x_{k-1} \to x_k)}{p_\sigma(x_{k-1} \to x_k)}$ with $\frac{G(x_{k-1} \leftrightarrow x_k)}{p_A(x_k)}$.

3.5 The Algorithm

Algorithm 1 already provided a short example of a basic path tracing algorithm that only used the BSDF to extend paths.

We will now expand that example to include BSDF sampling as well as NEE and add MIS weights according to the multi-sample model. We ignored $W_e^j(x_1 \leftrightarrow x_0)$, $p_A(x_1)$ and $G(x_1 \leftrightarrow x_0)$ in our implementation, since W_e^j is usually defined in a way that all three factors are cancelled out to prevent vignetting.

We use the following notation:

- $p_{A,nee}(x')$ is the pdf value for sampling a point x' on any light source, measured over surface area. $p_{\sigma,nee}$ is the same pdf measured over solid angle.
- Given points x_1, x_2 and x_3, $p_{BSDF}(x_3)$ is the probability of sampling the incident direction $x_3 - x_2$ from the local BSDF at x_2 with outgoing direction $x_1 - x_2$.

We use the power heuristic 2.10 to weight NEE and BSDF sampling together. A similar algorithm can be found in [vA11].

Algorithm 3 Path tracing with BSDF sampling and NEE

$x_0 \leftarrow$ point on Camera
$\omega_1 \leftarrow$ first ray direction
$x_1 \leftarrow x_{\mathcal{M}}(x_0, \omega_1)$
$w_{BSDF} \leftarrow 1$
$i \leftarrow 1$
$throughput \leftarrow 1$
$color \leftarrow black$
while true **do**
    if $L_e(x_i \rightarrow x_{i-1}) > 0$ **then**
        return $throughput \cdot w_{BSDF} \cdot L_e(x_i \rightarrow x_{i-1}) + color$
    else

        $x' \leftarrow samplePointOnLightSource()$
        if x' is visible from x_i **then**
            $w_{nee} \leftarrow \frac{p_{\sigma,nee}(x')^2}{p_{\sigma,nee}(x')^2 + p_{BSDF}(x')^2}$
            $L \leftarrow L_e(x' \rightarrow x_i)$
            $color \leftarrow color + L \cdot throughput \cdot w_{nee} \cdot f_s(x' \rightarrow x_i \rightarrow x_{i-1}) \cdot \frac{G(x_i \leftrightarrow x')}{p_{A,nee}(x')}$
        end if

        $\omega_{i+1} \leftarrow$ bsdf.sampleDirection(x_i, ω_i)
        if $ray(x_i, \omega_{i+1})$ intersects scene **then**
            $x_{i+1} \leftarrow x_{\mathcal{M}}(x_i, \omega_{i+1})$
            $throughput \leftarrow throughput \cdot f_s(x_{i+1} \rightarrow x_i \rightarrow x_{i-1}) \cdot \frac{cos(x_i \rightarrow x_{i+1})}{p_{BSDF}(x_{i+1})}$
            $w_{BSDF} \leftarrow \frac{p_{BSDF}(x_{i+1})^2}{p_{BSDF}(x_{i+1})^2 + p_{\sigma,nee}(x_{i+1})^2}$
        else
            return $color$
        end if
        $i \leftarrow i + 1$
    end if
end while

This algorithm usually creates multiple paths: One so-called "implicit path", where we continue to sample points from the BSDF until we accidentally hit a light source, and several "explicit paths" that are created by deliberately sampling a point on the light source. However, since we terminate any path as soon as it hits a light source, we always extend exactly one path. The explicit paths are terminated as soon as they are created and pose no problem in terms of an exponentially growing number of paths to pursue.

3.6 Russian Roulette

In an enclosed environment, paths generated by BSDF sampling can get arbitrarily long before reaching a light source. Not only do these paths carry very little energy, they also take much more time to compute. Russian Roulette is a common solution to achieve better computing time while not manipulating the correct solution by ignoring these paths completely.

With Russian Roulette, all paths are randomly terminated after reaching a certain length: Every time before the path is extended, a random number r is compared to a threshold $\alpha \in [0, 1]$. If $r > \alpha$, the path is terminated. If $r \leq \alpha$, the path is continued and weighed with $\frac{1}{\alpha}$. As a result, there will be fewer but more heavily weighted long paths and more short paths with smaller weights. As this approach still explores all possible paths it still generates a correct solution.

Too small α values can lead to an extremely low number of long paths, that in exchange are extremely highly weighted and result in fireflies in the final image. On the other hand, Russian Roulette will rarely cancel any paths at all, if α is constantly too big. Usually α is determined dynamically, depending on criteria such as the path's current length, its contribution or surface properties (e.g. reflectance).
Since very short paths will almost always be important to a pixel's color, Russian Roulette should only be applied to paths above a certain length.

However, Russian Roulette will always lead to a higher variance, so we only applied it to paths with a length over 20.

4. Caching incident radiance

The main goal of this thesis is to approximate the irradiance across a scene using local caches. Opposed to irradiance caching (as described in chapter 6.4, [WRC88]), where all the irradiance at one point is cached in one single value, we want to cache the incident radiance for different directions.
This chapter will cover all relevant preprocessing steps: Choosing a cache representation, placing the caches in the scene, and filling them. The next chapter will cover the actual rendering process and describe how these caches can be used in a path tracer to importance sample the irradiance.
We will use the term **IRC** (incident radiance cache) to refer to a cache.

4.1 Environment Map Parameterization

Every IRC at a certain surface point is represented by a small local environment map. Every texel of this environment map will contain the relative incident radiance around the cache's position from the direction covered by that texel.

We chose the Octahedron representation proposed by [ED08]. The main advantage of this parameterization is that in comparison to cube maps and especially sphere maps, all texels cover roughly the same solid angle. Since all probabilities in our path tracing algorithm are measured over solid angles, this choice results in the least distortion when converting probabilities between texel space and solid angles. This conversion is explained in chapter 4.2.3.2.

Plus, the conversions between directions and texels are rather simple:
To map a direction $d = (d_x, d_y, d_z)$ on a texel coordinate $p = (p_x, p_y)$, we first use planar projection to convert the direction vector to a point on the unit octahedron surface:

$$d' = \frac{d}{|d_x| + |d_y| + |d_z|}. \tag{4.1}$$

Next we need to project this point on a two-dimensional texture coordinate p. There are several ways to do this, we chose one that yields a rectangular texture:

$$p = \begin{cases} (d'_x - d'_z - 1, d'_x + d'_z) & d'_y \geq 0 \\ (d'_z - d'_x + 1, d'_x + d'_z) & d'_y < 0. \end{cases} \tag{4.2}$$

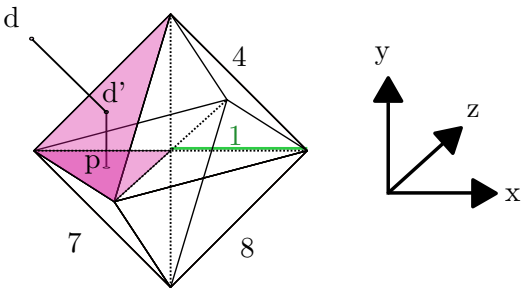

Figure 4.1: Planar Projection

Figure 4.1 depicts the complete projection. The resulting coordinate lies in $[-2,2]\times[-1,1]$, with directions of the upper hemisphere being mapped to $[-2,0]\times[-1,1]$ and directions of the lower hemisphere being mapped to $[0,2]\times[-1,1]$. As seen in figure 4.2, texels close to $p_x = 0$ belong to similar directions on adjoint octahedron surfaces.

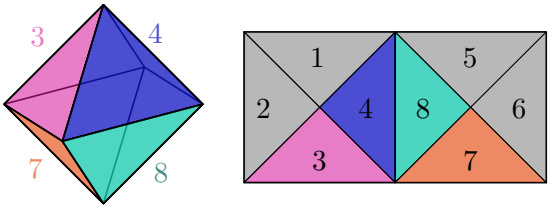

Figure 4.2: The upper hemisphere is mapped to the left side, while the lower hemisphere (if needed) is mapped to the right half.
The borders in the final map match the octahedron edges.

This projection is relatively easy to calculate back and forth, plus the resulting texels cover the whole area of the texture. Since both hemispheres are mapped to separate parts of the texture we can simply ignore the lower hemisphere at non-transmitting surfaces, when irradiance from below has no contribution to the outgoing light. We can then only use radiance arriving at the upper hemisphere by only using - and storing - the left half of the environment map, which also saves a considerable amount of storage space.

4.2 Creating Caches

Actually creating these caches consists of 4 steps: Photon Mapping, distributing the positions of the caches over the scene, filling the caches and storing them.
Since we intend to fill the caches with incident radiance, we need some way to approximate it. We do this with a modified version of photon mapping. When the photon mapping is done, the photons are used to place and fill the caches.

4.2.1 Modified Photon Mapping

The idea of photon mapping is to trace a huge number of emitted photons from all light sources along their paths and store a photon at every surface point along that path. Originally, the resulting photon map would then be used to render the image. We give a short introduction on photon mapping as a stand-alone rendering technique in chapter 6.1.

Creating a photon path

A photon path is initialized by sampling a point and outgoing direction for a photon and then tracking it across the scene. Whenever there are multiple light sources, a light source L is chosen with a probability p_L proportional to its surface area A_L and its flux Φ_L:

$$p_L = \frac{A_L \cdot \Phi_L}{\sum_{L'} A_{L'} \cdot \Phi_{L'}}. \tag{4.3}$$

For simplicity's sake we only consider area light sources here. A point x on light source L is sampled uniformly with probability $p_x = 1/A_L$, the probability for sampling direction ω_o at this point is p_ω. Given a light source L, let $L_e(x, \omega_o)$ be the power of that light source for the given origin x and direction ω_o. When N photons are emitted, the initial energy of the photon emitted from light source L at x into ω_o is given as

$$\frac{L_e(x, \omega_o)}{p_L \cdot p_x \cdot p_\omega \cdot N} = \frac{L_e(x, \omega_o)}{p_\omega} \cdot \frac{\sum_{L'} A_{L'} \Phi_{L'}}{\Phi_L \cdot N}. \tag{4.4}$$

This photon is then traced across the scene. Whenever the photon's path intersects a diffuse surface, a photon with the current energy is stored. At any intersection y, no matter if it's diffuse or specular, the outgoing direction ω_o of the photon path is sampled from the surface's adjoint (see 2.2.4) BSDF and the incoming direction ω_i. The path is continued into the sampled direction and the energy is weighed with the BSDF's value $f_s(\omega_i, y, \omega_o)$ divided by the probability of sampling ω_o.

Adjustments for IRCs

Since we plan to use photon mapping to approximate incident radiance instead of rendering an image we need to make some adjustments to this process. One of them are the stored photons themselves: Each of our photons consists of its position including the local surface coordinate system, its incoming direction, a pointer to the surface's BSDF and its energy. The next section will explain why the local coordinate system is necessary. The BSDF is needed in case we want to place a cache at the photon's position: It tells us whether we need an environment map for both hemispheres or only one. We don't need the actual energy distribution over different wavelengths; storing a photon's average energy in a single float value is sufficient.

Besides, we only need the relative incident radiance from all directions to create a cache. This allows us to ignore the constant factors in 4.4 (the fraction on the right) when computing a photon's initial energy; we can use

$$\frac{L_e(x, \omega_o)}{p_\omega} \tag{4.5}$$

instead. To avoid unnecessary computing, we cancel photon paths when their accumulated energy falls too close to zero, when the path length reaches 32 and using Russian Roulette. As soon as all photon paths are complete, all created photons are stored in a k-d tree ([Ben75], [nan14]).

4.2.2 Placing Caches

There are two interesting areas for placing IRCs: Those with much incident light, and those that are potentially hit very often when using path tracing later on.

To cover the latter, we send a camera ray through the upper left corner in every 8×8 pixel block. If the BSDF at its first intersection with the scene is smooth, we place a cache at that position. Otherwise the ray will be traced until it leaves the scene or until a surface with a smooth BSDF is hit so we can place a cache. We call the resulting caches *camera caches*.

The remaining cache positions are adopted from the first n photon positions from the photon mapping step. This will lead to many caches in areas where many photons arrive, and place less caches in areas where only few photons intersect the scene. Since the photon paths were constructed randomly, the distribution of cache positions generated from simply taking the first n photons will be similar (on average) to choosing the photons randomly.

Note that from now on we have almost no information concerning the surface properties (BSDF, geometry, normals) at the cache position. We only know the positions themselves and that they are placed at non-delta BSDFs. To avoid inconsistent coordinates later on, it is necessary to give each cache its local coordinate system. Luckily we already stored these systems with every photon and can simply use these for the photon caches. The camera caches get their coordinate system from the intersection information which is still available when the caches are placed in the scene.

At this point we also need to decide whether to create a cache for irradiance from both hemispheres or only one. To do so, we look at the BSDF (which is either stored in the photon or can be retrieved from the intersection information for camera caches): If it is transmitting, we create an empty environment map for both hemispheres and store it at the position with its coordinate system. Otherwise a map for the upper hemisphere is created and stored. Practically the only difference between these two IRCs is the length of the array containing the environment map's texels. We don't consider materials where only irradiance from below contributes to radiance leaving at the upper side.

4.2.3 Filling a Cache

After the caches are created and placed across the scene we need to fill them with an approximation of the incident radiance around them. The number k of photons used to fill a cache is fix and can be adjusted before the image is rendered. Filling a cache works like this:

$photonlist \leftarrow kdTreePhotons.getKClosestPhotons(k, cache.position)$
for all *Photon* p in $photonlist$ **do**
  $incoming\_direction\_local \leftarrow cache.transformToLocal(p.direction)$
  if $cos(local\_incoming\_direction) > 0$ || $cache\_for\_both\_hemispheres$ **then**
    $texel \leftarrow cache.convertDirectionToTexel(incoming\_direction\_local)$
    $cache.environmentMap[texel] += p.energy$
  end if
end for

Whenever a photon is deemed useful to fill a cache, its incoming direction is converted to a texture coordinate for our octahedron environment map (see equations 4.1 and 4.1).

We found that filtering the caches considerably reduces the noise in the final image. First we apply bilinear filtering to distribute the energy of each photon over the texel itself and its left/right, upper/lower and diagonal neighbour. This is done while the photons are added to the cache one by one.

After the cache has been filled, we use a Gaussian filter

$$\frac{1}{16}\begin{pmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{pmatrix}$$

on all texels. In order to filter the texels on the edge, we simply added the texels from the opposite side. Theoretically, this is not correct, as texels on opposing edges may correspond to opposing directions (see figure 4.2). But the improvement on the images was big enough, so we stayed with that simplification.
We found that Gaussian filters showed a bigger improvement than bilinear filtering, and the combination of both could noticeably reduce the noise of an image, see figure 4.3.

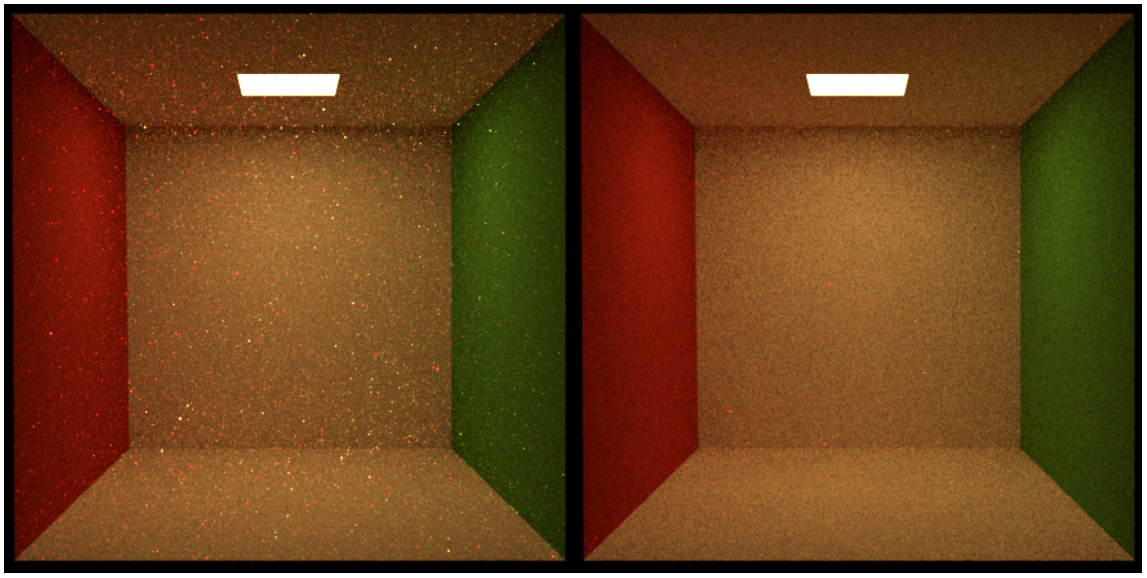

Figure 4.3: Left: No filtering.
Right: bilinear filtering and Gaussian filter.
Both images were rendered with IRCs only (no BSDF sampling or NEE), using 150.000 photon paths ($\approx$ 750.000 photons), 20.000 caches, 500 photons per cache and 64 samples per pixel.

We also tried to weight the energy contribution of a photon with its distance to the cache's position or with the Epanechnikov kernel [Suy02, chapter 8]. The difference was barely noticeable, but the images seemed a little better without these weights.

One last thing to note is that the number of photons actually contributing to a cache may be lower than k. Figure 4.4 illustrates an example.

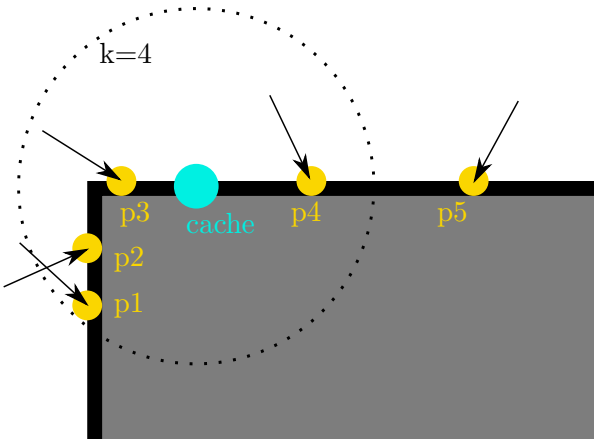

Figure 4.4: When filling the cache with 4 photons, photons p1 to p4 are considered and p5 is ignored. However, the incoming direction of p2 results in a negative cosine with the cache's surface normal, meaning that the photon arrives from below from the cache's point of view. Thus p2 is discarded for this cache and only p1, p3 and p4 are used to fill it.

4.2.3.1 Creating a useful probability density function

The next step is to convert the filled environment map to a probability density function that can be used to sample an outgoing direction proportional to the incident radiance during path tracing. The main idea is to normalize the sum of all texels of the environment map to 1 and then sample directions according to the probability values stored in the texels. Obviously, the first step towards a useful pdf is simply that: add up all energy values from the environment map and divide every texel's value by that sum.

However, we have to consider that Monte Carlo integration requires the pdf to be greater than zero wherever the integrated function contributes to the integral. So we have to make sure that our pdf isn't equal to 0 when radiance arriving from the corresponding direction can contribute to the integral. With non-delta BSDFs that is the case for all directions (in the upper hemisphere), so we have to set all texels with value 0 to a bigger value. This value should be big enough so it won't cause any floating point precision problems - either by being ignored completely in the final pdf or by causing fireflies due to dividing by very small numbers. On the other hand the value should not affect the quality of the radiance approximation stored in non-zero texels.

We tried several options: The first one was setting the zeros to the smallest positive value from all other texels before normalizing. However, in many cases this value turned out to be too close to zero and disappeared completely after the normalization. Even normalizing, setting all 0 texels to the smallest value after normalizing, and normalizing again resulted in near-zero values that didn't work well in the final image.
Next we tried setting every value below half average of all non-zero values to half average. The reason for that idea was that many caches had some texels with values around some 100, but a lot more texels containing positive values below 1. This approach caused by far the most noise in the final image.
After looking at several normalized environment maps, we tried to simply set all values smaller than 0.0002 to 0.0002 and normalize again. This seemed to be working quite well so far. We even tried combining this approach with the first one, but as it turned out the minimal value was always smaller than 0.0002, regardless of whether it was picked before

or after the first normalization. Different thresholds from 0.0001 to 0.001 didn't produce any noticeable differences in the final images.

There's also a reason not to set every 0 to a fix non-zero value before the first normalization: The sums of two caches can easily differ in an order of 2 magnitudes. So while setting all 0 texels to, say, 1 may work perfectly fine for one cache with a (previous) sum of some 100, it can totally destroy the irradiance approximation for another one with a sum of 50 or just disappear again for caches in bright regions with sums up to some 1000. Additionally, the initial sum of a cache depends on the number of photons used to fill it and the overall energy of these photons, while the average value of a texel also depends on the resolution of the octahedron map. Considering all these uncertainties it seems more reasonable to increase the 0-texels after a first normalization, when we can make a more precise statement about the overall properties of the cache.

The values of the environment map now form a valid probability density function that can be used to sample directions. This pdf is currently measured over the texel space, where each texel covers a unit square and all texel values sum up to 1.
To improve computation time we also create a second environment map (the cdf_map) containing the cumulative density function for that pdf and added it to the cache. See chapter 5.1.1 on how samples are torn from these environment maps.

4.2.3.2 Convert probabilities to solid angle

Generating samples is not the only purpose of these caches. The other one is to get the probability linked to the sampled direction, and to compute the probability of sampling a given direction from a cache. These probabilities are needed to compute weights for Multiple Importance Sampling and to compute the probability of sampling a path.
While probabilities for path tracing are usually measured over solid angle, our pdf is measured over a rectangle (or square in case only the upper hemisphere is important). Its width and height correspond to the number of columns and rows of the environment map, and each texel covers the area of one unit square. Remember that the pdf values were normalized to sum up to one.

Unfortunately, there is no easy way to analytically compute the solid angle covered by each unit square in our environment map. Ignoring this problem and directly using the probability value from a cache resulted in almost completely white images. This is due to the fact that with a 16×16 resolution, every pdf value is roughly $\frac{2\pi}{256}$ times as big as it should be. Thus the accumulated probability of sampling a certain path is several orders of magnitude smaller than the correct value, and dividing by it causes the energy to reach insanely high values.
Approximating the solid angle measure by multiplying with $\frac{size\_of\_environmentmap}{2\pi}$ isn't enough either: The overall energy of the resulting images seemed about right, but it was rather weirdly distributed (see figure 4.7). So we created another map to approximate the solid angle covered by each texel:

solid_angle_map ←new *octahedronMap* ▷ upper hemisphere only
numSamples ← 1000 · *solid_angle_map.width* · *solid_angle_map.height*
for $i = 0$ to *numSamples* **do**
    Vector3 *sample* ← *createRandomSampleOnHemisphere*()
    Vector2 *texel* ←*solid_angle_map.convertDirectionToTextureCoordinate*(*sample*)
    index ← ⌊*texel.x*⌋ + *solid_angle_map.width**⌊*texel.y*⌋
    solid_angle_map[*index*] ← *solid_angle_map*[*index*] + 1
end for
for ($i = 0$ to *solid_angle_map.length*) **do**
    solid_angle_map[*i*] / = *numSamples* ▷ normalize to 1
    solid_angle_map[*i*] * = 2π ▷ integral over one hemisphere is 2π
end for

At the end every texel represents the relative solid angle covered by its area. Central texels and those close to the border represent directions close to the surface's normal and along the surface itself respectively and will hold smaller values. The areas in the middle of each octahedron surface are close to parallel to the unit hemisphere and thus cover a bigger solid angle. Figure 4.5 shows the resulting maps for 4×4, 8×8 and 16×16 texels, and figure 4.6 depicts the relation between texel area and solid angle.

A pdf value from a texel in a cache's environment map can now be converted to a probability value measured over solid angle by dividing it by the value from the corresponding texel in the solid angle map. This can be done for all texels in all environment maps in a preprocessing step, as the number of accesses to caches during the actual rendering may easily exceed the total number of texels.
The solid angle map can also be doubled and then applied to the lower hemisphere. In that case its values will sum up to 4π, which equals the integral over the unit sphere.

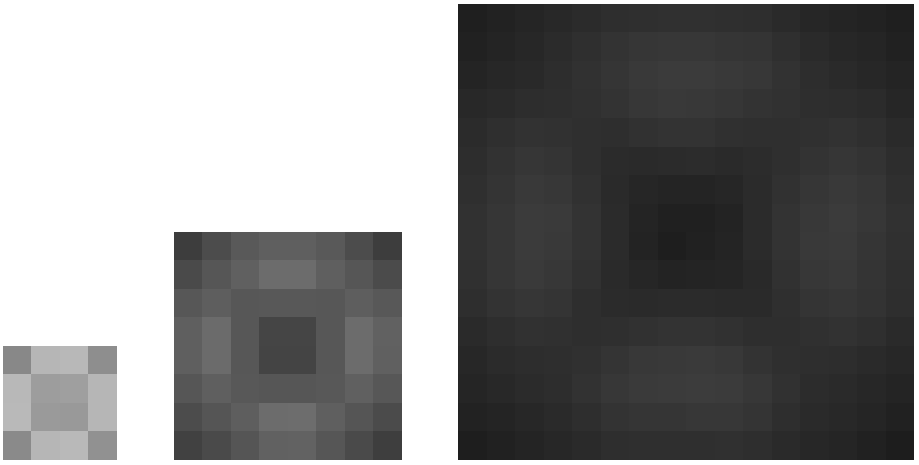

Figure 4.5: Solid angle maps with 4×4, 8×8 and 16×16 texels for one hemisphere

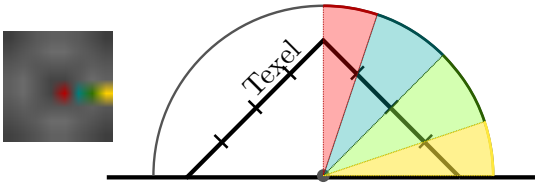

Figure 4.6: Solid angle areas covered by texels of different height for a 8x8 map. Texels on the octahedron surface's center (blue, green) cover a larger solid angle than those close to the surface (yellow) or at the top (red).
Note that the difference between the texel values also depends on their horizontal distribution.

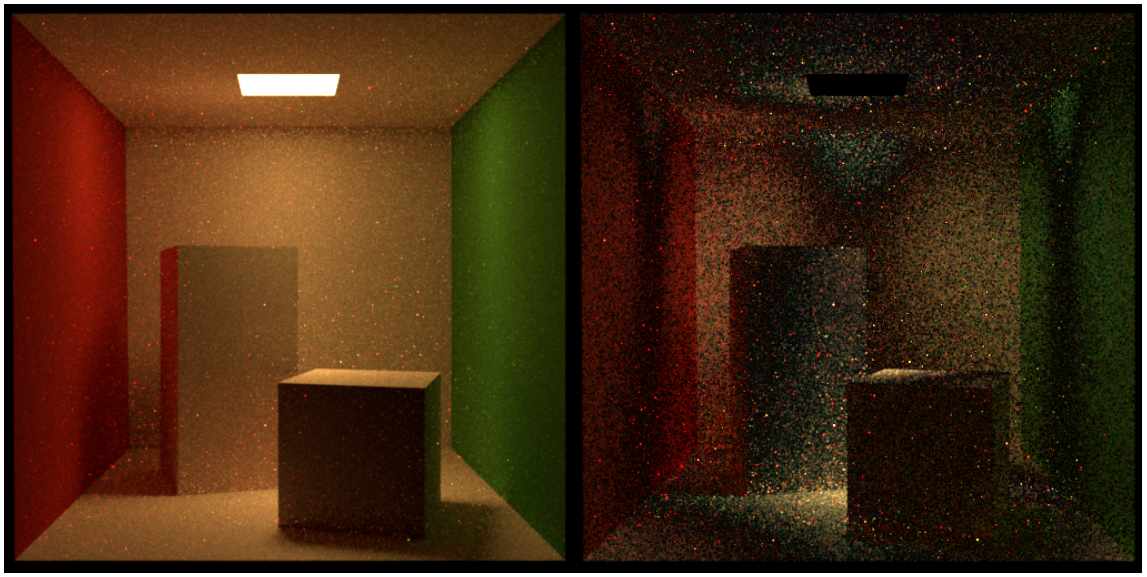

Figure 4.7: Left: Solid angle correction with $\frac{width \cdot height}{2\pi}$
Right: Difference with correct reference, multiplied by 4. Note the vertical stripes on the wall in both images and the bright area in the middle of the floor.

4.2.4 Storing Cache

All caches are stored in a k-d tree ([Ben75], [nan14]) for fast and easy access. The k-d tree containing the photons won't be needed again and can be deleted. In the end, each IRC consists of the following components:

- An octahedron environment map containing the probability density function, measured over solid angle, that approximates the incident radiance around the cache. We will call this one the *pdf_map*.
- An additional map containing the cumulative density function for faster sampling, called *cdf_map*. Note that the cdf is created after the last normalization, but before the pdf is adapted to the solid angle measure.
- Its position.
- Its local coordinate system.

5. Rendering

This chapter will describe how the IRCs from the previous chapter can be used to improve path tracing. We will use the terms "'BSDF sampling"' and "IRC sampling"' for the process of sampling an incident direction and its probability from a surface point's BSDF or corresponding IRC respectively. How samples can be generated from a BSDF will not be discussed here any further.
The first part of this chapter will cover generating samples and probabilities from an IRC. The second part combines IRC sampling with BSDF sampling or NEE using Multiple Importance Sampling. In the end we evaluate the results.

5.1 Sampling Caches

5.1.1 Path Extension

As described in chapter 3, path tracing creates random paths starting at the camera and traces these paths across the scene until either a light source is hit or the path leaves the geometry. To do that, we need to choose a new direction every time the path hits a surface at an intersection point x. This is normally done by sampling an outgoing direction from the surface's BSDF. In that case the applied pdf is proportional to the BSDF term from the rendering equation. The goal of our IRCs is to use a pdf which approximates the incident radiance $L_i(\omega_i, x)$ instead of the BSDF.

Our basic algorithm for sampling a new outgoing direction proportional to the irradiance consists of several steps: Picking an appropriate cache, sampling a direction and computing its probability.
So, first we have to choose the cache we want to use. Sometimes, especially at corners, thin walls or curved surfaces, the closest cache can have another normal than the surface at x itself. If the deviation is small enough, the cache can be used anyway. However, if the cache's rotation is too severe, it might be better to pick one of the next closest caches more parallel to the surface at x. We chose 0.9 to be the threshold for the cosine between the cache's and surface's normal and never considered more than the 5 closest caches.
This may seem arbitrary, but in our example scenes most caches violating that threshold had a cosine of 0 or less to the surface. Plus whenever a cosine was positive but below 0.9, some of the other closest caches often exhibited bigger values. We also didn't experience any problems with spheres, since the caches were always placed densely enough.

With more complex scenes and small spheres in rarely lit areas that are not seen by the camera directly but hit many times for indirect lighting, one would have to test if 0.9 is

still a good choice or if a smaller values would be a better option. On the other hand, if only few photons arrive in a region in the first place, it might prove more useful to just use normal BSDF sampling there.

Instead of always choosing the closest cache (or that one of the 5 closest ones with the best cosine), we decided to randomly pick one of those caches that was among the 5 closest to x and also had the best cosine value. In most cases these were all 5 closest caches with a cosine of 1. This process is depicted in algorithm 4.

Algorithm 4 Pick Cache

```
1:  procedure PICKCACHE(surfacePoint)
2:      cacheList ← kdTreeCaches.getKClosestCaches(5, surfacePoint.position)
3:      bestCos ← −1
4:      for i = 0 to 4 do
5:          currentCache ← cacheList[i]
6:          currentCos ← cos(currentCache.normal, surfacePoint.normal)
7:          if currentCos > bestCos then
8:              bestCos ← currentCos
9:              bestCache ← currentCache
10:         else if currentCos == bestCos ∧ getRandomNumber() > 0.6 then
11:             bestCache ← currentCache
12:         else
13:         end if
14:     end for
15:     if bestCos > 0.9 then
16:         return bestCache
17:     end if
18:     return null                                   ▷ sample BSDF instead
19: end procedure
```

Line 7 ensures that we only use caches with the highest cosine value. Line 10 takes care that we pick one of the best caches at randon, with closer caches having a higher probability of being chosen. If all five closest caches are rotated too much towards the surface at x, we resort to BSDF sampling just this once.

As it turns out, choosing between the 5 closest caches instead of just using the closest cache (if its cosine is high enough) consumes a third of the total computation time 5.4.3. Plus we could not see any difference in the resulting images as long as enough IRCs were available. Maybe picking one of 5 caches is more useful for more complex scenes than our simple examples, where there is more curved geometry or edges.

After a cache is chosen, the next step is sampling a direction from that cache. For that purpose three random numbers between 0 and 1 are required. The first one is used to pick a texel from the cache's environment map. To do that we look for the field in the cache's *cdf_map* (see chapter 4.2.4) with a value bigger or equal to the random number. The other two are needed to sample a point within the texel. This point is converted to a direction (reverse equations 4.1 and 4.2).

As we always need the probability of the sampled direction whenever a direction is sampled, algorithm 5 also returns the probability for sampling the direction. Note that while the cdf was created from a pdf that sums up to one, the returned probability is measured over solid angle and was computed after the cdf was created.

Algorithm 5 Sample Direction from Cache

```
1: procedure SAMPLEDIRECTION(cache, pdfValue, rand1, rand2, rand3)
2:     texel_index ← BINARYSEARCH(cache.cdf, rand1)
3:     pdfValue ← cache.pdf_map[texel_index]
4:     texel.x ← texel_index%cache.mapWidth + rand2
5:     texel.y ← texel_index/cache.mapWidth + rand3
6:     Vector3D direction ← cache.texelToDirection(texel)
7:     direction ← cache.coordinateSystem.toWorld(direction)
8:     return direction
9: end procedure
```

We use binary search to find the index of the sampled texel in our *cdf_map* more efficiently. This index yields the probability for its texel from the *pdf_map*, we write that probability to $pdfValue$ so it can be used later on.
Note that the pdf is designed for texels with an area of 1×1, so we can get a random point within that texel from lines 4 and 5. Line 6 converts this point to a point in $[-2, 2] \times [-1, 1]$ (or $[-2, 0] \times [-1, 1]$, see chapter 4.1) and then to a direction in local coordinates (the upper axis matches the top of the octahedron), which is converted to a direction in global coordinates in line 7.

After a (global) direction ω is sampled, we can continue the path along ω. The last thing left to do is to weight the energy carried along the path. Therefor we have to evaluate the BSDF at x with ω as incoming (!) direction ω_i and appoint the direction where the path came from as outgoing direction ω_o. The energy carried along the path can be weighted just as in algorithm 2 according to equation 3.8 as follows:

Algorithm 6 Weight path energy for Monte Carlo integration

```
bsdfValue ← x.bsdf.eval(ω_i, x, ω_o)
cos ← cos(ω_i, x.normal)
pathEnergy ← pathEnergy · bsdfValue · cos/pdfValue
```

Note that the BSDF technically evaluates directions represented in local coordinates at x, so we would actually have to convert the directions from world to local coordinates first. Fortunately, if the cosine of the used cache happens to be 1, we can skip the transformation of ω_i and just use the local cache coordinates from line 6 in algorithm 4 instead, as long as the rendering framework can guarantee consistency of both coordinate systems.

However, there are cases when IRC sampling is not a good option: At specular surfaces, only light from one direction (or two for transmitting materials) actually contributes to the integral to compute. It is therefore unnecessary to randomly sample several directions, we can directly sample the one (or 2) important direction from the surface's BSDF. Note that in chapter 4.2.2 we only placed caches at non-specular surfaces: The cache positions were created either from photons (which are only stored for non-delta BSDFs) or from camera rays (which were traced across the scene until a surface with a non-delta BSDF was found).
It might also happen that there are no caches nearby, or that the closest caches are rotated too much to provide useful information. In these cases we fall back on conventional BSDF sampling.

5.1.2 Computing probabilities for given directions

For simple path tracing with implicit paths from IRC sampling only, the previous section contains all the steps we need. However, if we want to combine IRC sampling with NEE or BSDF sampling, we also need to be able to extract a probability value from a cache for a given direction in order to compute MIS weights.
Therefor we first pick a cache just like in algorithm 4. Next we convert the direction to local coordinates in the cache's coordinate system and then on to texture coordinates according to equations 4.1 and 4.2. A simple access to the cache's *pdf_map* at the computed texel yields the probability for having sampled the given direction with this cache, measured over solid angle.

5.2 Combining BSDFs and Caches

The figures in chapter 3.3 showed the flaws of choosing only one local sampling method. Figure 3.4 already compared BSDF sampling and NEE. And while our IRCs are capable of rendering caustics, they also produce a considerable amount of noise. This noise can be worse or better than the noise we get from sampling the BSDF, depending on the scene and the quality of the caches.
As opposed to combining BSDF sampling and NEE, BSDF and IRC sampling cannot be done at the same time: Since we use path tracing, we don't want to split up a path into two paths whenever we hit a surface. Hence we need a way to choose between BSDF and IRC sampling and a way to combine them.

We use the BSDF's roughness to decide whether to sample the BSDF or IRCs. If the BSDF is ideally diffuse, the BSDF itself will have the same value for every direction and we cannot gain much information or increase the integrand by sampling the BSDF. On the other hand, if we hit a (near) delta BSDF, for most of the directions the incident radiance will have (almost) no contribution due to the BSDF being close to zero for most angles.

There are certain reasons not to importance sample for irradiance at all. One are delta-BSDFs: Whenever a surface is perfect specular, perfect transmitting or both, there is only one (or two) directions that actually contribute light to the current path. Whenever we hit a surface with a delta-BSDF, we will automatically only sample the BSDF with MIS weight 1, and also skip the next event estimation (if included) completely, since the probability of it having any contribution at all is zero.

Another case depends on the placement of the caches. Sometimes there might just be no caches available close to the current surface point. In simple scenes, this is almost never the case, but we can never be sure to prevent this completely, especially with more complex scene geometry.
A similar case is a more complex, curved geometry. For example, consider placing caches on a sphere. Since we can never cover the whole surface area of a sphere with points, there will always be points that are hit by camera rays but have no cache at their exact position. There might be a cache very close to them, but judging from an analytical point of view the angle between their normals will never be 0. If that angle is still small (or the cosine between them big) enough, we can ignore the deviation and still use the cache, for everything we have was only approximated in the first place. We allowed a small margin of difference between the cache's normal and the surface point's normal (see 5.1.1). But whenever the cosine is smaller than 0.9 we ignore the caches and sample the BSDF with weight 1.
If the cosine of the selected cache is bigger than 0.9 but less than 1, the outgoing direction of our path can be sampled from the IRC. It will however be converted to global and then to local coordinates at the intersection point in order to evaluate the BSDF at that point. Note that the sampled drection was represented in local cache coordinates at first and would also have been converted to global coordinates for a cosine of 1 anyway. See algorithm 4 on how caches are picked depending on their cosine to the surface normal at the current intersection point.

5.2.1 Multiple Importance Sampling of BSDFs and IRCs

As implied in the previous segment, we will only actually combine BSDF and IRC sampling whenever we hit a surface with a non-delta BSDF and if a cache with a good enough cosine to the surface normal is near. In all other cases we continue our path by generating a sample from the BSDF with MIS weight 1.
In this section we show how we decided on a probability to select either BSDF or IRC sampling, and how to compute the MIS weights for each of them.

The Mitsuba Renderer, which was used as a framework to implement this work, offers the roughness of a BSDF in the form of a floating point value $\in [0, 1]$. A roughness value of ∞ is used in addition to indicate perfectly diffuse surfaces.
We decided that even if a surface is completely diffuse, BSDF sampling should not be ignored completely. Then again we never want to pick IRC sampling at perfectly specular surfaces for reasons explained above. So we decided on $\alpha = min(\text{roughness}, 0.9)$ as a threshold for randomly selecting BSDF or IRC sampling.

Chapter 2.1.3 explained how n exclusive sampling techniques can be combined with the one-sample model. In our case BSDF sampling is chosen with probability $1 - \alpha$ and IRC sampling with probability α. These probabilities obviously sum to one, they match c_1 and c_2 in equation 2.11.
Assume we generated a random number $> \alpha$ and sampled some direction ω with a probability p_{bsdf} from the BSDF. To compute that sample's MIS weight we also need to compute the probability p_{cache} for having sampled ω from the chosen cache (see chapter 5.1.2). With both pdfs being measured over solid angle, the weighting function w_{bsdf} for the sampled direction can now be determined by the balance heuristic:

$$w_{bsdf} = \frac{p_{bsdf}}{p_{bsdf} + p_{cache}}.$$

According to equations 2.11 and 3.8 the total weight we have to multiply to the energy carried along the path is

$$pathEnergy^* = \frac{w_{bsdf}}{(1-\alpha)} \cdot \frac{f_s \cdot cos(\omega)}{p_{bsdf}}.$$

Assuming that the path's energy was already weighted similar to algorithm 6 with p_{bsdf} as $pdfValue$ before, we can weight the path energy like this:

$totalWeight \leftarrow \frac{p_{bsdf}}{(1-\alpha)\cdot(p_{bsdf}+p_{cache})}$
$pathEnergy \leftarrow pathEnergy \cdot totalWeight$

If the initial random number was $\leq \alpha$ and 6 was already executed, the MIS weight for IRC sampling can be computed accordingly:

$totalWeight \leftarrow \frac{p_{cache}}{\alpha\cdot(p_{bsdf}+p_{cache})}$
$pathEnergy \leftarrow pathEnergy \cdot totalWeight$

5.3 Combining NEE and Caches

We also combined NEE with IRCs. Basically we replaced BSDF sampling with generating samples from the IRCs, and used the probabilities from the caches instead of the probabilities from the BSDF.
At ideal specular surfaces (those with a delta BSDF) we still used BSDF sampling instead of the IRCs. The probabilities won't matter in that case, as the MIS weights at delta-BSDFs are 1 for BSDF sampling and 0 for any other strategy anyway. Theoretically we also rely on BSDF sampling if no caches are close enough or the closest caches are rotated too much, but this was never the case with our test scenes.

5.4 Evaluation

All scenes are original or modified versions of the scenes available on the Mitsuba renderer's download website [Jak10]. All caches had a resolution of 16×16 per hemisphere, they were filtered bilinear while they were filled and filtered with a Gaussian filter afterwards.

We used the following scenes:

- **Glass Spheres**

  With this scene, BSDF sampling combined with NEE (figure 5.2, left) produced the worst result, i.e. the image with most noise around the caustics. The best image resulted from combining NEE and IRCs - the IRCs perform better than BSDF sampling around the caustics, and NEE produces close to zero noise in the remaining areas (even with less samples per pixel).
  Figures 5.1 and 5.2 each had 200k photon paths, 20k caches, 2k photons per cache and 256 samples per pixel.

- **Mirror Boxes**

  The images with the mirror boxes were rendered with 150k photon paths, 20k caches, 1k photons per cache and 128 samples per pixel. Note that with the glass spheres, the image rendered with NEE and IRCs showed less noise than the other two images. Here, the overall noise is roughly the same for figure 5.3 (NEE and IRCs) and the left part of figure 5.4 (NEE and BSDF). The caustics are still the smoothest with NEE and IRC sampling.

- **MIS Test Scene**

  This scene was initially used by Veach to illustrate the effects of Multiple Importance Sampling, see figure 3.4. The combination of NEE and BSDF sampling is the best choice for this scene; in figure 3.4 there is almost no noise even though it was rendered with only 64 samples per pixel. It seems like combining BSDF sampling and NEE covers the cases of direct lighting pretty well and there was never much room for improvement in the first place.
  As it turns out, either combination of IRCs with BSDF sampling or NEE results in worse images. A possible reason for this is that our caches have a resolution of only 16×16 texel for one hemisphere and are too inaccurate for the smallest light source, so they don't perform well on the left. Plus, we don't expect them to work fine for the upper board either, for reasons explained in section 5.2.
  The images were rendered with 150k photon paths, 20k caches, 1k photons per cache and 64 samples per pixel. 3.4 contains the image for BSDF sampling and NEE, but figure 5.5 should suffice to see the reasons why both combinations with IRCs are inferior.
  Maybe combining NEE, BSDF sampling and IRC sampling results in a more robust algorithm that can handle scenes like this one as well.

- **Glass Egg on Table**

  Veach used this scene in [Vea98, chapter 10] to demonstrate the benefits of bidirectional path tracing over standard path tracing as we used it. While bidirectional path tracing is without a doubt the better choice for this scene, the combination of NEE and IRC sampling results in a distinctly smoother image than the conventional combination of NEE and BSDF sampling (shown in figure 5.6).

5.4.1 Comparison of Different Sampling Strategies

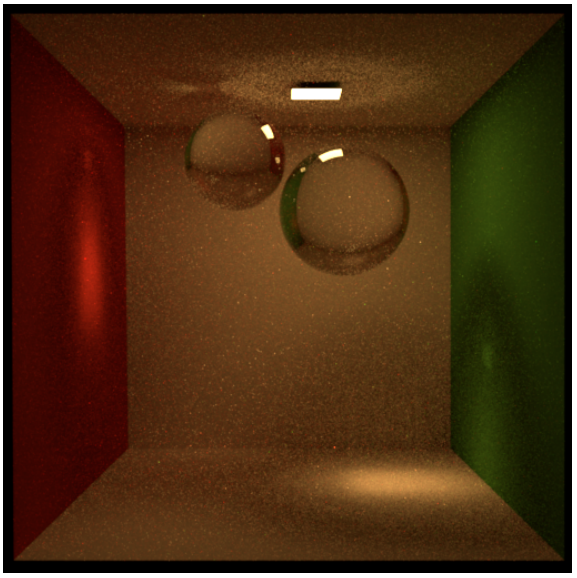

Figure 5.1: NEE and IRC sampling: The IRCs handle the caustics, and NEE takes care of an overall smooth appearance.

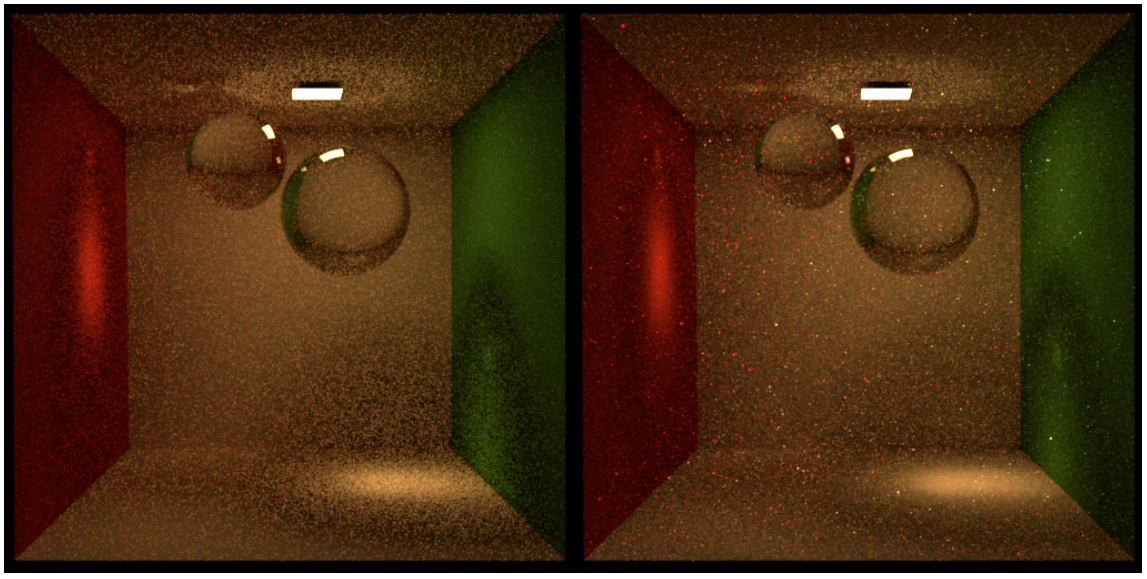

Figure 5.2: Left: NEE and BSDF sampling. In comparison to IRCs, BSDF sampling produces more noise in the caustics. The general noise produced by BSDF sampling consists of darker dots, while the noise of IRCs are colored sparks which can be cancelled out easier by NEE.
Right: BSDF sampling and IRCs. The caustics look better than on the left, but the overall noise is worse.

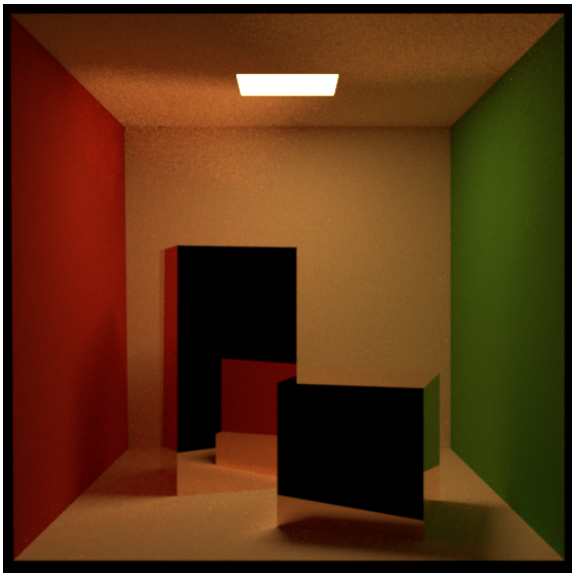

Figure 5.3: NEE and IRC sampling.

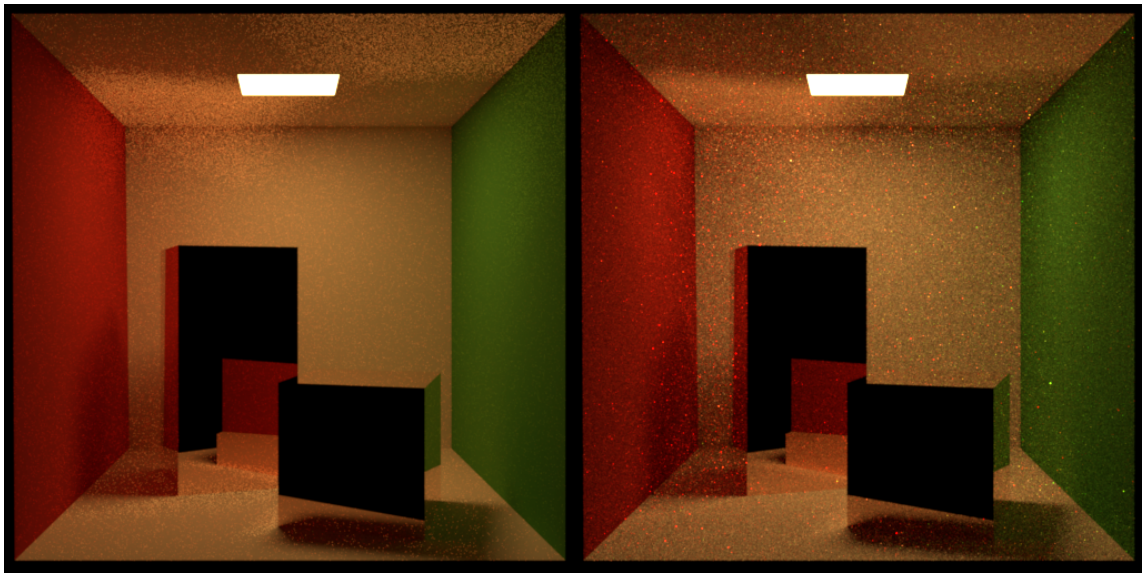

Figure 5.4: Left: NEE and BSDF sampling. Just as with the glass spheres, the caustics show more noise than figure 5.3. The noise on the walls is roughly the same, which might be the result of the brighter light source.
Right: BSDF sampling and IRCs. While the caustics are smooth, there is notable noise from the IRCs.

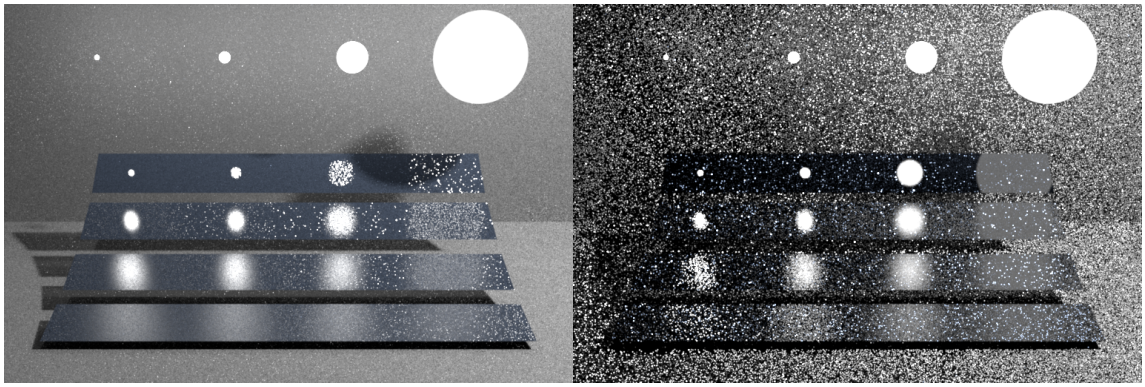

Figure 5.5: Left: NEE and IRCs. Right: BSDF sampling and IRCs.

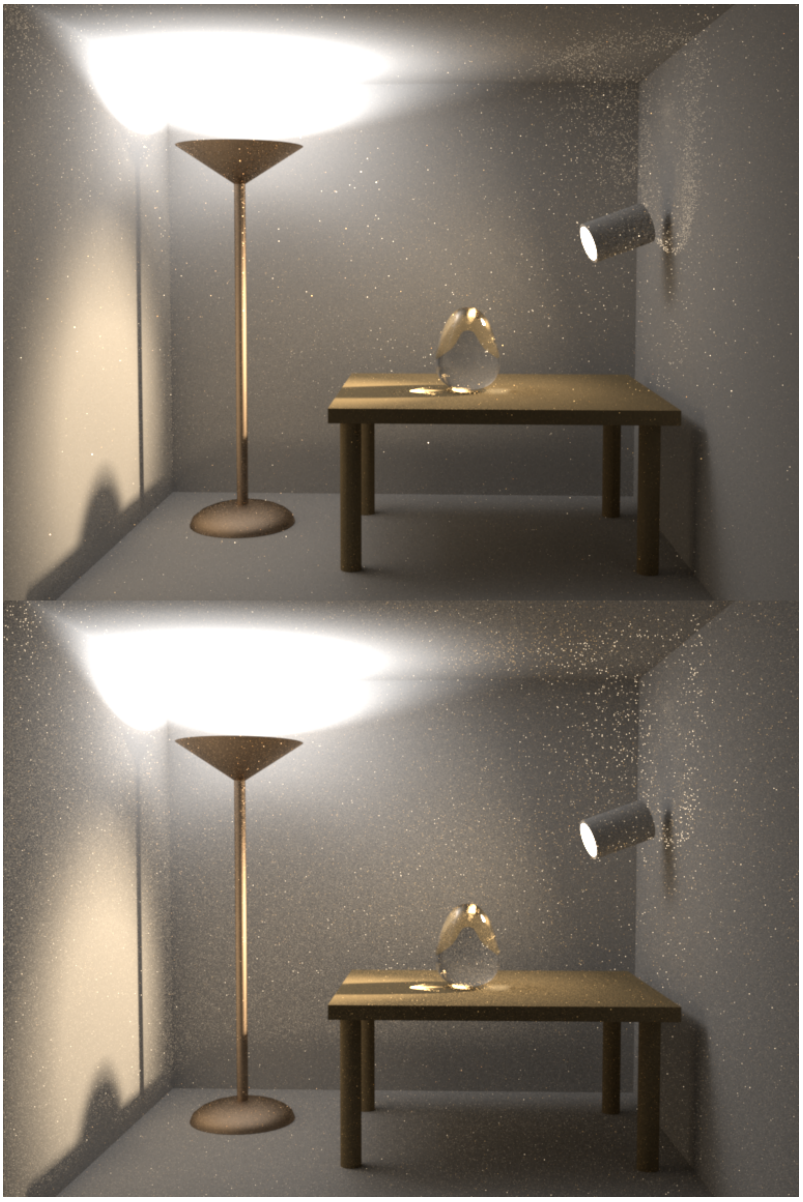

Figure 5.6: Top: Rendered with NEE and IRCs, 250k photon paths ($\approx$ 1.500.000 photons), 30k caches, 2k photons per cache, 4096 samples per pixel.
Bottom: Rendered with NEE and BSDF sampling, 4096 samples per pixel.

5.4.2 Different Cache Configurations

In this section we investigate the impact of a different amount of photon paths, caches and photons per cache.
We found that environment map resolutions of 4×4 and 8×8 were too small, while a resolution 16×16 produced decent results that 32×32 was not able to improve.
As a limitation, we always created the same amount of camera caches (one for every 8×8 pixel block). All additional caches were created from photons. The images below are rendered with NEE and IRC sampling, as this produced the best results.
The preprocessing time is proportional to the number of caches and number of photons per cache. The photon tracing pass itself is several orders of magnitude faster than filling the caches.

As it turns out, for simple scenes where most of the geometry is visible anyway the camera caches are mostly sufficient. However, there may appear artefacts along the edges. These can be removed by either picking one of the five closest caches with a better angle (see algorithm 4), or by adding more caches. For example, the glass sphere scene with camera caches only had 3768 caches and noticable artefacts along the edges within the caustic. These were gone when the same scene was rendered with 20k caches (figure 5.7).

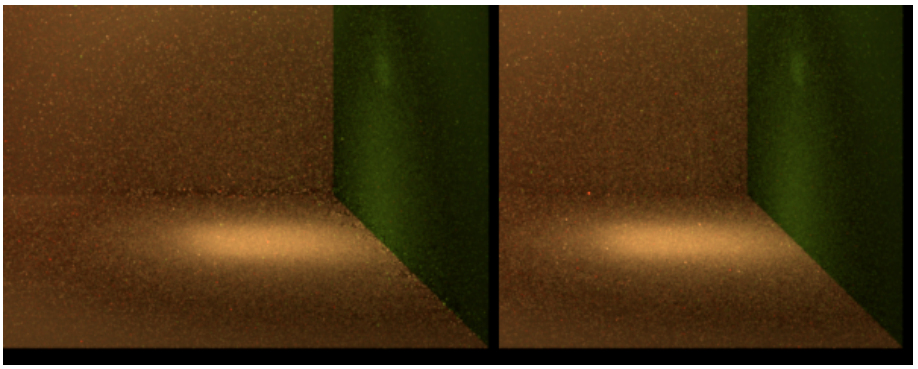

Figure 5.7: Left: camera caches only (3768).
Right: 20k caches including camera caches.
Note how the artefacts vanish on the left where the caustic ends.

We also noticed that reducing the number of photon paths resulted in a general noise not unlike the noise from NEE + BSDF sampling, while reducing the number of photons per cache generated a brighter, more colorful noise. Figure 5.8 illustrates two extremes that can be gradually improved by increasing the number of photon paths or photons per cache.

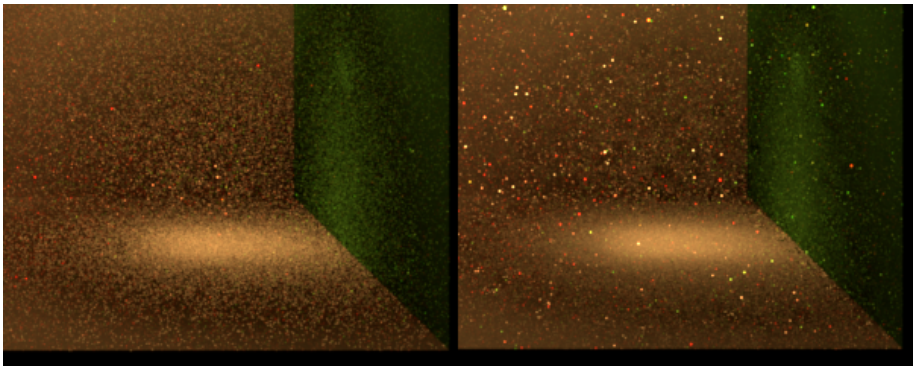

Figure 5.8: Left: 2k photon paths, 3768 caches, 2k photons per cache.
Right: 200k photon paths, 3768 caches, 20 photons per cache.

We settled on around 200k paths and 20k caches with 2k photons each for most of the images shown in section 5.4.2, as bigger numbers were not able to improve the images any further. Smaller numbers either produced noise (with less photon paths or photons per cache 5.8) or caused artefacts along edges (too few caches 5.7).
As a general rule, the number of photon paths should always be at least two orders of magnitude bigger than the number of photons per cache, if we want each cache to only use photons from its immediate surroundings. Our scenes contained an average of 4-8 photons per path. The more open the scene, the smaller this number gets. All our scenes were missing the wall towards the camera; within a completely closed scene there are most likely significantly more photons stored per path.

5.4.3 Durations

We used an existing library to handle k-d trees and added a cdf to the caches so we could sample points faster using binary search, but apart from that we did not optimize our code in any way.
The images were rendered on an Intel i5-3570 CPU with 8GB RAM. The preprocessing of the caches runs in a single thread on one core. The path tracing is multithreaded over 4 cores. We believe the preprocessing step is well suited for parallelization as well, however the main part of the computation time is consumed by the path tracing step anyway.

We measured times for Mitsuba's default path tracer, our path tracer with NEE + BSDF sampling (which does basically the same things as Mitsuba's version) and our path tracer with NEE + IRC sampling. As BSDF + IRC sampling produced the worst results for all of our test scenes, we don't have a detailed list of times for that approach, but it took roughly as long as NEE + IRC sampling.
All times are given in minutes. We used 250k photon paths for 20k IRCs with 2500 photons each. The preprocessing time for the IRCs was 1 min and is included in the quoted times.

	Samples per Pixel	256	512	4096
Glass Spheres	NEE + BSDF sampling	1.4	2.8	22.6
	NEE + IRC sampling	10.9	20.7	152.1
Glass Egg	Mitsuba's path tracer	3.5	7.4	58
	NEE + BSDF sampling	4.4	8.5	71
	NEE + IRC sampling	19.0	37.0	281

Observations:

- Our path tracer (with NEE and BSDF sampling) is not as optimized as Mitsuba's default version. So we believe there is still room for improving the render times.
- The extra time for NEE + IRC sampling is mainly the result of querying a k-d tree every time the ray intersects the scene. The smaller part is consumed by sampling a direction from the cache.
- Picking one of the five closest caches resulted in approximately 50% additional render time but no observable difference in the image. As a consequence we only used the closest cache for the listed measurements. All observed artefact problems can also be solved faster by using more caches.
- For the Glass Spheres scene, NEE + IRC sampling took about 10 times as long as NEE + BSDF sampling, while it only took 4 times as long for the Glass Egg on Table. We assume that the relative overhead for NEE + IRC sampling decreases further with increasing scene complexity, when more time is required to compute ray intersections with the scene.

6. Related Work

This chapter presents some alternative approaches to solve physically based light transport. We will start with Photon Mapping as a basis and introduce Final Gathering, Progressive Photon Mapping and Importance Driven Path Tracing as possible variations.
We also present Irradiance Caching, a technique that does not involve photon maps, but still is rather close to what we were trying to do.

6.1 Photon Mapping

Jensen and Christensen first introduced the idea to use photon maps in a bidirectional Monte Carlo ray tracing algorithm in 1994 ([JC94], [Jen96]). Over the years, many improvements and variations to the original approach were developed. Examples are Final Gathering 6.2, Progressive Photon Mapping 6.3 and Importance Driven Path Tracing using the photon map 6.5.

Most photon map - based algorithms consist of two passes: One pass to create the photon map, and one pass to render the image.
Chapter 4.2.1 already explained how we created our photon map. The exact properties stored for each photon may vary, and sometimes several maps with different degrees of accuracy are built (e.g. *caustic map* for photons that cause caustics, *global photon map* for global illumination [Jen04, chapter 1.5]), but the basic steps for creating a photon map are always the same.

Originally photon maps were used in a distributed ray tracing algorithm. The incident radiance is divided into separate parts, such as a direct illumination component (L_e), a component for caustics and a component for indirect illumination ([Jen04]). The caustics component and indirect illumination are evluated with a radiance estimate from the matching photon map.
Photon Mapping is biased but consistent - the average of many rendered images does not converge to the correct solution, while increasing the number of photons does.

6.2 Final Gathering

Final Gathering is applied after the photon mapping pass and before (or while) the image is rendered. Direct illumination and caustics are handled as described before. To compute the indirect global illumination, we create a camera ray and take its first smooth intersection with the scene, called the *final gather point*. Next, a number of *final gather rays*

is created over the upper hemisphere at the final gather point. At each first non-delta intersection of a final gather ray with the scene, the global photon map is evaluated to compute the indirect illumination along that ray. That way the total indirect illumination at the gather point can be estimated. The final gather rays are not traced any further.
Computing the indirect illumination for a gather point is relatively expensive, so depending on the scene it can be sufficient to create fewer final gather points and interpolate between them.
Many common render engines include final gathering to their photon-map based algorithms, since it is capable of efficiently producing accurate results for indirect illumination ([Mas13], [Wat]).

6.3 Progressive Photon Mapping

The quality of conventional photon mapping is always limited by the memory space available for the photon map. The goal of progressive photon mapping is to use an arbitrarily high amount of photons without having to store all of them at the same time. [HOJ08]
Progressive photon mapping begins with a ray tracing pass to identify and store all visible points (*hit points*) in the scene. The hit points store their current radiance estimate, along with the number of photons and the radius used for that estimate. Thus the image can be rendered after any of the photon tracing passes.
The ray tracing pass is followed by an arbitrary number of photon tracing passes, where each pass increases the accuracy of the radiance estimate for the hit points. In the first photon tracing pass, a photon map is created and a radiance estimate is made for each hit point. Before each subsequent pass, the previous photon map can be deleted. The next pass creates a new photon map that is used to improve the previous radiance estimate. [HOJ08] developed a procedure that increases the number of photons while simultaneously decreasing the radius used for the radiance estimate in each pass, in order to achieve a consistent (though still biased) algorithm.
Progressive photon mapping works especially well for scenes with SDS paths (light transport paths from specular to diffuse to specular surfaces).

6.4 Irradiance Caching

Irradiance Caching was introduced 1988 by [WRC88]. This approach is built upon the observation that indirect illumination changes smoothly on regular diffuse surfaces, while direct lighting may cause “hard” shadow boundaries. Irradiance Caching is designed for ray tracing algorithms where the rendering integral is divided into one part for direct lighting, one for specular reflection and refraction, and a third part for diffuse indirect lighting.
The first two can be approximated with a small number of rays, as they only depend on light from a small solid angle. However, integrating the indirect illumination is quite costly and can lead to an exponentially growing number of rays, since it has to consider light from all directions.

Irradiance Caching aims to approximate and interpolate this diffuse indirect illumination through caches. The caches are created on the fly as necessary in order to avoid irrelevant computation of irradiance that is not needed for the image. They are rather sparsely distributed over the scene, such that regions with uniform irradiance have close to no caches, and regions with varying irradiance receive more.
Each cache contains one value for the total irradiance around its position, and the irradiance values are interpolated from the surrounding caches for each ray intersection. The value directly contributes to the integral.
In contrast, our caches store multiple values for radiance from different directions and our algorithm uses theses values to extend a path.

Ward and Heckbert further suggested irradiance gradients [WH92] to improve the interpolation between irradiance values. Another detailed overview on Irradiance Caching and derived methods is given in [Jar08, chapter 3].

6.5 Importance Driven Path Tracing using the Photon Map

Of all previously explored techniques, Importance Driven Path Tracing is probably the closest thing to our IRCs. In this approach "a rough estimate of the irradiance based on the photon map is combined with the local reflection model to construct more efficient probability density functions that can be used in an importance sampling scheme" ([Jen95, abstract]). The idea was introduced by Henrik Wann Jensen in 1995.

The first step here is photon mapping, where - as opposed to our approach - only photons that were reflected at least once are stored. Thus only indirect illumination is covered. In addition, a photon's flux is stored for different wavelengths.

[Jen95] skips the step of creating caches and starts rendering right after the photon map was created. Whenever they want to sample an outgoing camera path direction at some surface point x with incoming direction ω_i, they build a new pdf using the photon map:

- At first, the N closest photons to x are extracted from a k-d tree.
- Next, the contribution of each photon p with flux Φ_p and incoming direction ω_p to the camera path arriving from ω_i at x is computed as follows:

$$contribution = f_s(\omega_p, x, \omega_i) \cdot \Phi_p \tag{6.1}$$

  In contrast to our approach, this also includes the local BSDF to the pdf.
- Then the direction ω_p of each photon p is converted from a direction $\omega_p = (\theta_p, \phi_p)$ on the hemisphere to a point (u, v) on the unit square.
- The unit square is partitioned into $m \times n$ regions, and each photon's contribution is added to the corresponding part of the unit square.
- The finalizing steps contain some scaling and eliminating all zeros in order to reach a valid pdf.
- An outgoing direction for the path is then sampled from this local pdf.

Jensen was able to reduce the noise in an image through importance sampling with the photon map in comparison to no importance sampling at all.
However, the paper only used test scenes with mostly indirect illumination and - probably because it is relatively old - it did not contain images rendered with BSDF sampling and/or NEE.

7. Conclusion and Future Work

This thesis introduced caches for incident radiance as a possible sampling technique for Monte Carlo path tracing. A photon map was used to approximate incident radiance which was then stored in caches, so the photon map could be discarded before rendering.

We showed that IRCs combined with NEE are a better choice for rendering caustics of all sorts in comparison to conventional path tracing with BSDF sampling and NEE. However, the advantages are restricted to caustics and ordinary diffuse surfaces; for direct lighting and more specular surfaces BSDF sampling and NEE produce better results.

Investigating the rendering times revealed that the preprocessing of the IRCs only consumes a small amount of the additional render time needed for NEE and IRC sampling. The rest of the time is distibuted about evenly between querying the k-d tree for the closest cache and sampling a texel and pdf value from it.
By decreasing the number of samples per pixel, we were able to generate images using NEE and IRC sampling that showed approximately the same quality and computation time as images rendered with NEE and BSDF sampling and a higher sample count. We also believe that the relative overhead for including IRCs decreases for a more complex geometry where more time is spent computing ray intersections.

We would like to make a few concluding suggestions for future improvement.

First of all, our implementation was rarely optimized. The preprocessing of the caches is highly suited for parallelization, we only neglected that because our test scenes were rather simple and therefore the benefit would have been minimal. Plus, the overall structure of our code was designed so that new features could be integrated and tested quickly, it was never optimized for a faster computation time.

Combining NEE, IRC sampling and BSDF sampling together could lead to a more robust algorithm that inherits the benefits from NEE and IRC sampling for caustics on smooth surfaces as well as the strengths of NEE and BSDF sampling for small light sources and more specular surfaces. The computation time should stay approximately the same as with NEE and IRC sampling, and the only modification necessary would be new MIS weights that are able to combine the one-sample model for BSDF and IRC sampling with the many-sample model for NEE and either BSDF or IRC sampling.

Another option to improve the performance of NEE and IRC sampling on specular surfaces might be to include the BSDF when the cache's pdfs are created. A similar approach was already taken with Importance Driven Path Tracing [Jen95], where each photon's energy was weighted with the BSDF for the outgoing direction and the incident direction of the current ray. This was possible because the pdfs were created as soon as a surface was hit, so information on the outgoing direction was available. Our caches are created before the image is rendered, so this method cannot be adopted as it stands.

If necessary, the quality of the caches might be improved by filling them progressively with information that has to be computed anyway in order to render the image.
Progressive photon mapping [HOJ08] already investigated how several photon tracing passes can iteratively increase the accuracy of an image rendered with photon mapping. Similarly, the pdf in the caches could be improved with information from multiple photon tracing passes.
It might even be possible to include the estimated incident radiance that is computed during path tracing. Thus the quality of the pdf in each texel could be increased progressively. However, the resolution of the environment maps used in the IRCs remains a limiting factor.

The concept of IRCs could also be used in a bidirectional path tracing algorithm. In addition to caching incident radiance, it should be possible to add analogous caches for importance from camera rays (instead of radiance from photon rays). Those caches could be employed to sample light paths, while the IRCs we introduced here could be used to sample camera paths.

Bibliography

[Ben75] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, pp. 509–517, Sep. 1975. [Online]. Available: http://doi.acm.org/10.1145/361002.361007

[ED08] T. Engelhardt and C. Dachsbacher, "Octahedron environment maps," in *Proceedings of Vision, Modeling, and Visualization 2008*, 2008.

[HOJ08] T. Hachisuka, S. Ogaki, and H. W. Jensen, "Progressive photon mapping," in *ACM SIGGRAPH Asia 2008 Papers*, ser. SIGGRAPH Asia '08. New York, NY, USA: ACM, 2008, pp. 130:1–130:8. [Online]. Available: http://doi.acm.org/10.1145/1457515.1409083

[Jak10] W. Jakob, "Mitsuba renderer," 2010, http://www.mitsuba-renderer.org.

[Jar08] W. Jarosz, "Efficient monte carlo methods for light transport in scattering media," Ph.D. dissertation, UC San Diego, September 2008.

[JC94] H. W. Jensen and N. J. Christensen, "Photon maps in bidirectional monte carlo ray tracing of complex objects," 1994.

[Jen95] H. W. Jensen, "Importance driven path tracing using the photon map," in *in Eurographics Rendering Workshop*. Springer-Verlag, 1995, pp. 326–335.

[Jen96] ——, "Global illumination using photon maps," in *Proceedings of the Eurographics Workshop on Rendering Techniques '96*. London, UK, UK: Springer-Verlag, 1996, pp. 21–30. [Online]. Available: http://dl.acm.org/citation.cfm?id=275458.275461

[Jen04] ——, "A practical guide to global illumination using ray tracing and photon mapping," in *ACM SIGGRAPH 2004 Course Notes*, ser. SIGGRAPH '04. New York, NY, USA: ACM, 2004. [Online]. Available: http://doi.acm.org/10.1145/1103900.1103920

[Kaj86] J. T. Kajiya, "The rendering equation," in *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, ser. SIGGRAPH '86. New York, NY, USA: ACM, 1986, pp. 143–150.

[Laf96] E. Lafortune, "Mathematical models and monte carlo algorithms for physically based rendering," Tech. Rep., 1996.

[LW93] E. P. Lafortune and Y. D. Willems, "Bi-directional path tracing," in *Proceedings of third international conference on computational graphics and visualization techniques (Compugraphics '93)*, 1993, pp. 145–153.

[Mas13] M. Masters, "Understanding final gather," 8.12.2013. [Online]. Available: http://blog.digitaltutors.com/understanding-final-gather/

[nan14] (2014, october) nanoflann: a c++ header-only fork of flann, a library for k-d trees. [Online]. Available: https://github.com/jlblancoc/nanoflann

[Suy02] F. Suykens, "On robust monte carlo algorithms for multi-pass global illumination," 2002.

[vA11] D. van Antwerpen, "A survey of importance sampling applications in unbiased physically based rendering," 2011.

[Vea98] E. Veach, "Robust monte carlo methods for light transport simulation," Ph.D. dissertation, Stanford, CA, USA, 1998, aAI9837162.

[Wat] Z. Waters, "Photon mapping," visited: 25.01.2015. [Online]. Available: http://web.cs.wpi.edu/~emmanuel/courses/cs563/write_ups/zackw/photon_mapping/PhotonMapping.html

[WH92] G. J. Ward and P. S. Heckbert, "Irradiance gradients," 1992.

[WRC88] G. J. Ward, F. M. Rubinstein, and R. D. Clear, "A ray tracing solution for diffuse interreflection," *SIGGRAPH Comput. Graph.*, vol. 22, no. 4, pp. 85–92, Jun. 1988. [Online]. Available: http://doi.acm.org/10.1145/378456.378490

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

Karlsruhe, 02.02.2015

.....

(**Alisa Jung**)