A Survey of Caching Techniques in Global Illumination

Jinfeng Guo

January 29, 2017

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

In Computer Science, caching is an old technique. Generally speaking, caching means to store certain type of data for faster future (re)use. From the perspective of computing hardware, caching serves as a bridge between hardwares running at different speeds. Data to be cached can be either a copy of an existing piece of data or previously calculated. In the field of global illumination, caching is widely used. Some techniques use caches as an approximation to ground truth, while others use caches as guides for more accurate results.

In this survey, we discuss different types of caching techniques in the field of global illumination. We do not merely focus on techniques with the word *caching* in their name, but ones that we categorize as caching in general. We discuss the problem each technique trying to solve, what kind of data they are storing, how is the data represented, calculated and stored, and finally how are they used to get the final results.

In chapter 2, we present the problem of global illumination, along with foundations and formulations of the problem. Following that, we discuss five different caching techniques. In chapter 3, we talk about irradiance caching and radiance caching, which is considered to be the pioneering caching techniques in the filed of rendering. Chapter 4 is about photon mapping. In chapter 5, we discuss visibility caching. Chapter 6 is about importance caching. Finally, we conclude this survey in chapter 7.

Chapter 2

Global Illumination

GLOBAL illumination is a term used in realistic image synthesis. The goal of global illumination is to capture both direct illumination from light sources as well as indirect illumination scattered from non-emissive surfaces or media in the scene. An example of direct and indirect illumination is shown in Fig. 2.1.

There are two major families of methods solving the global illumination problem [7], viz. *ray tracing* and *radiosity*. Within the ray tracing family, there are shooting based methods, gathering based methods and hybrid methods. We will only focus on the ray tracing based methods, since many good features of radiosity based methods make it computationally expensive, even infeasible. One exemplary good feature thereof is view independence. However, the gap between the two are becoming unclear, as many finite element techniques in shooting based methods of the ray tracing family are brought from radiosity.

2.1 The Problem

The input of any renderer is a description of a virtual scene, with geometry information, light emitting information, surface material information, and possibly participating media information. The output is, of course, imagery. For the output to be realistic, the way each pixel is computed should reflect the true physics processes of light transport in the scene.

In the context of realistic image synthesis, many essentially identical concepts are often presented using different terminologies. Light is a typical example, with names such as *photon*, *particle* or simply *energy*. For now we will simply call a single unit of light that we are interested in *light*. In this section, we will follow the path of *light* from light sources all the way to the camera/eye/sensor.

The light source *light* emitted from could be modeled in numerous ways. It could be a point source, emitting light in all directions. It could be a directional light source such that all peers are going in the same direction prior to initial bounce. Most often light sources have an area, and emit light towards certain direction with a statistical distribution. The way light sources are modeled affects the amount of energy each *light* carries, thus the final distribution of energy in the scene.

After *light* is emitted from a light source, it carries certain amount of energy. The scene contains

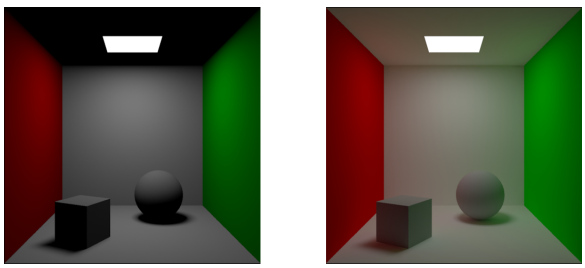


Fig. 2.1: An example of global illumination. Left: direct illumination only. Right: direct plus indirect illumination. Scene: *Cornell Box*. Notice the color bleeding effects with indirect illumination.

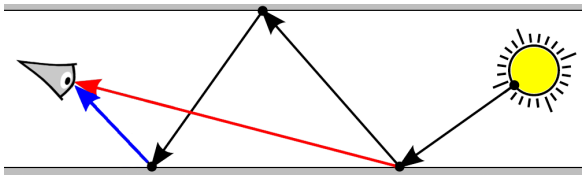


Fig. 2.2: The travel of *light* from the light source to eye. The red arrow represents direct illumination, while the blue arrow represents indirect illumination.

geometries and possibly participating media. It is possible that *light* will hit nothing and vanish. In this case, it is a waste the resources allocated to compute *light*. More probably is that *light* will hit a surface or reach a certain point in the participating media. In this case, *light* interacts with the surface or media.

The surface or media usually has a scattering model, or *albedo*, which gives the distribution of outgoing direction given the incident direction of *light*, as well as the fraction of the original energy that *light* can get away with. Take a red diffuse reflective surface for example. The outgoing direction distribution is uniform in the upper hemisphere. The fraction is determined by the *albedo* of the surface. After this scattering, *light* will leave the surface carrying a bit of red in a random direction.

If by any chance, *light* reaches the camera/eye/sensor only after first bounce, *light* is contributing to direct illumination. If not, it will continue traversing the scene. After certain amount of bounces, *light* may carry multiple information from previously visited surfaces and reach the camera/eye/sensor. In this case, *light* is contributing to indirect illumination. It is probable that *light* will not make it to the camera/eye/sensor. In that case, it would be a potential waste.

Above process is illustrated in Fig. 2.2.

2.2 The Foundation

Before we can formulate the above physical process, we need to lay some foundations. In this section, we will briefly give the physical models of light, transport assumption and scattering models.

2.2.1 Light Models

Light, in its essence, is electromagnetic radiation [26]. For human beings, the fraction that is perceived as visible light ranges approximately from 380 nm to 780 nm in wavelength. Fig. 2.3 gives an illustration of above description.

Visible lights are perceived as colors, which have unique *spectral power distribution*s (SPDs). Each SPD represents a histogram information over the whole visible frequency/wavelength range. There are many successful color models with different application purposes. Among those are *CIE XYZ* color space, *RGB* color model and *CMYK* color model.

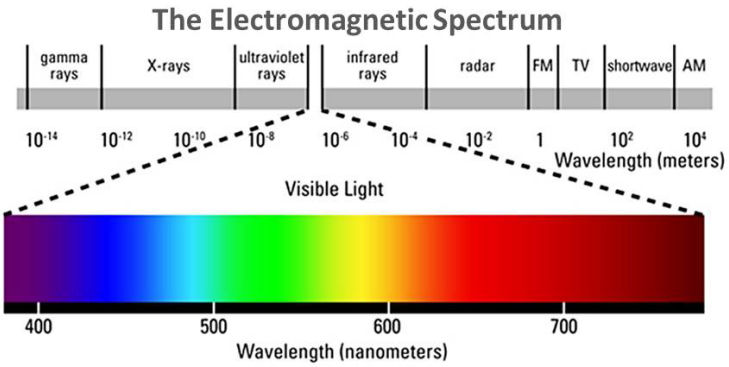


Fig. 2.3: An illustration of visible light wavelength range in the electromagnetic radiation spectrum. (Image courtesy of *Wikipedia*)

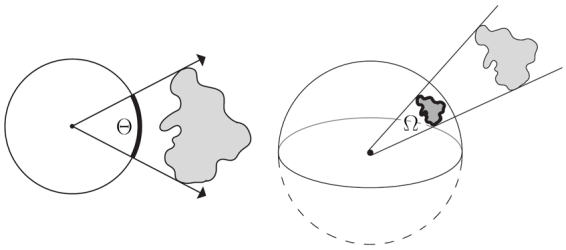


Fig. 2.4: Planar angle and its 3D analogue, solid angle. (Images courtesy of Pharr et al.)

In real world, light have finite speed, even in vacuum. With no participating media, light can travel $299\,792\,458\,\mathrm{m\,s^{-1}}$, or 1 *Planck unit*. However, given the scope of scenes and application purposes, light in computer graphics are usually modeled to have infinite speed. Furthermore, in real world, magnetic waves have effects such as diffraction and polarization when viewed at scales comparable to its wavelength. This is far from feasible given modern computing hardware. Geometric optics is the *de facto* model of light used in computer graphics, for its simplicity. In this model, light travels in straight lines with no energy loss, until a scattering event happens when light hits a surface or reach certain point in the participating media.

2.2.2 Transport Simulation

Flux

Flux, or radiant power, is a term used to measure how much energy is passing through a spatial region per unit time. It is denoted as Φ, with units in *watts* (W, joules per second).

Irradiance and Radiance Exitance

Irradiance (E) is the incident flux arriving at a surface, per unit area. Radiance exitance (M) is the exitant radiant power leaving a surface, per unit area. Both units are in $watts/m^2$.

$$E = \frac{d\Phi}{dA}, M = \frac{d\Phi}{dA}. \tag{2.1}$$

Solid Angle

Solid angle is the 3D extension of *planar angle*. It is denoted as Ω, with units in *steradians* (sr). While planar angle is the projected arc length onto a unit circle, solid angle is the projected area onto a unit sphere. As shown in Fig. 2.4. For example, the solid angle of a full sphere is 4π *sr*.

In spherical coordinates, the differential solid angle is defined as

$$d\Omega = \sin\theta d\theta d\varphi, \tag{2.2}$$

where θ is the colatitude and φ is the longitude. Thus the solid angle can be calculated using the integral

$$\Omega = \iint_S \sin\theta d\theta d\varphi. \tag{2.3}$$

Notice that there is a mis-conception about how (2.2) is derived. It is usually considered to be an approximation of the area using a rectangle with differential edges. However, in *Riemannian Geometry*, there is solid derivation for (2.2). The steps are beyond the scope of this survey, please refer to [24] for more details.

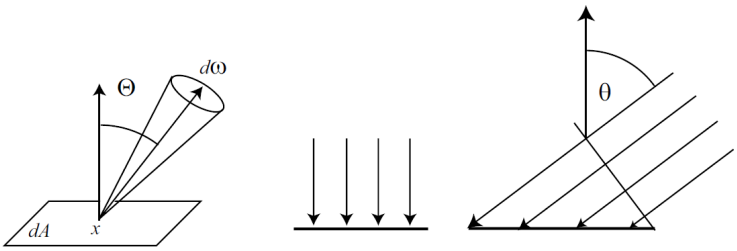


Fig. 2.5: The definition of radiance. (Images courtesy of Dutré et al.)

Radiance

Radiance (L) is the radiant power per unit projected area per unit solid angle. The unit of radiance is in $watts\ /\ (steradian \times m^2\ )$. Radiance can be incident or exitant power.

Radiance is a function of position and direction, see Fig. 2.5. The definition of radiance is given by

$$L(x, \Theta) = \frac{d^2\Phi}{d\omega dA^{\perp}} = \frac{d^2\Phi}{d\omega dA \cos\theta}. \tag{2.4}$$

2.2.3 Surface Scattering

When light emitted from light sources hit a surface, or reach a certain point in the participating media, a scattering event happens. Different materials have different scattering properties. For example a perfect specular surface like the mirror only reflects light towards a direction that is uniquely determined by incident direction and surface normal. A perfect diffuse surface reflects light towards all directions uniformly distributed in the upper hemisphere. Glossy surfaces behaves somewhere in between. Fig. 2.6 illustrates how above three types of surfaces work.

BRDFs

Bi-Directional Reflectance Distribution Functions (BRDFs), are functions that model the fraction of incident flux that is reflected towards a specific direction. This function usually takes as input incident direction, surface point, and outgoing direction. The function is usually defined in the form of

$$f_r(\Psi, x, \Theta)$$

where Ψ is the incident direction, x is the scattering point, Θ is the outgoing direction. This is often referred to as the hemispherical formulation. BRDF can also be written in the form of

$$f_r(x_{i-i} \leftarrow x_i \leftarrow x_{i+1})$$

where x_i is the scattering point, while x_{i+1} and x_{i-1} are previous point and next point along current path. This is referred to as the area formulation or three point formulation. Two formulations are essentially identical, but the latter proves more convenient for path space based techniques.

If we extend the model of BRDF to refractive surfaces, the function remains pretty much unchanged. The only difference is that the outgoing direction now lies in the lower hemisphere. This

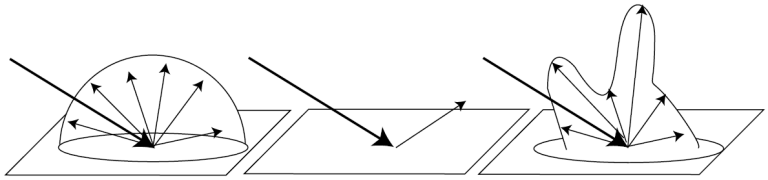


Fig. 2.6: Three different scattering models. Left: perfect diffuse reflection. Middle: perfect specular reflection. Right: glossy reflection. (Images courtesy of Dutré et al.)

function is called *Bi-Directional Transmittance Distribution Function*, or BTDF. The generalization of BRDF and BTDF is BSDF, *Bi-Directional Scattering Distribution Function*. Notice that current model assumes the incident light and the outgoing light have a common point, the scattering point. That is, the ending point of the incident light is the starting point of the outgoing light. If we extend this further, and make the outgoing light initiating from a different point, we get the *subsurface scattering effect*.

There are many BSDF models, and details of those are not the focus of this survey. Please refer to [7] and [26] for more information.

Physically Based Models

One property to notice about BSDF models is that for a model to be physically based, it has to be energy conservative. That is, the energy of all scattered outgoing light combined has to be no larger than the incident energy. Expressed using the hemispherical formulation

$$\int_{\Omega} f_s(\Psi, x, \omega) d\omega \leq 1 \tag{2.5}$$

Apart from energy conservation, a physically based model has to be reciprocal, i.e. $f_s(\Psi, x, \Theta) = f_s(\Theta, x, \Psi)$, thus the name *bidirectional*.

2.3 The Formulation

Before we go into the details, we first briefly recap the history along the path.

The most classical style ray tracer is called *Whitted-style*, which is named after Whitted for he introduced the concept of using recursive ray tracing to solve reflection and refractions in 1979 [41]. However, Whitted-style ray tracers has its limitations, for it can only handle lighting in simple scenes with limited effects.

Cook et al. introduced distributed ray tracing in 1984 [5], which can handle glossy reflections, area light sources, depth of field and motion blur with no additional work than regular Whitted-style. This is achieved by randomly distributing samples in the BRDF, on the area light, on the camera lens and in time, respectively.

In 1986, Kajiya formulated the global illumination problem as the rendering equation [19]. The equation, which is an integral in the form of a Fredholm equation of the second kind, has no deterministic solutions but the Monte Carlo numerical methods (see section 2.3.2). Interestingly enough, Monte Carlo methods was first applied in global illumination by the researchers in the Radiosity family. Since Monte Carlo method is a stochastic process, the method Kajiya introduced is also called *stochastic ray tracing*. However, people favor the other name for this method, *path tracing*, although it only appeared once in the original paper.

Ward et al. introduced the concept of irradiance caching in 1988 [39]. Krivanek et al. build the concept of radiance caching [21] on top of the work of Ward et al. in 2005. Irradiance caching and radiance caching are two methods that will be discussed in detail in chapter 3.

After this the border between gathering based and shooting based methods became unclear. Many hybrid methods were introduced, such as *bidirectional path tracing* (BDPT) [23, 36], independently developed by Lafortune et al., and Veach et al. and *photon mapping* (PM) [16] by Jensen. Photon mapping will be talked about in chapter 4. In 1997, Keller introduced *instant radiosity* by using *virtual point lights* (VPLs) to represent direct and indirect illumination.

The path towards efficient and robust solutions for global illumination problem has not yet come to an end. The best from different sides can be eventually brought together, as we saw the recent advancement of combining photon mapping with bidirectional path tracing in *vertex connection and merging* (VCM) [11] or *unified path sampling* (UPS) [13], independently developed by Georgiev et al. and Hachisuka et al.

2.3.1 The Rendering Equation

The rendering equation is also known as the *light transport equation*, which is essentially a *Boltzmann transport equation*. The original rendering equation is expressed in three-point formulation. We will first present the hemispherical formulation, since it is easier to understand. The rendering equation in hemispherical form is

$$\begin{aligned} L(x \to \Theta) &= L_e(x \to \Theta) + L_s(x \to \Theta) \\ &= L_e(x \to \Theta) + \int_{\Omega_x} f_s(\Theta, x, \omega) L(x \leftarrow \omega) \left|\cos(N_x, \omega)\right| d\omega \end{aligned} \tag{2.6}$$

where:

- $L(x \to \Theta)$ is the radiance from surface point x towards direction Θ
- $L_e(x \to \Theta)$ is the emitted radiance from surface point x, if the surface is non-emissive, this term is simply $\mathbf{0}$
- $L_s(x \to \Theta)$ is the scattered radiance from surface point x, which is intuitively the combination of all scattered incoming radiance from Ω_x, thus the expansion
- Ω_x is the hemisphere or sphere where incoming radiance may come from
- $f_s(\Theta, x, \omega)$ is the BSDF
- $L(x \leftarrow \omega)$ is the incoming irradiance from direction ω
- $\cos(N_x, \omega)$ is the cosine term for computing radiance coming from direction ω.

Notice that in this formulation, we need to trace rays to get the radiance to be scattered, i.e. $L(x \leftarrow \omega)$. The hemispherical form is intuitive, yet it is less convenient for path space based methods, for which paths containing vertices need to be constructed from the camera to light sources, or *vice versa*. The path vertices correspond to points in the three-point formulation, or area formulation. This brings us to the three-point, or area form of the rendering equation, which is

$$\begin{aligned} L(x \leftarrow x') &= L_e(x \leftarrow x') + L_s(x \leftarrow x') \\ &= L_e(x \leftarrow x') + \int_A f_s(x \leftarrow x' \leftarrow x'') L(x' \leftarrow x'') G(x' \leftrightarrow x'') dA(x'') \\ G(x' \leftrightarrow x'') &= V(x' \leftrightarrow x'') \frac{\left|\cos\theta\right| \left|\cos\theta'\right|}{\left\|x'' - x'\right\|^2} \end{aligned} \tag{2.7}$$

where:

- $G(x' \leftrightarrow x'')$ is the geometry term between surface point x' and x''
- A is all the surfaces that x'' could locate
- $V(x' \leftrightarrow x'')$ is the visibility between x' and x'', which is 1 if they are mutually visible and 0 otherwise
- $\cos\theta$ and $\cos\theta'$ are cosine terms of $x'\vec{x}''$ with $N_{x'}$ and $x''\vec{x}'$ with $N_{x''}$
- $\cos\theta'$ together with $\frac{dA(x'')}{\|x''-x'\|^2}$ forms the differential solid angle

This is essentially identical to the hemispherical form, with only a few minor differences. The first difference is that the integral transform from a integral over the (hemi)sphere to a integral over all surface points. This is a different perspective to the same question. The problem that comes along is that not all points are visible from current surface point, x'. Thus the geometry term is introduce to handle both the solid angle issue and the visibility issue.

It is sometimes useful to explicitly write the equation in a form such that direct illumination and indirect illumination are separated. In the three-point form, it is

$$\begin{aligned} L(x_0 \leftarrow x') &= L_e(x_0 \leftarrow x') + L_s(x_0 \leftarrow x') \\ &= L_e(x_0 \leftarrow x') + L_{direct} + L_{indirect} \\ L_{direct} &= \int_{A_e} f_s(x_0 \leftarrow x' \leftarrow x'') L_e(x' \leftarrow x'') G(x' \leftrightarrow x'') dA(x'') \\ L_{indirect} &= \int_{A_{e^c}} f_s(x_0 \leftarrow x' \leftarrow x'') L_s(x' \leftarrow x'') G(x' \leftrightarrow x'') dA(x'') \end{aligned} \tag{2.8}$$

where A_e is the set of all light source surfaces and A_{e^c} is the set of all non emissive surfaces.

2.3.2 Monte Carlo Methods

As mentioned earlier, the rendering equation, whether in hemispherical form or in three-point form, is a Fredholm equation of the second kind. This means the term we are trying to solve also appears on the right side of the equation, hence a recursive nature. For non-toy scenes, it is basically impossible to solve analytically. The only practical solution is to use numerical methods.

Randomized Algorithms

There are two families of randomized algorithms, viz. *Las Vegas methods* and *Monte Carlo methods* [25].

If an algorithm can always gives correct result every time it is run, the only variation from one run to another is the running time, such an algorithm is called a *Las Vegas method*. A typical example is the *Quicksort* algorithm.

If an algorithm can sometime output incorrect results, but the the probability of incorrect results can be bounded, such an algorithm is called a *Monte Carlo method*. This means that given enough runs with independent random inputs each time, the probability of producing incorrect results can be made arbitrarily small, i.e. the average results are statistically close to correct results. The *Min-cut* algorithm is a typical Monte Carlo method.

Monte Carlo integration

For integration problems in real world applications, it usually turns out to be infeasible to compute analytically. Monte Carlo integration is a technique based on the spirit of Monte Carlo methods to solve such problems. The spirit is to use the expected value of some random samples to approximate the true integral. To get the necessary Probability Theory background, please refer to the chapter on same topic in [26], or [22].

Suppose we want to solve the integral for function $f(x)$ over domain $x \in [a, b]$

$$I = \int_a^b f(x)dx = \int_a^b \frac{f(x)}{p(x)} p(x)dx$$

where we can evaluate $f(x)$ but not the integral itself. In Monte Carlo integration, an estimator is used to compute an estimation. The estimator is denoted as $\widehat{I}$ and expressed as

$$\widehat{I} = \frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{p(x_i)}$$

where $p(x)$ is some *probability density function* (PDF) from which N random samples x_i are sampled.

The strength of Monte Carlo integration is that it can handle complex integrals regardless of dimension or the presence of discontinuities. Traditional numerical methods usually fail to converge at acceptable rates or even fail to converge at all when facing integrals in high dimensions or with discontinuities. However, the downside of Monte Carlo method itself is that its convergence rate is of $O(1/\sqrt{n})$. This means we have to evaluate four times more samples if we want to cut the variance in half.

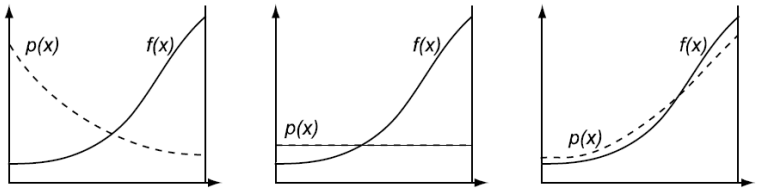


Fig. 2.7: Three different sampling PDFs and the integral. (Images courtesy of Dutré et al.)

Random Samples

The way random samples are generated is very important for Monte Carlo integration. The techniques we are about to discuss in later sections require that we could sample random variables according to an arbitrary distribution.

There are two basic ways to sample random variables given a specific PDF. The first is to uniformly sample the inversed *cumulative density function* (CDF). The other is to reject a sample when the evaluated result of the sample is not what we want. Full details are beyond the scope of this survey, please refer to [26] for details about inversed sampling and rejection methods, and [22] for more sampling methods.

Importance Sampling

Given the integral $\int f(x)dx$ as in Fig. 2.7, if we were to sample according to the PDF in the left image, we might end up putting too many samples where values of $f(x)$ are low. If we were to use the PDF in the middle image, we will get better results, but still lower than correct result. Ideally, we would want to sample according to a PDF like the one in the right image. Intuitively, the optimal PDF is

$$p(x) = \frac{|f(x)|}{\int f(x_i)dx_i},$$

but this is almost impossible since the denominator is the integral we are trying to solve. This process is called *importance sampling*, which is a major *variance reduction* technique. The core idea is to put more samples where the contribution is higher. Many other variance reduction techniques also share the same spirit.

Markov Chain Monte Carlo

A *Markov Chain* is a sequence random events $\{X_1, X_2, \dots\}$ where the conditional probability of event X_{i+1} given $\{X_1, X_2, \dots, X_i\}$ only depends on X_i. The space of X_i is called *state space*, in which each element or event is called a *state*.

Markov Chain Monte Carlo (MCMC), like the original Monte Carlo method, also has a origin in the making of nuclear weapons [2]. It was introduced to simulate thermodynamic equilibrium by the same group of people who introduced Monte Carlo shortly after the introduction of Monte Carlo. The Markov Chain Monte Carlo method is also called the *Metropolis* algorithm, named after the researcher who invented it, or the *Metropolis-Hastings* algorithm, joint-named after the research who generalized the original algorithm.

The core idea of MCMC is to start randomly in the state space. In each step of the following steps, a new state is proposed by mutating current state. A transition probability p_t is calculated, together with the evaluation of the proposed state. Then a random number ξ is used to reject or accept the proposed state by comparing ξ and p_t. If the proposed state is rejected, current state along with its evaluation is recorded (again). Otherwise the proposed state along with its evaluation is recorded. The algorithm keeps running in this way until reaching equilibrium.

The power of MCMC is that it can guarantee *ergodicity*. Take the integral of $\int f(x)dx$ for example, MCMC could always give the correct evaluation without any *a priori* knowledge of $f(x)$. This property is referred to as *perfect sampling*. The original MCMC has the issue of *startup bias*, which was solved by throwing away results from fist few pilot runs.

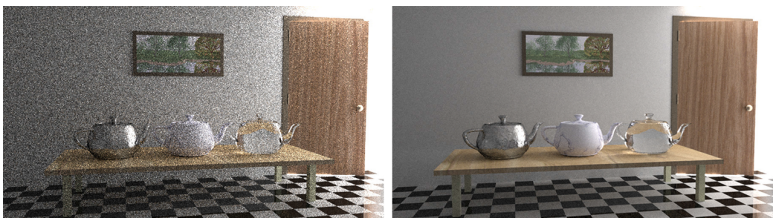


Fig. 2.8: A difficult lighting scene [35]. The scene is lighted by a light source behind the door. All lights come through the gap. The left image is the result of regular BDPT, while the right image is the result of MLT with same rendering time. (Images courtesy of Veach)

The application of MCMC in global illumination is the *Metropolis Light Transport* (MLT). It was first introduced by Veach et al. in 1997 [37]. The original MLT was implemented on top of BDPT, where the state space corresponds to the path space. Please refer to the PhD thesis of Veach for details [35]. MLT has a number of variants, including *primary sample space MLT* (PSSMLT), in which the state space corresponds to the space of random numbers that are used to generate path samples. MLT is very good at handling difficult lighting situation, as it can explore important regions by its nature. This can be seen in Fig. 2.8.

2.4 The Trend

In 1995, Veach et al. introduced *Multiple Importance Sampling* (MIS) into global illumination. It was originally used to combine the contribution of different sampling strategies in importance sampling. The Monte Carlo estimator with MIS is expressed as

$$\begin{aligned} \widehat{I} &= \frac{1}{N}\sum_{i=1}^{N} w(x_i)\frac{f(x_i)}{p(x_i)} \\ &= \frac{1}{N_f}\sum_{i=1}^{N_f} w_f(x_i)\frac{f(x_i)}{p_f(x_i)} + \frac{1}{N_g}\sum_{j=1}^{N_g} w_g(x_j)\frac{f(x_j)}{p_g(x_j)} \\ w_s(x_i) &= \frac{(n_s p_s(x))^\beta}{(\sum_i n_i p_i(x))^\beta} \end{aligned} \tag{2.9}$$

where p_f and p_g are two sampling PDFs, $w_s(x)$ is a weighting function for strategy p_s, and β is the exponent for the *power heuristic*, which, according to Veach et al., gives good result when $\beta = 2$. An example of MIS is shown in Fig. 2.9.

MIS is a bridging tool that brings together different powerful techniques. The only problem is to solve the weighting function as different techniques typically do not belong to the same system, and it only makes sense to compute weights using probabilities when they lie in the same system.

Recent work of Georgiev et al. and Hachisuka et al. brings photo mapping and BDPT together using MIS [11, 13]. The work of Šik et al. explores the combination of MLT and BDPT with VCM/UPS [32]. MIS is playing important roles in these new trends.

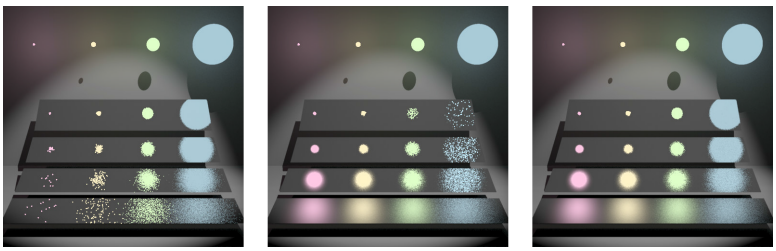


Fig. 2.9: An example of MIS [38]. The left image is sampling the BSDF. The middle image is sampling lights. The right image is the combination of the two using MIS ($\beta = 2$). Surface properties vary from highly specular (1st board) to highly glossy (4th board). (Images courtesy of Veach)

Chapter 3

Irradiance and Radiance Caching

INDIRECT illumination, as mentioned in the previous chapter, adds to the crucial part of realism for rendered images with global illumination. From the knowledge of the previous chapter, we know that indirect illumination usually takes up the majority of the computation time, since we need to average many samples to get the result. However, many research find that indirect illumination tend to vary smoothly over surfaces, as shown in Fig. 3.1.

Fig. 3.1: The rendering of the *conference hall* scene. Left image shows global illumination. Right image shows indirect illumination only. Notice the slow changes of indirect illumination for points on the same surface. (Images courtesy of Ward et al.)

The core idea of irradiance caching is to exploit this smoothness and only evaluate irradiance using sampling at a sparsely distributed set of points on scene surfaces and store them for later use. Irradiance at later points are evaluated by interpolating between caches of available records. New cache records are created when no caches are available. Ward et al. first introduced this idea in 1988 [39].

Since irradiance is the incident flux at surfaces, irradiance caching is limited to outgoing direction independent surfaces only, i.e., diffuse surfaces. Radiance caching is introduced by Krivanek et al. in 2005 [21]. Radiance caching is a generalization of irradiance caching by augmenting caches with complex directional incident distribution. The result is that glossy surfaces can also work with caching scheme.

In this chapter, we first discuss the details of irradiance caching and then present the modification of radiance caching to irradiance caching. In section 3.1, we present the algorithm overview. In section 3.2, we discuss what data to store, how to calculate them and the data structure to store them. Section 3.3 is about the way caches are used during the rendering process. Finally, we present the addition and modification of radiance caching to irradiance caching in section 3.4.

3.1 Irradiance Caching Overview

The overall algorithm of irradiance caching is shown in Algorithm 1. An illustration of one demonstration process is given in Fig. 3.2.

Algorithm 1 Pseudo code for Irradiance Caching.

```
1: procedure IrradianceCaching(p)
2:     query p in the data structure
3:     if one or more records available for interpolation around point p then
4:         interpolate using values and gradients from available records
5:         return interpolated irradiance value
6:     else
7:         calculate irradiance value and irradiance gradients using hemispherical sampling
8:         store new value and gradients as a new record in the data structure
9:         return new irradiance value
```

When no records are available, irradiance values and gradients are calculated at each point using Monte Carlo techniques, i.e. sampling. Otherwise an interpolated result is returned. *Irradiance Gradients* [40] is a piece of geometric information at $\mathbf{p}$ that can be calculated along with the irradiance value without additional cost. It is used in the interpolation phase.

In Algorithm 1, line 2 and 8 correspond to section 3.2.3. Line 3 corresponds to section 3.3.1. Line 4 corresponds to section 3.3.2. Line 7 corresponds to section 3.2.1 and 3.2.2.

3.2 Cache Data and Data Structure

During stochastic ray tracing, we only sample one secondary ray per bounce per iteration and average the results of many iterations to get the final result. This progressive style calculation is prohibitive for calculating cached data, as the later use of caches is based on the assumption that they are accurate enough as a reference. This means, for each cache record, we need to capture irradiance from all directions arriving at current point before we store it as a valid record

$$E(\mathbf{p}) = \int_{H^+} L_i(\mathbf{p}, \omega) \cos\theta d\omega \tag{3.1}$$

where H^+ is the upper hemisphere over $\mathbf{p}$, $L_i(\mathbf{p}, \omega))$ is the incoming radiance from direction ω, and θ is the absolute angle formed by $-\omega$ and surface normal $N_{\mathbf{p}}$.

3.2.1 Data Representation

Since the surfaces irradiance caching are dealing with are diffuse surfaces, plus the fact that no *a priori* knowledge is given about the scene, the only importance sampling technique we could use is *cosine weighted hemispherical sampling*, as the irradiance are weighted by the cosine term. The PDF for hemispherical sampling is

$$p(\theta, \phi) = \frac{\cos(\theta)}{\pi} \tag{3.2}$$

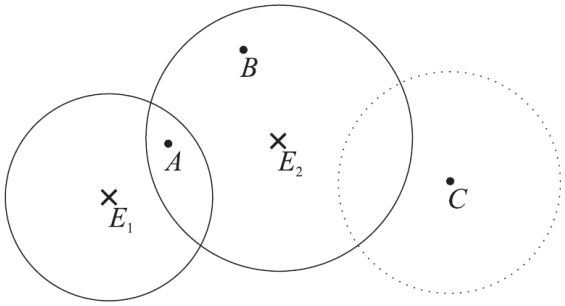


Fig. 3.2: The overall scheme of irradiance caching. Two existing records E_1 and E_2. For point A, irradiance value is the interpolation between E_1 and E_2. For point B, irradiance value is the interpolation of E_2. For point C, a new record has to be calculated and inserted into the data structure. (Image courtesy of Ward et al.)

where π exists so that $\int_0^{\pi/2} p(\theta,\phi)d\theta = 1$.

Stratification is used to make the hemisphere better covered. The hemispherical coordinate θ and ϕ are stratified into M and N strata respectively. Within each stratum, one sample is generated. The estimator for irradiance at point $\mathbf{p}$ is

$$\widehat{E}(\mathbf{p}) = \frac{1}{MN}\sum_{j=0}^{M-1}\sum_{k=0}^{N-1}\frac{f(\theta_{j,k},\phi_{j,k})}{p(\theta_{j,k},\phi_{j,k})},$$

where $\theta_{j,k}$ and $\phi_{j,k}$ are each determined by two random numbers $\xi^1_{j,k}$ and $\xi^2_{j,k}$ using following equation

$$\begin{aligned}\theta_{j,k} &= \arccos\sqrt{1-\frac{j+\xi^1_{j,k}}{M}},\\ \phi_{j,k} &= 2\pi\frac{k+\xi^2_{j,k}}{N}.\end{aligned} \tag{3.3}$$

With reference to Equation 3.1, where $f(\theta,\phi) = L_i(\mathbf{p},\omega)\cos\theta$ and Equation 3.2, we get

$$\widehat{E}(\mathbf{p}) = \frac{\pi}{MN}\sum_{j=0}^{M-1}\sum_{k=0}^{N-1}L_{j,k}(\mathbf{p},\theta_{j,k},\phi_{j,k}). \tag{3.4}$$

Irradiance Gradients

During the rendering phase, irradiance values are interpolated. To better weight available records, it is better to have few geometric information. Intuitively, point position and surface normal information are necessary. Irradiance gradients are introduced by Ward et al. to provide more information than merely position and normal [40]. It provides the first-order approximation of irradiance changes, c.f. derivatives of functions.

Two gradients are defined to capture above information, viz. *rotation gradient* and *translation gradient*. Usually different channels in color models have different gradients.

Rotation gradient gives the rotational axis vector about which the irradiance changes most quickly. The magnitude of the gradient vector denotes the slope in that direction, i.e. how quick is the change. Rotation gradient is calculated along with the calculation of irradiance value, since no additional information is required. It is denoted as $\nabla_r E$ and is expressed as

$$\nabla_r E \approx \frac{\pi}{MN}\sum_{k=0}^{N-1}\left(\mathbf{v}_k\sum_{j-0}^{M-1}-\tan\theta_j L_{j,k}\right) \tag{3.5}$$

where $\mathbf{v}_k$ is a vector in the tangent plane at $\mathbf{p}$ in direction $\phi_k + \frac{\pi}{2}$, $L_{j,k}$ is the sampled incoming radiance for stratum (j,k), same as the radiance term in Equation 3.1.

Translation gradient is an analogue of rotation gradient, only the domain is smaller, as we only care about the change along the surface. For this reason, translation gradient only take into account the changes in the tangent plane. Translation gradient is denoted as $\nabla_t E$ and expressed in a much more complex way as

$$\begin{aligned}\nabla_t E \approx \sum_{k=0}^{N-1}\Bigg[&\mathbf{u}_k\frac{2\pi}{N}\sum_{j=1}^{M-1}\frac{\cos^2\theta_{j_-}\sin\theta_{j_-}}{\min(r_{j,k},r_{j-1,k})}\left(L_{j,k}-L_{j-1,k}\right)+\\ &\mathbf{v}_{k_-}\sum_{j=0}^{M-1}\frac{\cos\theta_j\left(\cos\theta_{j_-}-\cos\theta_{j_+}\right)}{\sin\theta_{j,k}\min(r_{j,k},r_{j,k-1})}\left(L_{j,k}-L_{j,k-1}\right)\Bigg]\end{aligned} \tag{3.6}$$

where:

- $r_{j,k}$ is the distance the sample ray in stratum (j,k) travels before hitting something
- $\theta_{j_-} = \arccos\sqrt{1-\frac{j}{M}}$

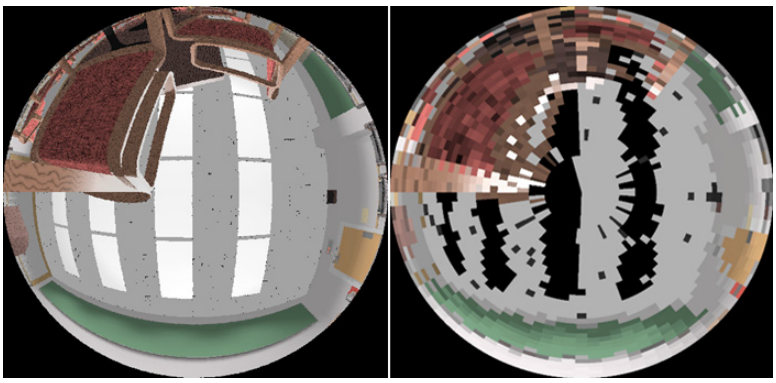


Fig. 3.3: Left: a fish-eye view of a point on the floor in Fig. 3.1. Right: stratified view of indirect illumination from the same point. Notice the black areas in the right image. This is because all direct illumination are ignored and these strata corresponds to lights. Values from all strata are averaged to approximavicinityte the incoming radiance of the full hemisphere. (Images courtesy of Ward et al.)

- $\theta_{j_+} = \arccos\sqrt{1 - \frac{j+1}{M}}$

- $\mathbf{u}_k$ is a vector in the tangent plane at $\mathbf{p}$ in direction $(\pi/2, \phi_k)$, $\phi_k = 2\pi\frac{k+0.5}{N}$

- $\mathbf{v}_{k\_}$ is a vector in the tangent plane at $\mathbf{p}$ in direction $(\pi/2, \phi_{k\_} + \pi/2)$, $\phi_{k\_} = 2\pi\frac{k}{N}$

Just like rotation gradient gives the rotational axis information w.r.t. irradiance changes, translation gradient gives the direction in the tangent plane along which the irradiance changes the fastest. The magnitude of translation gradient represents how fast the change is.

Data to be stored includes irradiance value, rotation gradient, translation gradient and point position and normal. The first term is a vector valued color, e.g. RGB color. The gradients are three-dimensional vectors.

3.2.2 Data Calculation

The way data calculated is a typical Monte Carlo process. The only difference from regular rendering is that at each diffuse bounce, the incoming radiance from the full hemisphere is calculated using Equation 3.4 instead of only one sample per iteration. In each of the $M \cdot N$ strata, a ray is shot towards a direction according to Equation 3.3, which can be easily mapped to world space from local space. Given the equations in the previous section, all necessary data can be calculated in a straightforward way. An example of stratified hemisphere with incoming radiance calculated is shown in Fig. 3.3.

There is a matter of style in caching schemes, two-pass or progressive. The original irradiance caching algorithm as shown in Algorithm 1 shows a progressive nature. However, näive implementation would lead to artifacts.

Consider a surface point like point B in Fig. 3.2. The only available cache record is E_2. If no more records were to be created and added in the empty region around E_2, many vicinity points within the affective radius of E_2 would be only affected by E_2. This is a major cause of artifacts. Many methods have been proposed to handle this, such as the *Best Candidate Pattern* [26], which resembles a *Poisson process*.

Ideally, we would like to render with many cache records already available and distributed in such a way that the surfaces in the scene are covered well enough. This leads to the most popular way of doing irradiance caching, a two-pass style. In the first pass, cache records are created and stored in the data structure. Typically one cache record per pixel to better cover the image. In the second pass, cache records are interpolated to generate final render. One good feature of the two-pass style is that we can still add new records in the second pass. This can be done by simply setting the search radius smaller, so that gaps will appear between existing cache records.

3.2.3 Data Structures

The whole point of using caching in the first place is that it should be faster than direct computation. Therefore the data structure should be efficient at handling cache queries. Apart from queries, one other important task for a spatial structure is to update the structure by inserting new records. Even though record deletion is not used in this case, it turns out to be useful for other caching techniques, as we will see in later chapters.

Intuitively, it should be a spatial structure, since the query entry is point position and normal information. One way is to directly store cache records at surfaces, i.e. via storing at vertices or texturing. However, this makes query for nearest points from different objects hard and adds to the complexity of the scene geometry query structure. Therefore, a better way is to maintain a separate spatial structure for irradiance caches. As we will shortly see in Section 3.3.2, each usable cache record is assigned a weight value during interpolation. This requires usable records to be within certain radius. Each record also possesses a spatial extent as defined in gradients, i.e. a cache record has a radius.

Give the requirements above, the optimal structure is a k-dimensional tree (k-d tree) or an octree. In the original paper, an octree is used to store records. Each record is referenced by a node in the octree. The node contains record position information and the size of the node is proportional to the radius of the record, which is determined by record gradients. The pseudo code for query in the octree is given in Algorithm 2, where $w_i(\mathbf{p})$ is a weighting function used to assign each usable record a weight. Details will be shortly presented in Section 3.3.2.

Algorithm 2 Pseudo Code for Cache Octree Query - Slow

```
1: procedure CacheOctreeSlowQuery(node, p)
2:     for all cache records r_i in node do
3:         if w_i(p) > 0 and p_i not in front of p then
4:             report r_i
5:     for all children child of node do
6:         if p is no further away than half the cube size of child from its boundary then
7:             CacheOctreeSlowQuery(child, p)
```

CacheOctreeSlowQuery is named *slow* because all children are recursively examined under relevant nodes. An improvement could be made by referencing a record to all overlapping nodes within a valid distance. The improved query algorithm is given in Algorithm 3. 2D illustrations of octree structure are given in Fig. 3.4.

Algorithm 3 Pseudo Code for Cache Octree Query - Fast

```
1: procedure CacheOctreeFastQuery(node, p)
2:     node ← root
3:     while node≠NULL do
4:         for all cache records r_i in node do
5:             if w_i(p) > 0 and p_i not in front of p then
6:                 report r_i
7:         node ← child node containing p
```

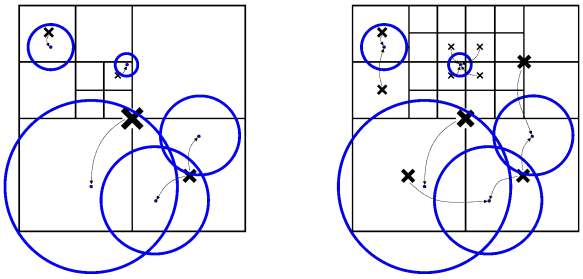


Fig. 3.4: 2D illustrations of record reference assignment for an octree. Left: single referenced octree. Right: multiple referenced octree. (Images courtesy of Ward et al.)

3.3 Rendering Using Cache Data

As shown in Algorithm 1, during rendering, usable records, if any, are queried and interpolated to get the final result. Given the query structure, i.e. octree, we need to further define some terms before completion.

3.3.1 Determine Usable Records

As in line 3 of Algorithm 2 or line 5 of Algorithm 3, a weighting term need to be defined for a record to be considered *usable*. The weighting term $w_i(\mathbf{p})$ of record $\mathbf{r}_i$ for query point $\mathbf{p}$ is expressed as

$$w_i(\mathbf{p}) = \frac{1}{\frac{\|\mathbf{p}-\mathbf{p}_i\|}{R_i} + \sqrt{1-\mathbf{n}\cdot\mathbf{n}_i}} - \frac{1}{\alpha} \tag{3.7}$$

where:

- $\mathbf{p}$ and $\mathbf{p}_i$ are positions for point and record $\mathbf{r}_i$
- $\mathbf{n}$ and $\mathbf{n}_i$ are normals for point and record $\mathbf{r}_i$
- R_i is a distance calculated along with irradiance value and gradients, which is the average distance to visible surfaces
- α is a user specified value to tweak for error

All records with non-zero weight are considered usable records, only their contribution are scaled by different weights.

3.3.2 Advanced Interpolation

The final irradiance at point $\mathbf{p}$ is calculated as the weighted average of all usable records. Rotation and translation gradients are exploited here to yield a better result. The final irradiance value $E(\mathbf{p})$ is calculated as

$$\begin{aligned} E(\mathbf{p}) &= \frac{\sum_i E_i(\mathbf{p}) w_i(\mathbf{p})}{\sum_i w_i(\mathbf{p})}, \\ E_i(\mathbf{p}) &= E_i + (\mathbf{n}_i \times \mathbf{n}) \cdot \nabla_r E_i + (\mathbf{p} - \mathbf{p}_i) \cdot \nabla_t E_i. \end{aligned} \tag{3.8}$$

3.4 Radiance Caching

As mentioned at the beginning of the chapter, irradiance caching mainly targets at indirect illumination for pure diffuse surfaces. This is the major limitation.

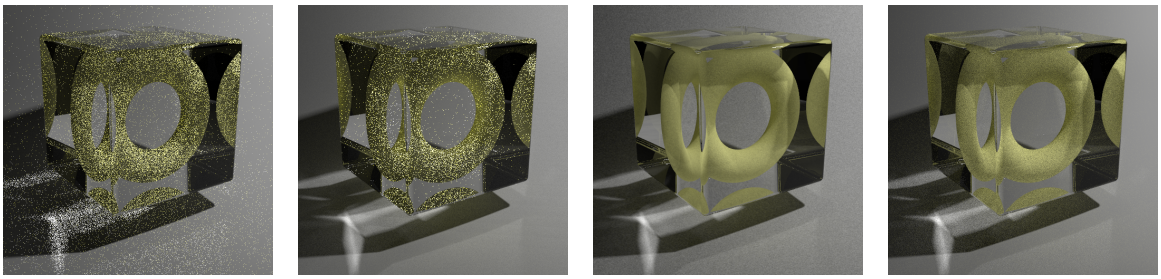
Recall in Section 2.2.3, three types of surfaces exist. For pure specular surfaces, no stochastic evaluation is needed, since they can be handled deterministically. For glossy surfaces, BSDF models thereof are view-dependent. To make irradiance caching scheme compatible with glossy surfaces, the incoming direction information need to be cached along with incoming radiance values.

This is the core idea of radiance caching, that is to cache the directional distribution of incoming radiance at a surface point. The presence of this additional directional distribution information makes it possible to calculate radiance for glossy surfaces.

It is necessary to notice that both irradiance caching and radiance caching are biased techniques. Since the cached values are directly used for final render, bias is inevitable. There are many improvements toward both methods, including making the scheme adaptive and improving error controls.

Chapter 4

Photon Mapping



(a) PT (b) BDPT (c) Photon Mapping (d) Reference

Fig. 4.1: A comparison of different techniques. Scene: diffuse torus in a specular box. The reference image is generated using path tracing with 51,500 samples per pixel. Other three images were rendered with same amount of time. Notice how path tracing fails to efficiently capture the caustics in image (a). BDPT did capture caustics efficiently, however, it fails to efficiently capture specular-diffuse-specular paths. (Images courtesy of Jensen and Hachisuka et al.)

TRADITIONAL ray tracing and Monte Carlo ray tracing techniques are both gathering based techniques. The irradiance and radiance caching fall in this category as well. Shooting based techniques, like light tracing [8], a.k.a. *particle tracing*, faithfully represents the physical process that is really going on in real world: light travels from light sources to eyes. However, pure light tracing algorithm is generally considered inefficient for the same reason we briefly mentioned in Chapter 2. BDPT, as a typical hybrid method, inherit good features from both gathering methods and shooting methods. However, for the notoriously difficult lighting situations, specular-diffuse-specular paths, BDPT performs poorly. A comparison of photon mapping with other techniques is shown in Fig. 4.1.

The core motivation for photon mapping is to distribute illumination in the scene and render with this pre-calculated data. This pre-calculated illumination can be used in several ways, as we will discuss in detail in this chapter. In section 4.1 we present the photon mapping algorithm overview. We discuss about the data used in photon mapping, i.e. photons, and the data structure, i.e. photon map, in section 4.2. Section 4.3 is about how to use the photon map during rendering.

4.1 Photon Mapping Overview

Irradiance caching, as discussed in the previous chapter, pre-calculates and stores illumination information from a gathering point of view, i.e. eye-driven. Photon Mapping is working the other way around from a shooting point of view, i.e. light-driven.

4.1.1 The Algorithm

The algorithm used in photon mapping is a two-pass method, as shown in Algorithm 4. In the first pass, photons are emitted from light sources according to light source properties. At each bounce of

Algorithm 4 Pseudo code for Photon Mapping

```
1: procedure PhotonMapping
2:     photonMap ← ∅
3:     PhotonMappingFirstPass(scene, photonMap)
4:     PhotonMappingSecondPass(scene, photonMap)
```

photon-surface interactions, certain actions are taken according to both photon status and surface property. Actions include keeping bouncing, being terminated by *Russian roulette* or creating a photon record and storing it in the photon map.

In the second pass, ray tracing is used to produce final render. Rays from the camera traverse through the scene. Scattering events are categorized as situations where accurate computation is needed or where approximate estimate is enough. If first bounce happens at specular or highly glossy surfaces, accurate computation is needed, in which case traditional Monte Carlo method is used to sample a ray. Importance sampling is used by jointly sampling BRDF and incoming flux. For bounces at diffuse surfaces, illumination information is estimated using photon *density estimation*.

4.1.2 The Formulation

Recall in section 2.2.2 and 2.3.1, where we discussed the definition of radiance and the rendering equation. Since photon maps store incoming power, it is necessary that we reformulate the rendering equation in a corresponding form. Substituting the radiance term in the hemispherical rendering equation 2.6 with equation 2.4, we get

$$\begin{aligned} L(x \to \Theta) &= L_e(x \to \Theta) + L_s(x \to \Theta) \\ &= L_e(x \to \Theta) + \int_{\Omega_x} f_s(\Theta, x, \omega) L(x \leftarrow \omega) \left|\cos(N_x, \omega)\right| d\omega \\ &= L_e(x \to \Theta) + \int_{\Omega_x} f_s(\Theta, x, \omega) \frac{d^2\Phi(x, \omega)}{d\omega dA \left|\cos(N_x, \omega)\right|} \left|\cos(N_x, \omega)\right| d\omega \\ &= L_e(x \to \Theta) + \int_{\Omega_x} f_s(\Theta, x, \omega) \frac{d^2\Phi(x, \omega)}{dA}. \end{aligned} \tag{4.1}$$

This equation will be used later in section 4.3.1.

4.2 Photon and Photon Map

Photon is originally a concept in Physics. It is used as a elementary particle of electromagnetic radiation. The use of term *photon* and *photon map* in global illumination was first introduced by Jensen et al. in 1995 [18] to distinguish the technique from the term *illumination map*, which stores illumination information as texture maps.

4.2.1 Photon

Like in irradiance caching, each photon record stores information about both geometry and illumination. Geometry information includes spatial position and incident direction. The incident direction is here because we still need it to use the BSDF. Incident direction does not have to be accurate. In the original paper the hemisphere is subdivided into 256×256 cells. Surface normal is not needed because when using photons, we assume that photons within certain radius to a query point have same surface normal.

Illumination information can be a vector-valued spectrum color. Apart from these, since each photon is stored as a node in the data structure, we need to store a flag to indicate spatial split information for the *k-d tree* (see section 4.2.3).

A demonstrating photon record node in code would look like:

Algorithm 5 Pseudo code for Photon Mapping first pass.

```
1: procedure PhotonMappingFirstPass(scene, photonMap)
2:     emit photons from light sources
3:     trace photons in scene
4:     for all photon in photons do
5:         if photon hits a specular surface then
6:             compute reflect/refract direction according to to perfect reflection/refraction
7:             scale photon flux by the reflectivity/refractivity of the surface
8:         else if photon hits a glossy surface then
9:             compute outgoing direction by sampling the BSDF
10:            scale photon flux by the albedo of the surface
11:        else
12:            create a photon record and store it in photonMap
13:            RussianRoulette(photon)
14:            if photon not terminated then
15:                compute outgoing direction by sampling the BSDF
16:                scale photon flux by the albedo of the surface
```

```
struct {
  Point3D p;
  Vector3D incidentDir;
  RGBColor flux;
  Flag flag;
} photonRecord;
```

4.2.2 Photon Tracing

This section gives the process of the first pass, of which the steps are summarized in Algorithm 5.

Photon Emission

Photons, as defined in its Physics origin, carry a certain amount of energy P_i. This energy is a fraction of the energy P_l emitted by the light source l per unit time. Therefore, the power of a light can be considered as the amount of photons emitted per unit time $P_l = \sum P_i$. It is naturally to define the initial energy of a photon by $P_i = \frac{P_l}{N}$ if energy is uniformly distributed among N photons.

Photons emitted from different light sources have different properties. The distribution of directions is one major property. Usually this is identical to the behavior of lights in the traditional rendering process. For scenes with many lights, the problem of balancing between lights could be solved by distributing the fixed amount of photons among lights according to their potential contributions.

Photon Scattering

When photons are emitted from light sources, each one is traced until termination. This process is identical to light tracing in BDPT, particle tracing or reversed ray tracing, where items being traced all behave according to the basic assumptions of lights mentioned in Chapter 2.

When a photon hits a specular surface, it is scattered in the direction that is uniquely determined by perfect reflection/refraction. The outgoing photon should carry a fraction of the original energy that is scaled by the reflectivity/refractivity of the specular surface. No photon record node is created at such scattering events.

When a photon hits a glossy surface, the photon is only reflected or absorbed but no record is created. This is because arbitrary non-diffuse BSDFs usually have somewhat concentrated scattering lobes and can be inefficient if used with cached illumination. This has the same reasoning behind irradiance caching. Since usually in photon mapping, millions of photon records are stored in the photon map, adopting the technique of radiance caching would cost too much memory.

When a photon hits a diffuse surface, a photon record is created and stored as a node in the data structure. Up until now, the photon can still bounce or be absorbed. This is determined by Russian roulette. The direction of the outgoing photon, if not absorbed, is randomly selected in the upper hemisphere, just like what a diffuse surface would do to scatter incident rays. The energy of the outgoing photon is scaled by the surface *albedo*.

Photon Storing

When a photon record is created, it is inserted into the photon map. For each photon we traced, multiple records can be created, as long as it hits diffuse surfaces and survives Russian roulette. Notice that photons are accumulated on surfaces and areas that are receiving more light energy should receive more photons. Therefore the distribution of illumination is in fact a map of density in the scene. Thus we need a data structure that can efficiently answer range queries and find k nearest neighbors.

4.2.3 Photon Map

By the same reasoning in 3.2.3, the data structure behind photon maps should be separate from the scene geometry structure.

The original photon mapping algorithms only generate photon records in the first pass, and uses the map as a static structure. Thus no progressive update of the structure is needed.

Girds are not suitable because photon records are distributed rather non-uniformly in the scene. Octree, as used in irradiance caching is not ideal for range queries.

The original photon mapping algorithm uses a k-d tree, where $k = 3$ in this case. In a k-d tree, each node stores a photon record and two pointers to two subtrees. The two subtrees can be left/right, front/back/ or top/bottom, depending on the flag of the record. For a balanced k-d tree, the query time is $O(\log n)$ where n is the number of photons. With the advantage in range search, a balanced k-d tree can answer k nearest neighbors problem in $O(k+\log n)$ time.

Another suitable data structure is the Voronoi diagram. However, Voronoi diagram uses $O(n^2)$ memory, which is not practical for a map with millions of photon records.

4.3 Rendering with Density Estimation

This section gives the process of the second pass, of which the steps are summarized in Algorithm 6.

Algorithm 6 Pseudo code for Photon Mapping second pass.

```
1: procedure PhotonMappingSecondPass(scene, photonMap)
2:     trace rays from image plane
3:     if hit diffuse surfaces after first bounce then
4:         calculate radiance using DensityEstimate
5:     else
6:         calculate radiance using Monte Carlo ray tracing
```

4.3.1 Density Estimation

Recall in equation 4.1, the scattered radiance term is reformulated in a form that directly exploits incoming flux. Since in the first pass, photons are accumulated on surfaces, we can use the density of photon distribution to estimate radiance at a surface point. This is done by *density estimation*.

In density estimation, the radiance to compute is approximated by locating N nearest photons, summing their illumination information $\Delta\Phi_i$ and dividing by an area ΔA. Thus the scattered radiance

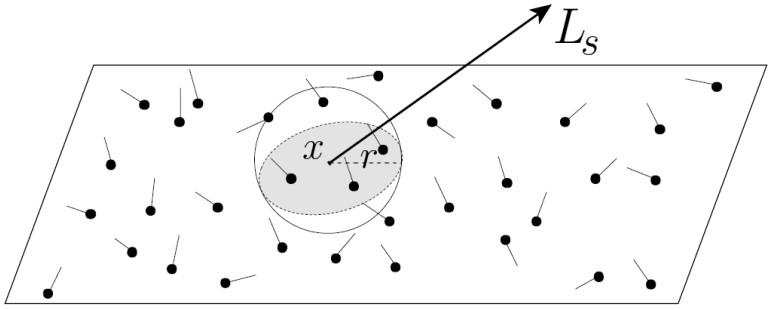


Fig. 4.2: An illustration of density estimation. The process can be considered as growing a sphere until N photons are located. (Image courtesy of Jensen)

term in equation 4.1 can be written as

$$L_s(x \to \Theta) = \int_{\Omega_x} f_s(\Theta, x, \omega_i) \frac{d^2\Phi(x, \omega_i)}{dA}$$
$$\approx \sum_{i=1}^{N} f_s(\Theta, x, \omega_i) \frac{\Delta\Phi_i(x, \omega_i)}{\Delta A}$$

where

- $\Delta\Phi_i(x, \omega_i)$ is the flux stored in photon i, the incident point is assumed to be x
- $\Delta A = \pi r^2$ is the area determined by the furthest photon of the N nearest neighbor photons.

That is

$$L_s(x \to \Theta) \approx \frac{1}{\pi r^2} \sum_{i=1}^{N} f_s(\Theta, x, \omega_i) \Delta\Phi_i(x, \omega_i) \tag{4.2}$$

This process is illustrated in Fig. 4.2. The density estimation process is given in Algorithm 7.

4.3.2 Rendering

Rendering is categorized into accurate computation and approximate evaluation. This is based on how important each region is.

Accurate Computation

For direct illumination, where human eyes can be extremely sensitive to errors, accurate computation is used. As mentioned in section 4.2.2, no photon records are created at specular and glossy surfaces. Thus for specular and glossy scatterings, accurate computation is used. These situations correspond to line 6 in Algorithm 6.

Algorithm 7 Pseudo code for Density Estimate

```
1: procedure DensityEstimate(x, Θ, N, photonMap)
2:     locate N nearest photon records in photonMap
3:     r ← distance to the N-th record
4:     L ← 0
5:     for all photon record p in photonMap do
6:         ω_p ← p.incidentDir
7:         Φ_p ← p.flux
8:         L += f_s(Θ, x, ω_p) × Φ_p
9:     return L / (πr^2)
```

Approximate Evaluation

For scattering events between diffuse surfaces, approximate evaluation is used. This is in fact the majority of indirect illumination. Caustics is handled using a dedicated photon map, which will be discussed in section 4.4.1. This corresponds to line 4 in Algorithm 6.

Filtering

During density estimation, illumination effects like caustics could be blurred because of the kernel density estimator nature. This can be solved by using filtering techniques. Two filters giving good results are the cone filter and the Gaussian filter. The cone filter gives each photon to be used a weight according to the distance to query point. The weight is given as

$$w_{cone} = 1 - \frac{d_p}{kr}$$

where $k \geq 1$ is a user specified constant and r is the distance to the N-th photon record as mentioned in equation 4.2.

The Gaussian filter assigns weights as

$$w_{Gaussian} = \alpha \left[1 - \frac{1 - e^{-\beta \frac{d_p^2}{2r^2}}}{1 - e^{-\beta}} \right]$$

where empirically $\alpha = 1.818$ and $\beta = 1.953$.

4.4 Miscellaneous Photon Maps

In previous sections, we discussed a naïve way of rendering using photon maps. In fact, photon maps can be used in many other ways. In this section, we discuss the usage of photon maps.

4.4.1 Classifying Photons

Caustics Photons

Caustics photons are photons that arrive at diffuse surfaces after being scattered by specular or highly glossy surfaces. Caustics effects are difficult for traditional Monte Carlo ray tracing, as shown in Fig. 4.1 (a). Caustics effects via specular surfaces are even harder for Monte Carlo ray tracing and BDPT, as shown in Fig. 4.1 (b). Since the way we used photon maps in previous section limits the advantage of photons, it is necessary that we dedicate a separate photon map for caustics.

To distinguish this new photon map with previously defined photon map, the new photon map containing only caustic photons is called *caustic photon map* and the original photon map containing all sorts of photons is called *global photon map*. Notice we still construct the global photon map in the first pass.

To construct a caustic photon map, photons are emitted towards specular and highly glossy surfaces in the scene. Since the caustic photon map is directly visualized to give caustic effects, we usually need higher quality than the global photon map. During rendering, the caustic component can be separated and rendered alone. Notice despite caustics are directly visible, we still use approximate evaluation to compute it instead of using traditional Monte Carlo ray tracing.

Shadow Photons

As discussed in previous sections, photon records accumulated on surfaces indicate how much illumination is distributed. It is natural to conclude that shadow regions will accumulate less photons than directly illuminated regions. During the computation of direct illumination, shadow ray visibility tests take a big share of the computation time. Often in a scene, regions are either fully illuminated or fully in shadow. The penumbra regions are usually limited. Based on this observation, we would

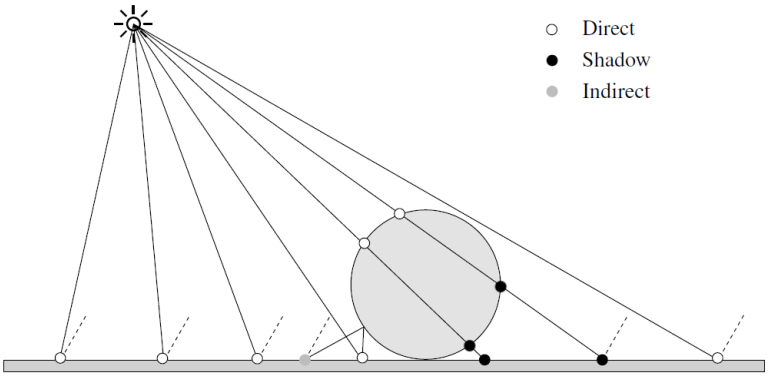


Fig. 4.3: Shadow photon records are created by tracing photons from lights through all geometries without scattering. Only events after first intersections create shadow photon records. (Image courtesy of Jensen et al.)

like to take advantage of photon maps and distinguish penumbra regions from fully illuminated and fully shadowed regions.

Shadow photons are extensions to regular photons. Unlike regular photons records, a shadow photon record is created by tracing photons through all geometries in the scene without scattering, and only create record after first intersections. Each shadow photon should be given a flag indicating the light source from which it is emitted and a flag indicating that it is a shadow photon. The creation of shadow photons is illustrated in Fig. 4.3.

Recall that for direct illumination and specular-reflected/refracted illumination we are still using Monte Carlo ray tracing. The only region that a shadow ray pays off is in the penumbra region. To exploit shadow photons, we simply gather few photon records like we did in density estimation. If all of them are direct illuminated photon records, this region is very likely to be a directly illuminated region. If all of them are shadow photons, this region is for sure a shadow region. If a combination of direct illuminated photon records and shadow photon records happens, this region is then a penumbra region, and we can start tracing shadow rays.

4.4.2 Working with Irradiance Caching

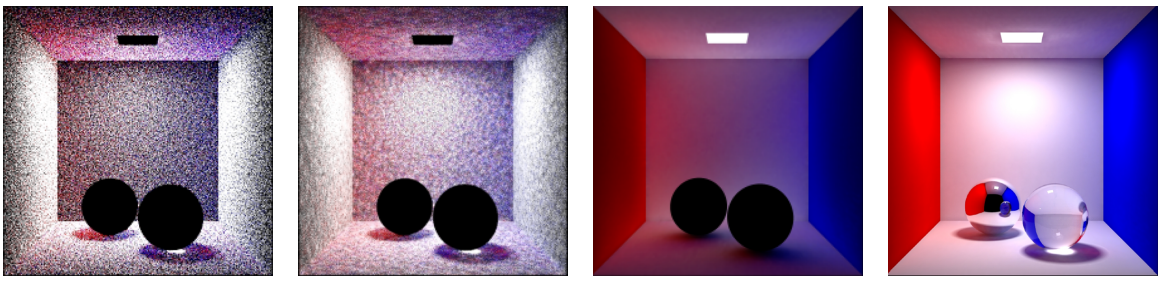
Both irradiance caching and photon mapping are trying to cache illumination information before real rendering. Christensen observed that many photon map query for diffuse inter-reflections in the rendering pass of photon mapping can be simplified by precomputing irradiance values at photon records [3].

As we know from the previous chapter, the computation of irradiance value can be costly. A photon map usually contains millions of photon records, thus it is impractical to compute irradiance value for all photon records. Christensen proposed to only compute for a selected set of all photon records.

Selected photon records are augmented with a computed irradiance value and surface normal information. During the rendering phase, only the nearest photon record with precomputed irradiance value is needed. To make results more accurate, it is better to select a record with similar surface normal as the query point. The results rendered with this technique is shown in Fig. 4.4.

4.4.3 Working as Importance

As we discussed in section 2.3.2, it is almost impossible to get the optimal distribution to importance sample a function without any *a priori* knowledge of the function. With photon mapping, we are distributing illumination information across the scene, and cache the information in a data structure. Intuitively, this information can be exploited as a guidance. Recall in section 2.3, where we solve the



(a) (b) (c) (d)

Fig. 4.4: The Cornell box scene rendered using photon mapping with precomputed irradiance. (a) Visualization of the global photon map. (b) Visualization of precomputed irradiance. (c) Soft indirect illumination term. (d) Final render with global illumination. (Images courtesy of Christensen)

integral in the rendering equation

$$\begin{aligned} L_s &= \int_{\Omega_x} f_s(\Theta, x, \omega) L(x \leftarrow \omega) \left|\cos(N_x, \omega)\right| d\omega \\ &= \int_{\Omega_x} \frac{f_s(\Theta, x, \omega) L(x \leftarrow \omega) \left|\cos(N_x, \omega)\right|}{p(x, \omega)} p(x, \omega) d\omega \end{aligned}$$

using a Monte Carlo estimator

$$\begin{aligned} \widehat{L}_s &= \frac{1}{N} \sum_{i=1}^{N} \frac{f_s(\Theta, x, \omega_i) L(x \leftarrow \omega_i) \left|\cos(N_x, \omega_i)\right|}{p(x, \omega_i)} \\ &= \frac{1}{N} \sum_{i=1}^{N} \frac{f_s(\Theta, x, \omega_i) d^2\Phi(x, \omega_i)/dA}{p(x, \omega_i)}. \end{aligned}$$

Notice the equations are given in a flux-based formulation, which is what a photon map can provide. To importance sample this integral, the PDF is ideally

$$p(x, \omega) \propto \frac{f_s(\Theta, x, \omega) d^2\Phi(x, \omega)}{dA}. \tag{4.3}$$

With this formulation, it is now clear that the photon map we calculated can work as an *a priori* knowledge to guide sampling. Jensen first proposed to use pre-calculated photon map as an importance sampling guidance [17]. To exploit photon map in importance sampling, BSDF and flux information need to be jointly sampled. A simple way to sample flux information by Jensen is to stratify the hemisphere into cells, and build a histogram containing flux information. By uniformly sampling the cumulative histogram, we can easily get an importance sampled direction according to flux information. Fig. 4.5 gives a comparison of traditional path tracing and photon map driven importance sampled path tracing.

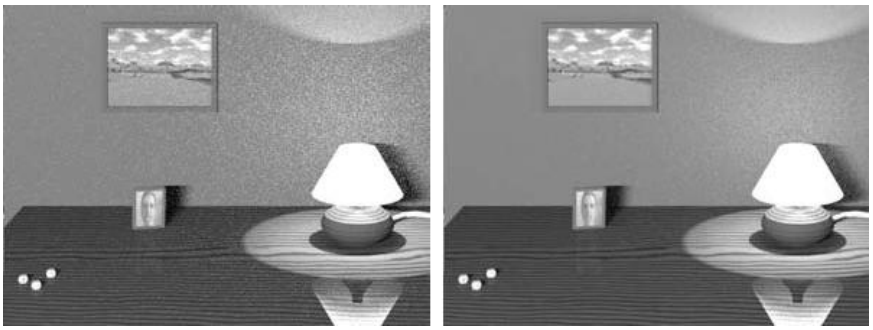
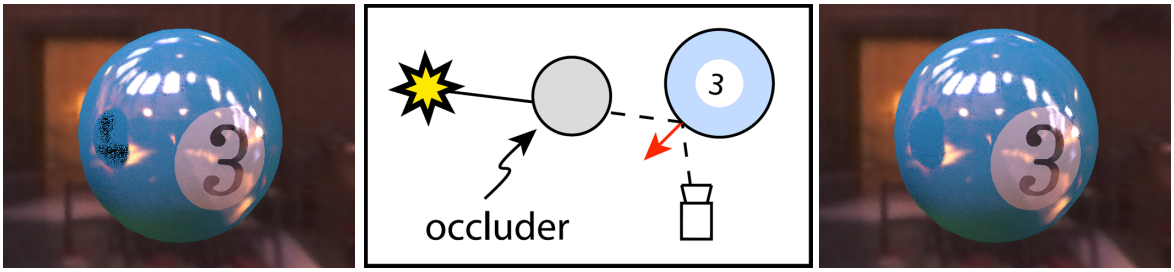


Fig. 4.5: Left: traditional path tracing. Right: path tracing with photon map driven importance sampling. Notice the obvious variance reduction. (Images courtesy of Jensen)

Chapter 5

Visibility Caching



(a) Variance due to visibility (b) Scene setup (c) Reference

Fig. 5.1: Variance caused by visibility. (a) Variance can be obviously seen in the reflection part, where a light source is occluded by another object. (b) The scene setup for the render. (c) The reference image. Notice the difference in corresponding part. (Images courtesy of Clarberg et al.)

Recall in previous chapters, all we did was working around the rendering equation. In irradiance caching, we cache the incoming radiance from all directions, which corresponds to the term $\int L(x \leftarrow \omega) \left|\cos(N_x, \omega)\right| d\omega$ in equation 2.6. In radiance caching, with similar spirit as in irradiance caching, we cache incoming radiance as well as incoming distribution, which makes radiance caching compatible with glossy surfaces. In photon mapping, we work with a Radiosity spirit and use photons to solve the flux term in equation 4.1, which is an equivalent expression of the rendering equation in another form.

With slight reference to equation 2.7, we can find that visibility, as a non-deterministic term, is not exploited yet. On the one hand, in real situations, visibility does introduce variance. Imagine for example in a penumbra region, we sample a point on a light source. The chances are that the sample point on the light source is occluded. In this case, a black pixel will be returned if no further actions are taken. Fig. 5.1 shows one such situation.

On the other hand, visibility tests are expensive. Usually in ray tracing, an accompany acceleration structure is built to accelerate ray-object intersections. To find intersections, a ray must be traversed through the acceleration structure. Rays are categorized into *intersection rays* and *shadow rays*, and corresponding intersections are called *closest hit* and *any hit* [33]. Intersection rays contribute by providing critical information that the rendering equation need and can not be simplified. Shadow rays, recall in section 4.4.1, tend to show correlation. According to many research, shadow rays take up a significant portion of all rays.

In this chapter, we discuss visibility caching, which aims at the visibility term in the rendering equation. Visibility caching is not as old as previous schemes, and still maturing. In section 5.1 we present situations where visibility tests are used. We study the correlation of visibility in section 5.2, which provides theoretical premise for following content. We discuss ways to represent visibility in section 5.3. Section 5.4 is about the data structure to hold visibility caches. We discuss different ways of using visibility caches in section 5.5.

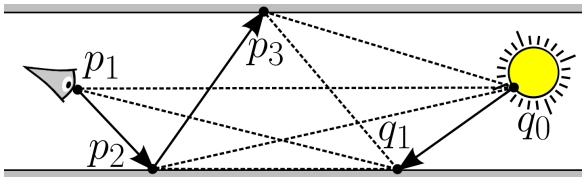


Fig. 5.2: An illustration of visibility tests in BDPT. A eye sub-path of length 3 and a light sub-path of length 2. Dotted lines mean the visibility tests.

5.1 Visibility Tests

As just mentioned, visibility tests are shadow ray queries in the acceleration structure. A shadow ray is a ray formed by two fixed points. The input of a visibility test is a shadow ray $r_s(\mathbf{o}, \vec{d}, t_0)$ with a fixed starting point $\mathbf{o}$ and a finite and fixed t_0 value in direction $\vec{d}$. If during the acceleration structure traversal, there is any hit that gives a smaller t value than t_0, then shadow ray r_s is occluded and two points are not mutually visible. The output of a visibility test is thus a boolean value.

The above process in the pre-ray tracing era is handled by the *z-buffer* technique in rasterization.

5.1.1 Sampling Lights

Visibility tests can happen in many situations in the scope of rendering. One of the major situation is in sampling lights. In fact this is the situation where it gets the name *shadow ray*. In traditional ray tracing, we cast a shadow ray to a light source to determine if a point is in shadow region. In unidirectional path tracing with *next event estimation*, we not only sample the BSDF, but also lights, see Fig. 2.9. In *many-light rendering*, shadow rays connect shading points to VPLs, see section 6.2.1.

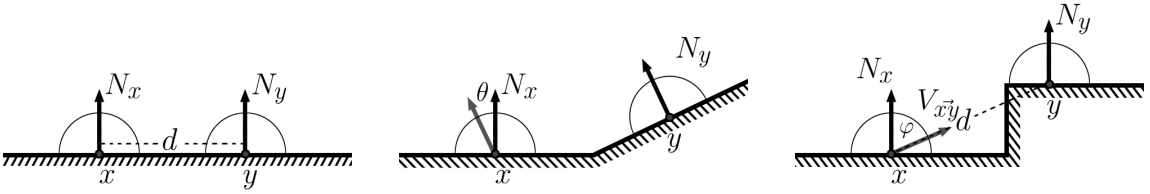
5.1.2 Connecting Vertices in BDPT

In BDPT, for each sample within a pixel, two sub-paths are generated. An eye sub-path $\{p_1, p_2, \dots p_x\}$ and a light sub-path $\{q_y, \dots q_1, q_0\}$ are constructed from the camera and a light source with length x and y respectively, where p_i is an eye sub-path vertex and q_j is a light sub-path vertex.

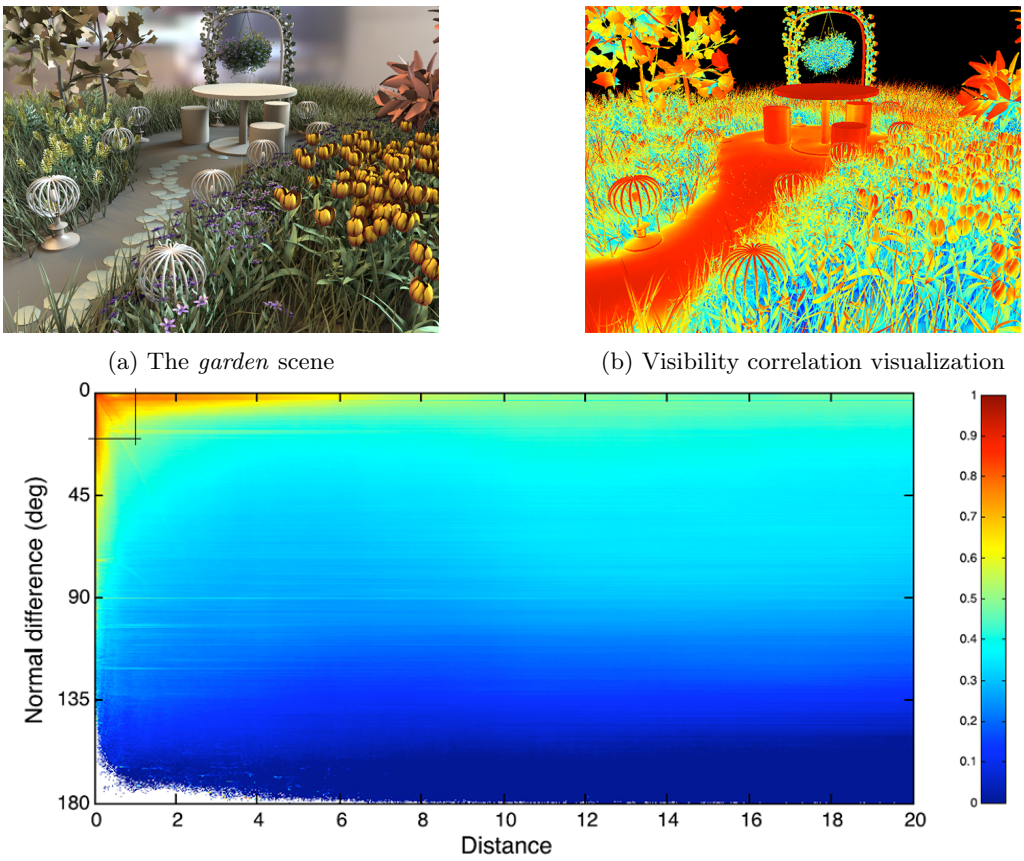
To construct a final path that connects the eye sub-path and the light sub-path, visibilities between vertices from two sides are tested and the contribution of each connection strategy is combined using MIS (see section 2.4). In some literature, these rays are referred to as *visibility rays* to better describe their functionality and distinguish them from shadow rays. This process is shown in Fig. 5.2.

5.2 Visibility Correlation

Recall in section 4.4.1, where we introduced shadow photons. Intuitively, a significant portion of scene surfaces share similar hemispherical visibility with their vicinity. In 2008, Clarberg et al. measured visibility correlation and proposed a robust algorithm to exploit this correlation [4]. There are mainly three components that are affecting visibility correlation, viz. distance, normal difference and surface offset, as shown below.



(a) Distance (b) Normal difference (c) Surface offset



(a) The *garden* scene (b) Visibility correlation visualization

Fig. 5.4: Visibility correlation in a complex scene. Scene: the *garden*. The average correlation is a function of distance and normal difference. The results are clamped to $[0, 1]$, where 1 means strongly positively correlated, 0 means not correlated. (Images courtesy of Clarberg et al.)

Clarberg et al. setup experiments measuring the relationship between average correlation as a function of distance and normal difference $\bar{\rho}(d, \theta)$. During the experiments, environmental light is used so that an outgoing direction ω can be used to indicate illumination from the same light source, i.e. at point x and y, if $V_x(x, \omega) = V_y(y, \omega)$, point x and y are receiving same illumination from direction ω. The results of one example scene from [4] are shown in Fig. 5.4.

As can be seen from the results, visibility is often highly correlated over large smooth surfaces [4]. For flat areas like the walk path on the ground, correlation is very strong, while for complex areas like the grass, correlation is negligible. Thus it is possible to exploit this correlation in global illumination.

5.3 Visibility Representation

Visibility is extremely direction dependent, thus we need a representation that preserves directional information. Intuitively, the starting point is the upper or lower hemisphere, or full sphere when participating media exists.

Another representation is point-point mutual visibility. However, this representation require huge amount of memory and expensive lookups, especially as scene grows complex.

In this section we present several visibility representations, which are mostly based on hemispherical or spherical visibility.

5.3.1 Shadow Maps

Shadow map is originally used to determine if a point in scene is lit by a specific light source. The method was proposed by Williams in 1978 [42]. Shadow maps and its variants are considered to be

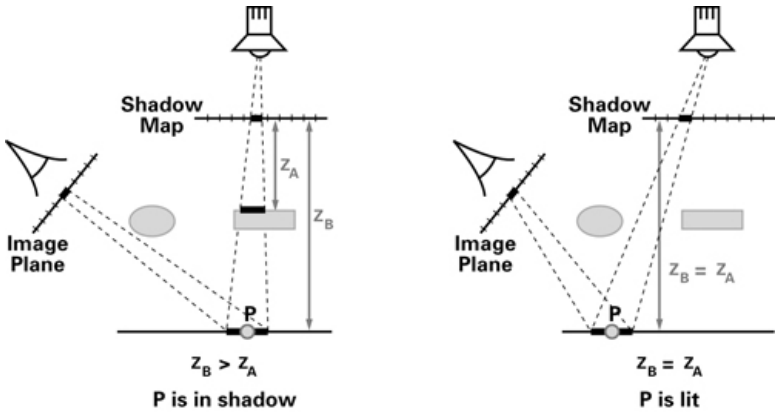


Fig. 5.5: An illustration of shading with shadow map. Left: depth value of point **p** from the light source is larger than the stored value in corresponding SM pixel, thus in shadow. Right: depth value of point **p** from the light source equals stored value in corresponding shadow map pixel, thus lit. (Images courtesy of NVIDIA)

the *de facto* standard technique to solve shadow problems in rasterization.

The observation that shadow mapping based on is the fact that every point visible from a light is lit by the light, and those invisible lie in the shadow region of this light. This observation does not take specular surfaces into consideration, which makes sense because in rasterization reflections and refractions are not solved the way we solve them in ray tracing. The process of creating a shadow map is conceptually simple, a point on the light source is used as a view point, then the scene seen from this viewpoint is rendered to an image. This can be a unidirectional view, or a omni directional view, with difference being the opening angle of the view frustum.

The scene is not rendered as a full colored representation to the viewpoint, but a map of depth values, i.e. distance of visible point to the view point. When using this shadow map during rendering, the distance to the light source from the point to be shaded is checked and compare to the depth value stored in corresponding pixel of the shadow map. This process is illustrated in Fig. 5.5. For more details of shadow mapping techniques, refer to the original paper [42] and the book *Real-time shadows* [9].

The depth value corresponds to the t value of a ray in ray tracing, which is a very convenient coincidence. We can use shadow maps with the same spirit and render the (hemi-)spherical view of the scene into depth/distance values. For this step, we can use multiple techniques. Rasterization can be fast for its efficient hardware implementation. Ray casting, on the other hand, provides similar results but less rapid. Since all we need for this visibility shadow map is the depth/distance value of direct visible surface points, rasterization is a good option, as is the case in [34].

The pseudo code for creating visibility caches using shadow map as visibility representation is shown in Algorithm 8. The procedure DistributePoints shares same spirit as the pre-pass in (ir)radiance caching and photon mapping. Records are only created at non-specular surfaces. The procedure RenderDepth renders the corresponding view into a shadow map, either a rasterizer or a ray caster can be used here .

5.3.2 Variants of Shadow Maps and Others

Imperfect Shadow Maps

Imperfect Shadow Maps (ISM), is a technique proposed by Ritschel et al. to work with VPL based global illumination algorithms [30]. VPL based instant radiosity methods, as global illumination techniques using rasterization, benefit greatly from hardware implementations. Despite this advantage, it is still impractical to render a regular shadow map for each VPL. On the one hand, accurate visibility is not necessary for indirect illumination to be physically plausible. On the other hand, a regular shadow map for each VPL is inefficient both memory-wise and lookup-wise.

In short, an ISM is a low resolution shadow map, e.g. 32×32. The core idea is to distribute a point set to crudely represent the scene. The points in the point set are VPLs. For each point in

Algorithm 8 Pseudo code to create visibility caches

```
1:  procedure CreateVisCache(scene, visibilityMap)
2:      P ← DistributePoints(scene, non-specular)
3:      for all p_i in P do
4:          r_i ← ∅
5:          r_i.p ← p_i
6:          r_i.n ← p_i.n
7:          if p in participating media then
8:              r_i.map ← RenderDepth(p_i, sphereical-view)
9:          else
10:             r_i.map ← RenderDepth(p_i, hemisphereical-view)
11:         Store r_i in visibilityMap
```

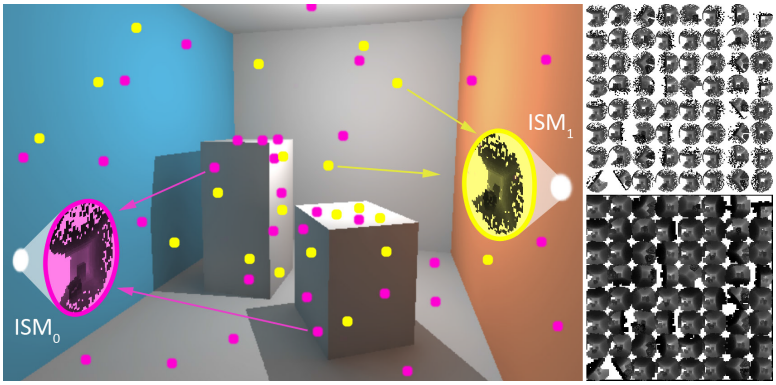


Fig. 5.6: A scene rendered using imperfect shadow maps. Two ISMs are shown as a demonstration in the left image, with corresponding colored dots contributing to the ISM. All ISMs used are shown to the right, with the top image presenting ISMs rendered without pull-push, the bottom image with pull-push. (Images courtesy of Ritschel et al.)

the point set, an ISM is rendered and later used to approximate visibility in the indirect illumination computation.

Though initially proposed to solve indirect illumination, where for many applications accurate visibility is not necessary to produce physically plausible images, ISMs are used for direct illumination as well. This will introduce obvious errors, such as holes, due to its low resolution. This problem is solved by using a pull-push correction. An example scene rendered with ISM is shown in Fig. 5.6.

There are many improvements toward ISM. Ritschel et al. makes ISM view adaptive by placing more sample points near camera viewpoint, instead of uniformly distribute samples through out the scene [31]. Hollander et al. proposed a *ManyLoDs* algorithm to compute desired data, e.g. a shadow map, from many views using the scene acceleration structure [15]. The process resembles a node cut style traversal scheme, like the *light cuts* algorithm. This improvement is more beneficial for GPUs where many ISMs need to be computed in parallel.

Octahedral Representation

Other than regular (hemi)spherical representations, octahedral map is also frequently used to represent data. Radziewsky et al. use an octahedral map to represent visibility in volume rendering [29], where a visibility grid is set up and in each of the eight faces, a sweeping plane algorithm is used to sweep through all grids and accumulate the visibility throughput. Both the use of octahedral map and sweeping scheme make the algorithm efficient for their case. An example of octahedral map is shown in Fig. 5.7.

Cluster-based Representation

Popov et al. proposed to cluster scene surfaces and use inter-cluster visibility to approximate the point-point visibility tests [27]. The assumption is that points belong to the same cluster share

(a) A 2D representation of an octahedral map (b) The octahedral map in 3D view

Fig. 5.7: An example of octahedral map.

exactly same visibility. A hash map is used to store cluster caches. The scheme is shown in Fig. 5.8.

5.4 Visibility Query Structure

Recall in previous chapters, where we talked about requirements for a data structure to be eligible. For the case of visibility caches, we still face the same issue. Thus, from the experience of previous caching techniques, the suitable data structure for visibility caching is a k-d tree.

5.5 Using the Visibility Cache

When using the visibility cache, we are again facing the problem when we did with photon mapping: whether we want to directly use the cache in the computation or we want to use it as a guidance for importance sampling. For direct illumination, the scheme we represent and cache the visibility would not benefit much since we are not using rasterization during rendering. For indirect illumination, as mentioned previously, visibility can be approximated by interpolating caches.

5.5.1 Indirect Illumination

To use visibility cache, we need to restrict usable caches to be within a certain range. In other words, we need to use the scheme of irradiance caching to locate cache records within an acceptable range, instead of the scheme of photon mapping.

When we have the records, we need to assign each record a weight according to the distance, normal difference and displacement. With slight reference to the illustrations in section 5.2 ,the weight function is given by

$$w(x,y) = \left(1 - \frac{\arccos(|N_x \cdot N_y|)}{\pi}\right)\left(1 - \frac{d/d_{max}}{1+5d/d_{max}}\right)\sqrt{1-|N_x \cdot V_{\vec{xy}}|} \tag{5.1}$$

where:

- $1 - \frac{\arccos(|N_x \cdot N_y|)}{\pi}$ is the normal difference

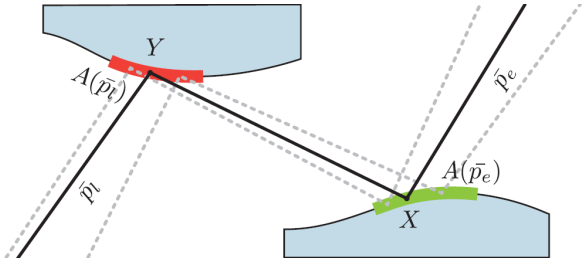


Fig. 5.8: Paths containing vertices in the same clusters share same visibility. (Image courtesy of Popov et al.)

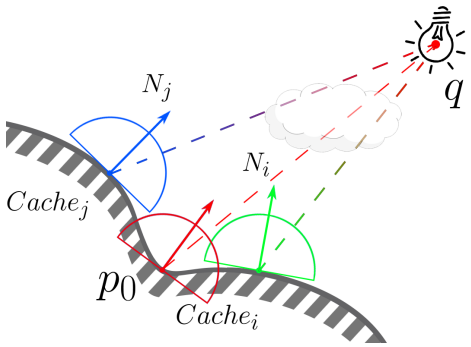


Fig. 5.9: An illustration of using the visibility cache. For query point $\mathbf{p}_0$ and $\mathbf{q}$, two cache records $Cache_i$ and $Cache_j$ are located. The approximated visibility is returned as the weighted sum of results from $Cache_i$ and $Cache_j$.

- $1 - \frac{d/d_{max}}{1+5d/d_{max}}$ is the spatial difference, where d is the distance between point x and y
- $\sqrt{1 - |N_x \cdot V_{\vec{xy}}|}$ is the surface offset, $V_{\vec{xy}}$ is the unit vector pointing from x to y.

The process of using the visibility cache for visibility approximation is shown in Fig. 5.9 and Algorithm 9. Notice that the returned value is an weighted average of distance. The rest of the rendering process is identical to whatever techniques applicable and will not be repeated here.

Algorithm 9 Pseudo code to use visibility caches

```
1: procedure UseVisCache(p0, q, visibilityMap)
2:     P ← LocateRecords(p0)
3:     V ← 0
4:     for all pi in P do
5:         wi ← WeightRecord(p0, pi)
6:         vi ← QueryMap(pi.map, q)
7:         V += vi × wi
8:     return V
```

5.5.2 Direct Illumination

Clarberg et al. uses *control variates* technique to formulate the rendering equation with the cached visibility as a control variable. As we discovered in section 5.2 that visibility correlation does exist, the following equation can be formulated

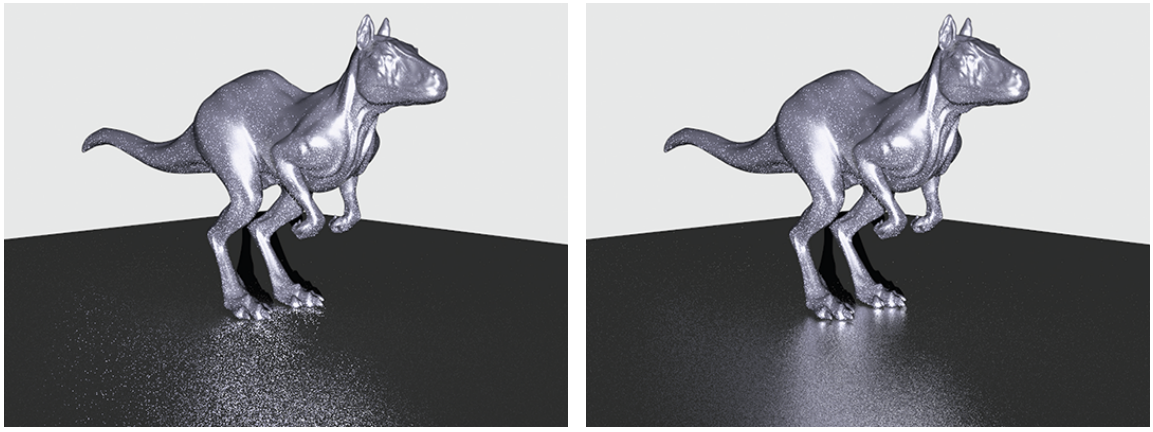
$$\begin{aligned}\widehat{L}_o &= \frac{1}{N}\sum_{i=1}^{N}\frac{L_i f_s G V}{p(\omega_i)} \\ &= \frac{1}{N}\sum_{i=1}^{N}\frac{L_i f_s G V - \alpha L_i f_s G \widetilde{V}}{p(\omega_i)} + \alpha \int L_i f_s G \widetilde{V} d\omega\end{aligned}$$

where $\widetilde{V}$ is the approximated visibility term using the visibility cache. This estimator is unbiased. If V and $\widetilde{V}$ are indeed correlated, this estimator will cause no more, if no less, variance than the original one.

To exploit this feature, the core idea is to augment the lighting-reflection product sampling and perform lighting-visibility-reflection joint sampling, i.e. $p \propto L_i f_s G \widetilde{V}$. Clarberg et al. uses environmental maps as light source, which makes this joint sampling more practical than with unstructured lights. This reduces variance as expected, refer to [4] for more details.

Chapter 6

Importance Caching



(a) Uniform sampling the hemisphere (b) Importance sampling the BRDF

Fig. 6.1: A comparison of same time rendering of w/o importance sampling and w/ importance sampling. Notice the obvious difference in the variance level on the ground. (Images courtesy of Pharr et al.)

Up until now, we have explored techniques to pre-calculate various terms in the rendering equation (equations 2.6, 2.7 and 2.8), from irradiance to radiance, from flux to visibility. These terms all either directly contribute to the final render or indirectly contribute by acting as a guidance for importance sampling. Fig. 6.1 shows an example of importance sampling the BRDF, which is an analytical term in the rendering equation. In a sense, if we are using pre-calculated values for importance sampling purposes, we are actually performing importance caching.

Georgiev et al. first proposed the technique of *importance caching* in 2012 [10]. This chapter will build mostly on top of his work. In this chapter, we discuss techniques to *explicitly cache importance*. We talk more about importance itself in section 6.1 to get a better sense of the importance of *importance*. In section 6.2, we present the algorithm overview of this caching scheme. We discuss the representations of importance and ways to calculate them in section 6.3, and briefly talk about the accompanied data structure in section 6.4. We finally talk about ways to use this cache in rendering and present applications of similar techniques in different contexts in section 6.5.

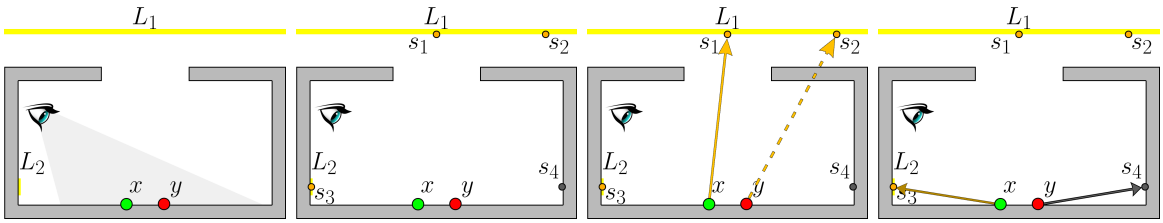
6.1 All About Importance

Importance, as we briefly discussed in section 2.3.2, is a term to indicate how much a region contributes to the final result. The more contribution there is, the higher the importance. From a statistical perspective, importance sampling is about determining certain properties of one distribution using another distribution. In our case, global illumination, where we are using Monte Carlo integration techniques to solve the equation, we want to sample as proportional to the final contribution as possible.

Most of the time, importance sampling for a 1-D variable in rendering is done by using a random number ξ that is uniformly distributed in range $[0, 1]$ to inversely sample a CDF, which is constructed by accumulating a desired PDF. In discrete cases, corresponding terms are *cumulative mass function*, CMF and *probability mass function*, PMF.

Recall in previous chapters when we use certain terms as importance, we are all dealing with discrete cases. We build certain PMFs, and consequently CMFs, from finitely many cache records, and normalize the resulting histogram-like information. In BSDF importance sampling, we sample according to the analytical expression of function, i.e. $p \propto f_s$. In photon driven importance sampling (see section 4.4.3), we pick $p \propto f_s d^2\Phi/dA$, where Φ is the information cached in a photon map. In visibility caching for direct illumination (see section 5.5.2), we pick $p \propto Lf_sG\widetilde{V}$, where $\widetilde{V}$ is the information cached in a visibility cache.

It is now clear that we can use any term that appears in the rendering equation as a guidance for importance sampling, analytical or not. Now we categorize samples using statistical jargon. We consider those samples that are sampled with decent probability as positive, those with low probability as negative.



(a) Scene setup (b) Four samples (c) TP and FP (d) TN and FN

Fig. 6.2: (a) Two area lights, two shading points. The bigger light L_1 lies outside the box, the smaller light L_2 lies inside the box but close to ground. Viewpoint is inside the box. Two shading points x and y located in the frustum. (b) Four samples. s_1 and s_2 lie on L_1, s_1 is visible to x and y but s_2 is occluded to both. s_3 lies on L_2. s_4 lies in the box, close to ground, but not shaded (due to limited depth). (c) Yellow arrow: true positive. Yellow dotted arrow: false positive. (d) Dark yellow arrow: false negative. Grey arrow: true negatives.

True Positives

True positive samples refer to those samples that are sampled with a decent probability according to a PDF and turn out to make decent contribution. One example is shown in Fig. 6.2c. Intuitively, TP samples are desirable samples.

False Positives

False positive samples refer to those samples that are sampled with decent probability but turns out to contribute little to the final result. This kind of sample is usually the source of dark pixels in the noise. One example is shown in Fig. 6.2c, where s_2 is sampled with good probability but is occluded.

True Negatives

True negative samples refer to those samples that are sampled with a low probability and turn out to contribute very little to the final image. This kind of samples usually would not cause huge amount of variance, since the low contribution will be divided by the low probability. One example of such sample is shown in Fig. 6.2d. TN samples are acceptable samples.

False Negatives

False negatives refer to those samples that are sampled with a low probability but turn out to have decent contribution. This kind of samples is the main source of the bright spots in the noise, as the decent contribution will be divided by the very low probability and the variance will explode. One

example is shown in Fig. 6.2d, where s_3 is sampled with very low probability but turn out to have decent contribution.

As discussed above, the majority of the excessive noise that we can perceive comes from false samples. False negative samples are particularly undesirable. To avoid this, we would like an importance sampling technique that could better handle all cases in general. Next, we present the importance caching algorithm proposed by Georgiev et al. to attach this problem.

6.2 Algorithm Overview

The original algorithm by Georgiev et al. was designed in the context of *Many-light rendering*, of which *instant radiosity* algorithm by Keller [20] is a major choice. We first briefly present instant radiosity and many-light rendering in section 6.2.1, and then discuss a importance driven distribution of *virtual point light*s, VPLs, in section 6.2.2. In section 6.2.3 we briefly overview the rest of the algorithm.

6.2.1 Instant Radiosity and Many-light Rendering

Many-light rendering is a bidirectional global illumination technique that was pioneered by instant radiosity. The idea is to use finite amount of virtual lights to simulate otherwise infinitely many light samples. Instant radiosity is a two-pass algorithm, of which a general overview of instant radiosity algorithm goes as follows.

In the first pass, sample certain amount of points on light sources and perform light tracing into the scene. At each non-specular bounce, create a VPL with its illumination property determined by incoming radiance and surface property. Certain amount of VPLs are created on light sources as well. Stop light tracing at certain depth. In the second pass, trace rays from the camera into the scene. At first non-specular hits, connecting shading points with all VPLs. The resulting radiance is the sum of contributions from all N VPLs, and the formulation is given as

$$\widehat{I}_{ml}(\mathbf{x}_i) = \sum_{k=1}^{N} L(\mathbf{x}_i \leftarrow \mathbf{v}_k) f_s(\mathbf{x}_{i-1} \leftarrow \mathbf{x}_i \leftarrow \mathbf{v}_k) G(\mathbf{x}_i \leftarrow \mathbf{v}_k) V(\mathbf{x}_i \leftrightarrow \mathbf{v}_k) \tag{6.1}$$

The nature of using finite amount of direct illumination to approximate global illumination makes many-light rendering a viable choice for many interactive applications. Few of the advantages thereof include scalability and easy implementation on graphics hardwares using rasterization. However, many-light rendering does have its limitation. The most obvious limitation is that it is a biased technique. Another exemplary disadvantage is the strong correlated sampling as a consequence of reusing a same set of VPLs. Please refer to the *STAR* report on the same topic by Dachsbacher et al. for more details about instant radiosity and many-light rendering [6].

For the purpose of a better algorithm overview, the pseudo code for importance caching is now given in Algorithm 10. Detailed pseudo code is given in respective sections later.

6.2.2 Importance-driven VPL Distribution

In standard instant radiosity, VPLs are distributed using light tracing. For a fixed amount of records, this will result in evenly distributed VPLs according to true illumination distribution. However, this might not be optimal. For scenes with difficult visibility, the majority of VPLs will probably lie outside the visible region seen from the camera.

Importance-driven VPL distribution was proposed by Georgiev et al. as an extension to standard VPL distribution process [12]. The extension is to first run few pilot VPL distributions and render

Algorithm 10 Pseudo code for Importance Caching

```
1: procedure ImportanceCaching(scene)
2:     V ← DistributeVPL(scene)
3:     IC ← BuildImpCache(scene, V)
4:     FinalRender(scene, V, IC)
```

Algorithm 11 Pseudo code for VPL distribution

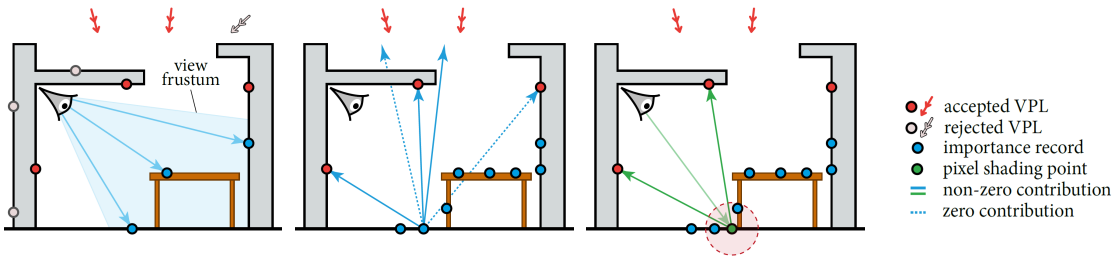
```
1:  procedure DistributeVPL(scene)
2:      V ← ∅
3:      V_p ← Run a few pilot VPL distributions
4:      Φ_v ← Average contribution for each VPL in V_p
5:      Start light tracing, for each sample:
6:      while Random walk not terminated do
7:          if Hit non specular surface then
8:              Φ_i ← Calculate contribution of current v_i
9:              p_i = min{ Φ_i/Φ_v + ε, 1 }
10:             if RussianRoulette(p_i) then
11:                 v_i.flux ÷= p_i
12:                 V ←(insert)— Create a VPL using updated v_i
13:                 Continue random walk
14:             else
15:                 Terminate random walk
16:         else
17:             Continue random walk
18:     return V
```

a low resolution image to get an estimate of average VPL contribution Φ_v. Next, during the light tracing pass of VPL distribution, rather than simply accept a bounce v_i as a VPL, a Russian roulette is used to determine whether current record v_i is accepted or not. The acceptance probability p_i is determined by $p_i = \min\left\{\frac{\Phi_i}{\Phi_v} + \epsilon, 1\right\}$, where Φ_i is the contribution of current candidate to v_i the image plane as one big pixel and ϵ is a tiny value to avoid zero probability. If records are rejected, current random walk of light tracing will simply be terminated. Otherwise, flux carried by the accepted records will be divided by p_i as what other techniques do when using Russian roulette to ensure unbiasedness.

The process is summarized in Algorithm 11 and Fig. 6.3a.

6.2.3 Caching Importance and Final Rendering

The idea of importance caching in the context of many-light rendering is to evaluate exact contribution at a sparse set of points in scene and cache them in a first pass. In the second pass, at each shading point, the importance distribution of current point is approximated by interpolating valid importance cache records nearby. When the approximated importance distribution is available, we can importance sample this distribution and continue with regular Monte Carlo rendering. This whole process is illustrated in Fig. 6.3. Next, we discuss the exact representations for importance, reasoning behind them and ways to calculate them.



(a) (b) (c)

Fig. 6.3: (a) Importance-driven VPL distribution. Red dots are accepted VPLs. (b) Evaluate contributions from all VPLs at each sample of a sparse set of points, blue points. Create importance records at these points. (c) Rendering with importance cache. At each shading point, green point, locate usable importance cache records, blue points within the circle. Importance sample according to the interpolation of these cache records. (Images courtesy of Georgiev et al.)

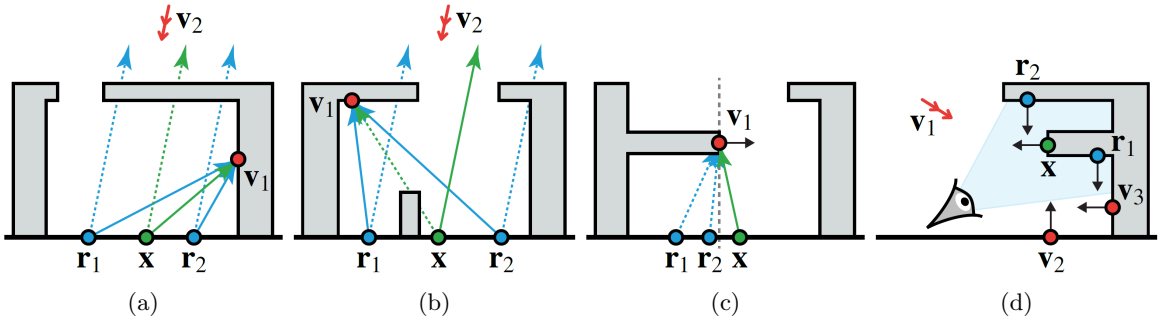


Fig. 6.4: Four cases for importance distribution. (a) Full distribution. (b) Un-occluded distribution. (c) Bounded distribution. (d) Conservative distribution. (Images courtesy of Georgiev et al.)

6.3 Importance Representation

From what we discussed above, we can see that an importance cache is a normalized histogram, with each bin representing the contribution ratio of a respective VPL. Intuitively, this contribution is the true and full contribution of each VPL, i.e. $L_i f_s G_i V_i$. However, due to the interpolation nature, which assumes smooth variation and strong correlation, this scheme might fail to work well at discontinuities. To robustly handle these difficult cases, which are common, it might be helpful to think differently. Recall in section 6.1, we want TP and TN, FP is ok, but FN is hardly desirable. As we will shortly find out, a combination of the four terms in $L_i f_s G_i V_i$ makes result more robust.

6.3.1 *F*: Full Distribution

Full distribution is, by its name, the full contribution of each VPL, i.e. for each VPL, we compute $\boldsymbol{F} := L f_s G V$. This distribution performs best in cases where surfaces vary smoothly and the occlusion situation remains pretty much the same. See Fig. 6.4a, where two importance cache records $\mathbf{r}_1$ and $\mathbf{r}_2$ capture the true illumination at $\mathbf{x}$.

6.3.2 *U*: Un-occluded Distribution

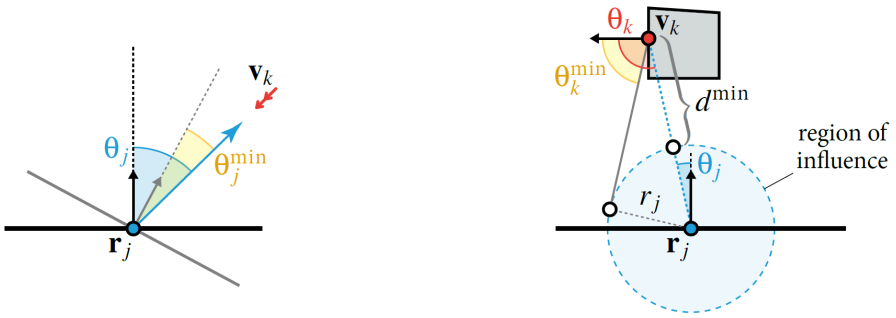
Discontinuities, as mentioned earlier, happen quite often. The main source of discontinuities is occlusion changes. See the case in Fig. 6.4b, where both $\mathbf{r}_1$ and $\mathbf{r}_2$ see $\mathbf{v}_1$ but not $\mathbf{v}_2$, which is the other way around for $\mathbf{x}$. In this case, if we sample according to full distribution, we will get a false positive, which is less desirable. Thus it would be helpful to take occlusion out of consideration, to some extent. The un-occluded distribution is the contribution of each VPL without considering the visibility term, i.e. $\boldsymbol{U} := L f_s G$.

6.3.3 *B*: Bounded Distribution

Sometimes for a shading point, nearby cache records will give low probabilities to some VPLs that are actually making a decent amount of contribution. This is the case of false negatives, which we want the least. One of such cases is caused by the geometry term, for which $G_{\mathbf{r}} < G_{\mathbf{x}}$, where $\mathbf{r}$ is cache record and $\mathbf{x}$ is a shading point. See the example in Fig. 6.4b, for which case if we sample $\mathbf{v}_1$ according to the distributions gathered from $\mathbf{r}_1$ and $\mathbf{r}_2$, resulting value will explode due to the very small probability.

False negatives caused by geometry term is handled by $\boldsymbol{B}$, the bounded distribution, which act as an extension to $\boldsymbol{U}$. For each record $\mathbf{r}_j$, we assign a valid region of influence with a radius r_j. Within the region, there are variations in position and normal orientation. Among all the range, we calculate the maximum value for the geometry term, G_j. Recall in equation 2.7, $G = V\frac{|\cos\theta\cos\phi|}{d^2}$. We take out the visibility term V as we did for $\boldsymbol{U}$, and the maximum geometry term is defined as

$$G^{max}(\mathbf{r}_j, \mathbf{v}_k) = \frac{|\cos\theta_j^{min}\cos\theta_k^{min}|}{(d^{min})^2}$$



(a) (b)

Fig. 6.5: Bounded distribution between cache record $\mathbf{r}_j$ and VPL $\mathbf{v}_k$. (a) For case when the virtual light is a virtual directional/ray light. The gray line indicate the surface with the smallest normal angle within the influence region of cache record $\mathbf{r}_j$. (b) For case when the virtual light is a point light on surface. (Images courtesy of Georgiev et al.)

where term θ_j^{min}, θ_k^{min} and d^{min} is defined in Fig. 6.5. For virtual lights with a surface point form, see Fig. 6.5b. For virtual lights with a directional light form, see Fig. 6.5a, where the gray arrow indicate the surface normal within influence region that has the smallest angle with the virtual light.

The bounded distribution is then defined as $\boldsymbol{B} := Lf_sG^{max}$. Now, for cases like Fig. 6.4b, for cache record $\mathbf{r}_2$, VPL $\mathbf{v}_1$ will be assigned a decent share of importance. Consequently $\mathbf{v}_1$ will be sampled with a decent probability at $\mathbf{x}$, thus avoiding the explosive noise. In practice, $\boldsymbol{B}$ can help find VPLs with small contributions.

6.3.4 $\boldsymbol{C}$: Conservative Distribution

For the worst case scenario, no records can provide useful information to a shading point. Such a case is shown in Fig. 6.4d, where non of the cache records are relevant for $\mathbf{x}$, much like the case in Fig. 6.4b. However, if sampled in the way as in un-occluded distribution, $\mathbf{v}_2$ and $\mathbf{v}_3$ are still very likely to be sampled while the only true option is $\mathbf{v}_1$.

For such cases with complex visibility, it is worth the expense to add another distribution that is uniform for all VPLs. The conservative distribution, $\boldsymbol{C}$, is thus defined as $\boldsymbol{C} := \frac{1}{\text{card}(VPLs)}$.

Now, the process of representing and calculating caches is clear. The whole process is summarized in Algorithm 12, where line 2 is pretty much identical to methods in previous chapters when we distribute points through out the scene.

Algorithm 12 Pseudo code for building Importance Cache

```
1:  procedure BuildImpCache(scene, V)
2:      R ← distribute cache candidates on non-specular surfaces in scene
3:      for all cache candidate r in R do
4:          f_s ← surface BSDF at r
5:          for all VPL v in V do
6:              L ← radiance contribution from v to r
7:              G ← |cos θ_r cos θ_v| / d^2
8:              G^max ← |cos θ_r^min cos θ_v^min| / (d^min)^2
9:              V ← shadow ray visibility test between v and r
10:             r.F ← L f_s G V
11:             r.U ← L f_s G
12:             r.B ← L f_s G^max
13:             r.C ← 1 / card(V)
14:     return R
```

6.4 Data Structure

Since the importance caching algorithm is actually a three-pass algorithm, two structures will be needed to hold data from the first two passes. For the first pass, where VPLs are distributed according to visual importance, the end result for later use is to answer indexed queries. For this reason, data structure for VPLs does not need to handle tasks such as range queries or nearest neighbors. Thus, an indexed array suffices to be the data structure for VPLs.

For the second pass, where importance cache records are created and stored, the end result will need to answer range queries and nearest neighbor searches. Given the experience we had from previous chapters, a k-d tree is a perfect choice.

6.5 Using the Importance Cache

The core idea of using importance sampling for many-light rendering is that sometime there might be too many virtual lights, explicitly evaluating them one by one is prohibitively expensive. Thus using our cached importance as a guidance, we importance sample a smaller set of whole VPLs. The estimator from equation 6.1 for this process is given as

$$\widehat{I}_{ml} \approx \sum_{k=1}^{M} \frac{L(\mathbf{x}_i \leftarrow \mathbf{v}_k) f_s(\mathbf{x}_{i-1} \leftarrow \mathbf{x}_i \leftarrow \mathbf{v}_k) G(\mathbf{x}_i \leftarrow \mathbf{v}_k) V(\mathbf{x}_i \leftrightarrow \mathbf{v}_k)}{p(\mathbf{v}_k|\mathbf{x}_i)} = \widehat{I}_{ic} \tag{6.2}$$

where $M < N$, representing the number of Monte Carlo samples used to approximate the full set of VPLs.

Before we can make proper use of existing caches, we need to locate them in the query structure. A similar weighting function as equation 3.7 is used, and the metric is defined as

$$d(\mathbf{x}, \mathbf{r}_j) = \|\mathbf{x} - \mathbf{r}_j\| + \lambda\sqrt{1 - n_{\mathbf{x}} \cdot n_{\mathbf{r}_j}}. \tag{6.3}$$

where λ is a user specified value to balance preference for distance or normal difference.

So far in each cache record, there are four distributions, with each one being a normalized histogram comprising N_{VPL} bins. One way to use these information is to interpolate. We can assign each record a weight according to their spatial distance and normal difference. However, this style of combination works on one dimensional values, in our case we are facing with an importance distribution matrix, see Fig. 6.6.

Georgiev et al. proposed a *bilateral combination* technique to take advantage of the strengths of each record [10]. We next briefly present the method. At last, we discuss the application of similar spirit in the context of PT [1] and BDPT [28].

6.5.1 Bilateral Combination of Cache Distributions

The bilateral combination is a two step operation on the distribution matrix. In the first step, an MIS estimator is used to combine cache records in rows. The rest four distributions in the combined one column distribution is then combined using another heuristic for final use.

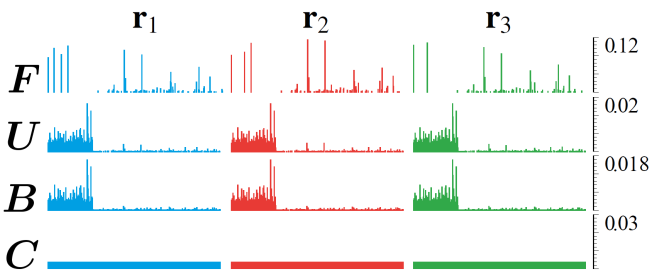


Fig. 6.6: Three importance cache records located near a shading point, with four distributions in a matrix layout. The goal of combining these distributions is to preserve their ability to handle unique cases as mentioned in the previous section. (Images courtesy of Georgiev et al.)

Column Combination

For all four rows, viz. $1:\boldsymbol{F}$, $2:\boldsymbol{U}$, $3:\boldsymbol{B}$ and $4:\boldsymbol{C}$, the MIS estimator for row $i \in \{1,2,3,4\}$ is given as

$$\widehat{I}_{mis}(\mathbf{x}_o,\mathbf{x},i)=\sum_{j=1}^{M}\frac{1}{n_i}\sum_{k=1}^{n_i}w_{i,j}^{col}(\mathbf{x}_o,\mathbf{x},v_{i,j,k})\frac{L(\mathbf{x}\leftarrow v_{i,j,k})f_s(\mathbf{x}_o\leftarrow\mathbf{x}\leftarrow v_{i,j,k})G(\mathbf{x}\leftarrow v_{i,j,k})V(\mathbf{x}\leftrightarrow v_{i,j,k})}{p(v_{i,j,k}|\mathbf{x})} \tag{6.4}$$

where M is the number of cache records, n_i is number of samples that remains constant across rows, $\mathbf{x}$ is current scattering point and $\mathbf{x}_o$ is the out going point, i.e. viewed from $\mathbf{x}_o$.

The weight function uses the balanced heuristic from [38], which is given in the form of

$$w_{i,j}^{col}(\mathbf{x}_o,\mathbf{x},v_{i,j,k})=\frac{p_{i,j}(\mathbf{x}_o,\mathbf{x},v_{i,j,k})}{\sum_{l=1}^{M}p_{i,l}(\mathbf{x}_o,\mathbf{x},v_{i,l,k})},$$

thus equation 6.4 can be reduced to

$$\widehat{I}_{mis}(\mathbf{x}_o,\mathbf{x},i)=\frac{1}{n_i}\sum_{k=1}^{n_i}\frac{L(\mathbf{x}\leftarrow v_{i,k})f_s(\mathbf{x}_o\leftarrow\mathbf{x}\leftarrow v_{i,k})G(\mathbf{x}\leftarrow v_{i,k})V(\mathbf{x}\leftrightarrow v_{i,k})}{p(v_{i,k}|\mathbf{x})}. \tag{6.5}$$

Row Combination

Now we have for each row an estimated value, viz. $\widehat{I}_{mis}(\mathbf{x}_o,\mathbf{x},\boldsymbol{F})$, $\widehat{I}_{mis}(\mathbf{x}_o,\mathbf{x},\boldsymbol{U})$, $\widehat{I}_{mis}(\mathbf{x}_o,\mathbf{x},\boldsymbol{B})$ and $\widehat{I}_{mis}(\mathbf{x}_o,\mathbf{x},\boldsymbol{C})$. All we need to do is to combine these four values into one.

Weighted combination of four values can easily break the advantages each distribution has for handling specific situations. See Fig. 6.7a, where a combination using balanced heuristic p_b results in significant amount of variance. Thus it is better to take a step further and divide the whole domain and pick one suitable distribution for a specific situation instead of weighting all four distribution in the same way through out the domain.

α-max Heuristic

Georgiev et al. proposed a α-max heuristic to solve this problem. The α-max heuristic is expressed as

$$w_s^{\alpha}(x)=\begin{cases}1, & \text{if } w_i^{\alpha}(x)=0, \text{for } 1\le i<s, \text{ and } p_s(x)\ge \underset{s<i\le m}{max}\,\alpha_i p_i(x)\\ 0, & \text{otherwise,}\end{cases} \tag{6.6}$$

where x is sampled according to p_s and $\alpha_i \in (0,1]$ is a confidence value associated with current distribution.

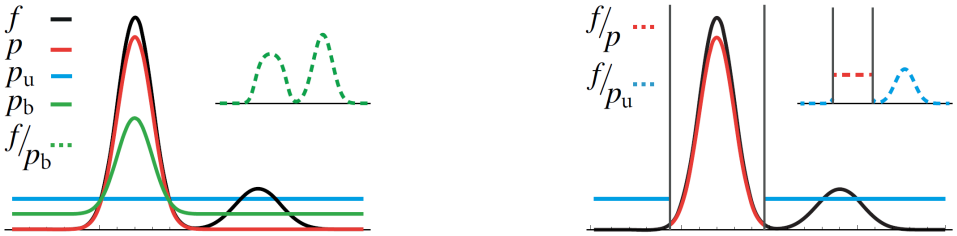
The α-max heuristic assigns probability according to which distribution has the highest value, thus reduce variance. Intuitively, this will partition the domain into several sub-domains, with each one dominated by one distribution. See Fig. 6.7b.

Few results of using importance caching from the original paper [10] are shown in Fig. 6.8.

6.5.2 Similar Applications in PT and BDPT

Bashford-Rogers et al. proposed to store the information of sampling BRDF and cosine term as *significance cache*, and use this significance cache to guide later scattering event [1]. The cache is represented as reflection lobe. Results show improved performance at difficult visibility scenarios and reduced variance comparing to regular sampling methods. While this method could also be categorized as *path guiding*, of which one popular way is to use photon map as a driver, this way of guiding paths shares the same spirit as importance sampling using importance cache.

Popov et al. proposed to use importance caching for the vertex connections in BDPT [28]. The method first trace light paths and eye paths into the scene. Then pick a uniformly distributed set of eye path and evaluate importance at each eye vertex of all pre-generated light paths and create a cache record with corresponding values stored in a normalized PMF. During rendering, eye paths are



(a) (b)

Fig. 6.7: (a) p_b is the result of balance heuristic, note the level of variance due to the averaging operation. A uniform distribution p_u ruins all the effort. (b) The α-max heuristic. The domain is partitioned into three parts, with each part only using the distribution with the max probability. Notice the variance reduction. (Images courtesy of Georgiev et al.)

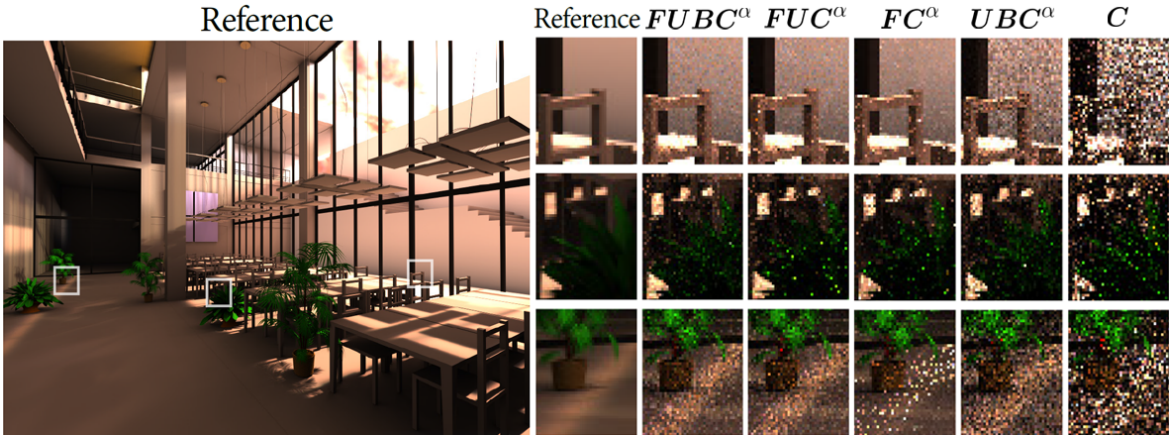


Fig. 6.8: Some results rendered using importance caching. The image is rendered using 35 VPLs per pixel. Different columns represent different strategies. Notice how the strength of each strategy is demonstrated in specific situations. The reference image is rendered using $\boldsymbol{FUBC}^{\alpha}$ in 1000 seconds, the rest images all use 20 seconds. (Images courtesy of Georgiev et al.)

generated as usual. When performing eye-light sub-path connections, at each eye vertex, nearby cache records are located and a combined PMF is calculated via interpolation. A importance sampled light path is picked by sampling according to this interpolated PMF. Results show significant improvement on variance reduction and speed up comparing to standard BDPT.

While various methods exist to make smarter choices when picking samples, what they have in common is to match probability with contribution. Caching, as a reliable technique, provides a glimpse of what the target might look like. Even though directly using caches results in correlation, using them as a hint guarantees unbiasedness. We cannot emphasize the importance of importance sampling, as it will always be the goal for Monte Carlo based rendering.

Chapter 7

Conclusion

In this survey, we studied various caching schemes in the field of global illumination. We studied irradiance caching and radiance caching, as they are the pioneering caching technique in this field. We studied photon mapping, of which the original algorithm essentially uses a caching scheme. We studied visibility caching, which tries to solve issues related to visibility tests. Finally we explored caching techniques that explicitly targeting at importance.

In each part, we present the problem, the formulation and theoretical foundations of using caches. We studied representation for the data to cache as well as data structures to hold these cache records. We studied different ways to make use of cached data.

It is very difficult to make a definite conclusion for caching techniques or for global illumination. Here we quote Dutré et al. from the book *Advanced global illumination* to conclude this survey, *the quest for realism and speed has not yet come to an end* [7].

Bibliography

[1] Thomas Bashford-Rogers, Kurt Debattista, and Alan Chalmers. "A significance cache for accelerating global illumination". In: *Computer Graphics Forum*. Vol. 31. 6. Wiley Online Library. 2012, pp. 1837–1851.

[2] Steve Brooks et al. *Handbook of Markov Chain Monte Carlo*. CRC press, 2011.

[3] Per H Christensen. "Faster photon map global illumination". In: *Journal of graphics tools* 4.3 (1999), pp. 1–10.

[4] Petrik Clarberg and Tomas Akenine-Moller. "Exploiting visibility correlation in direct illumination". In: *Computer Graphics Forum*. Vol. 27. 4. Wiley Online Library. 2008, pp. 1125–1136.

[5] Robert L Cook, Thomas Porter, and Loren Carpenter. "Distributed ray tracing". In: *ACM SIGGRAPH Computer Graphics*. Vol. 18. 3. ACM. 1984, pp. 137–145.

[6] Carsten Dachsbacher et al. "Scalable Realistic Rendering with Many-Light Methods". In: *Computer Graphics Forum*. Vol. 33. 1. Wiley Online Library. 2014, pp. 88–104.

[7] Philip Dutré, Philippe Bekaert, and Kavita Bala. *Advanced global illumination*. CRC Press, 2006.

[8] Philip Dutré, Eric P. Lafortune, and Yves D. Willems. "Monte Carlo light tracing with direct computation of pixel intensities". In: *3rd International Conference on Computational Graphics and Visualisation Techniques*. 1993, pp. 128–137.

[9] Elmar Eisemann et al. *Real-time shadows*. CRC Press, 2011.

[10] Iliyan Georgiev et al. "Importance caching for complex illumination". In: *Computer Graphics Forum*. Vol. 31. 2pt3. Wiley Online Library. 2012, pp. 701–710.

[11] Iliyan Georgiev et al. "Light transport simulation with vertex connection and merging." In: *ACM Trans. Graph.* 31.6 (2012), p. 192.

[12] Iliyan Georgiev and Philipp Slusallek. "Simple and Robust Iterative Importance Sampling of Virtual Point Lights." In: *Eurographics (Short Papers)*. 2010, pp. 57–60.

[13] Toshiya Hachisuka, Jacopo Pantaleoni, and Henrik Wann Jensen. "A path space extension for robust light transport simulation". In: *ACM Transactions on Graphics (TOG)* 31.6 (2012), p. 191.

[14] Toshiya Hachisuka, Shinji Ogaki, and Henrik Wann Jensen. "Progressive photon mapping". In: *ACM Transactions on Graphics (TOG)* 27.5 (2008), p. 130.

[15] Matthias Hollander et al. "ManyLoDs: Parallel Many-View Level-of-Detail Selection for Real-Time Global Illumination". In: *Computer Graphics Forum*. Vol. 30. 4. Wiley Online Library. 2011, pp. 1233–1240.

[16] Henrik Wann Jensen. "Global illumination using photon maps". In: *Rendering Techniques 96*. Springer, 1996, pp. 21–30.

[17] Henrik Wann Jensen. "Importance driven path tracing using the photon map". In: *Rendering Techniques 95*. Springer, 1995, pp. 326–335.

[18] Henrik Wann Jensen and Niels Jørgen Christensen. "Photon maps in bidirectional Monte Carlo ray tracing of complex objects". In: *Computers & Graphics* 19.2 (1995), pp. 215–224.

[19] James T Kajiya. "The rendering equation". In: *ACM Siggraph Computer Graphics*. Vol. 20. 4. ACM. 1986, pp. 143–150.

[20] Alexander Keller. “Instant radiosity”. In: *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co. 1997, pp. 49–56.

[21] Jaroslav Krivanek et al. “Radiance caching for efficient global illumination computation”. In: *IEEE Transactions on Visualization and Computer Graphics* 11.5 (2005), pp. 550–561.

[22] Dirk P Kroese, Thomas Taimre, and Zdravko I Botev. *Handbook of monte carlo methods*. Vol. 706. John Wiley & Sons, 2013.

[23] Eric P Lafortune and Yves D Willems. “Bi-directional path tracing”. In: *Proceedings of Third International Conference on Computational Graphics and Visualization Techniques (Compugraphics '93)*. 1993, pp. 145–153.

[24] Zhen Lin. *How to deduce the area of sphere in polar coordinates*. Nov. 17, 2011. URL: `math.stackexchange.com/questions/17850`.

[25] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Chapman & Hall/CRC, 1995.

[26] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation (3rd ed.)* 3rd. Morgan Kaufmann Publishers Inc., 2016.

[27] Stefan Popov et al. “Adaptive quantization visibility caching”. In: *Computer Graphics Forum*. Vol. 32. 2pt4. Wiley Online Library. 2013, pp. 399–408.

[28] Stefan Popov et al. “Probabilistic connections for bidirectional path tracing”. In: *Computer Graphics Forum*. Vol. 34. 4. Wiley Online Library. 2015, pp. 75–86.

[29] Philipp von Radziewsky et al. “Efficient Stochastic Rendering of Static and Animated Volumes using Visibility Sweeps”. In: *IEEE Transactions on Visualization and Computer Graphics* (2016).

[30] Tobias Ritschel et al. “Imperfect shadow maps for efficient computation of indirect illumination”. In: *ACM Transactions on Graphics (TOG)* 27.5 (2008), p. 129.

[31] Tobias Ritschel et al. “Making Imperfect Shadow Maps View-Adaptive: High-Quality Global Illumination in Large Dynamic Scenes”. In: *Computer Graphics Forum*. Vol. 30. 8. Wiley Online Library. 2011, pp. 2258–2269.

[32] Martin Šik et al. “Robust light transport simulation via metropolised bidirectional estimators”. In: *ACM Transactions on Graphics (TOG)* 35.6 (2016), p. 245.

[33] Brian Smits. “Efficiency issues for ray tracing”. In: *ACM SIGGRAPH 2005 Courses*. ACM. 2005, p. 6.

[34] Justus Ulbrich et al. “Progressive Visibility Caching for Fast Indirect Illumination”. In: *Proceedings of International Workshop on Vision, Modeling, and Visualization*. 2013.

[35] Eric Veach. “Robust monte carlo methods for light transport simulation”. PhD thesis. Stanford University, 1997.

[36] Eric Veach and Leonidas Guibas. “Bidirectional estimators for light transport”. In: *Photorealistic Rendering Techniques*. Springer, 1995, pp. 145–167.

[37] Eric Veach and Leonidas J Guibas. “Metropolis light transport”. In: *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co. 1997, pp. 65–76.

[38] Eric Veach and Leonidas J Guibas. “Optimally combining sampling techniques for Monte Carlo rendering”. In: *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*. ACM. 1995, pp. 419–428.

[39] Gregory J Ward, Francis M Rubinstein, and Robert D Clear. “A ray tracing solution for diffuse interreflection”. In: *ACM SIGGRAPH Computer Graphics* 22.4 (1988), pp. 85–92.

[40] Gregory J Ward and Paul Heckbert. “Irradiance gradients”. In: *Third Eurographics Workshop on Rendering*. Vol. 8598. 1992.

[41] Turner Whitted. “An improved illumination model for shaded display”. In: *ACM SIGGRAPH Computer Graphics*. Vol. 13. 2. ACM. 1979, p. 14.

[42] Lance Williams. “Casting curved shadows on curved surfaces”. In: *ACM Siggraph Computer Graphics*. Vol. 12. 3. ACM. 1978, pp. 270–274.