Global Illumination using Photon Ray Splatting

Robert Herzog, Vlastimil Havran,
Shinichi Kinuwaki,
Karol Myszkowski,
Hans-Peter Seidel

MPI–I–2007–4–007 May 2007

Authors' Addresses

Robert Herzog, Shinichi Kinuwaki, Karol Myszkowski, Hans-Peter Seidel
Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85
66123 Saarbrücken
Germany

Vlastimil Havran
Czech Technical University
Computer Science and Engineering
Prague
Czech Republic

Abstract

We present a novel framework for efficiently computing the indirect illumination in diffuse and moderately glossy scenes using density estimation techniques. A vast majority of existing global illumination approaches either quickly computes an approximate solution, which may not be adequate for previews, or performs a much more time-consuming computation to obtain high-quality results for the indirect illumination. Our method improves photon density estimation, which is an approximate solution, and leads to significantly better visual quality in particular for complex geometry, while only slightly increasing the computation time. We perform direct splatting of photon rays, which allows us to use simpler search data structures. Our novel lighting computation is derived from basic radiometric theory and requires only small changes to existing photon splatting approaches. Since our density estimation is carried out in ray space rather than on surfaces, as in the commonly used photon mapping algorithm, the results are more robust against geometrically incurred sources of bias. This holds also in combination with final gathering where photon mapping often overestimates the illumination near concave geometric features. In addition, we show that our splatting technique can be extended to handle moderately glossy surfaces and can be combined with traditional irradiance caching for sparse sampling and filtering in image space.

Keywords

Three-Dimensional Graphics and Realism - Rendering, Global Illumination, Photon Density Estimation, Filtering, Radiance Caching

Contents

Table 1: Main symbols used in this paper.

$x_0, x_1, ...$	Light path vertices (photon hitpoints, x_0 origin on light source)
$y_0, y_1, ...$	Eye path vertices (eye samples)
θ, ϕ	Polar angle and azimuthal angle relative to surface normal
ω	Incoming light directions (θ, ϕ)
ω'	Outgoing light directions (θ', ϕ')
$\cos\theta_x$	Cosine of angle θ between incoming direction and normal at x
M	Total number of photon hits recorded in the scene
N	Total number of eye path samples (i.e. computed illumination points on scene surfaces)
K	Number of nearest neighbors under the density estimation kernel
$\Phi(x, \omega)$	Incoming flux (photon) at x from direction ω
$L(x, \omega'), L(x, \omega)$	Radiance at x in direction ω', from direction ω, respectively
$E(x)$	Irradiance at x
h	Density estimation bandwidth (splat kernel radius)
C	Parameter determining the amount of noise in the density estimation
S	Parameter determining the variance of h
R	Maximum bandwidth scaling parameter, $R \in]0..1]$
$\mathcal{K}_h(x, y)$	Normalized 2D Kernel function at x with bandwidth h: $\mathcal{K}_h(x, y) = \frac{1}{h}\mathcal{K}\left(\frac{\|x-y\|}{h}\right)$
$\mathcal{K}_h(x, y, \omega)$	Normalized 2D kernel function ($\mathcal{K}_h(x, y)$) with domain oriented perpendicular to ω
$d\sigma(\omega)$	Differential solid angle for direction ω
$dA(x), \Delta A(x)$	Differential surface area at x, finite surface area at x
$dA^{\perp}_{\omega}(x)$	Projected differential surface area, $dA^{\perp}_{\omega}(x) = dA(x) \cdot \cos\theta$
$V(x, y)$	Visibility predicate, if x and y are mutually visible then $V(x, y) = 1$, else $V(x, y) = 0$
$f_s(x_{i-1}, x_i, x_{i+1})$	Bidirectional reflectance distribution function (BRDF) at x_i for scattering from x_{i-1} to x_{i+1}
$p(x)$	Probability density function (pdf)
p_s, p_e	pdf for sampling BRDF and light source emission, respectively
$p(y\|x)$	Conditional pdf for sampling light path vertex y given vertex x
$Y_l^m(\theta, \phi)$	Spherical harmonics (SH) basis function of degree m for band l with index $i = l(l+1) + m$
$c_l^m(\theta', \phi')$	BRDF SH coefficients representing the incoming hemisphere for the outgoing direction $\omega' = (\theta', \phi')$
λ_l^m	SH coefficients representing incoming radiance

1 Introduction

Many rendering applications used in industrial design and special effects in movie productions require high quality global illumination solutions, which are costly for complex scenes with general reflectance models. A common choice in such applications is the photon mapping algorithm [11], in which stochastic photon tracing is performed and the resulting photon hit points on the scene surfaces are registered in the photon map. The nearest-neighbor density-estimation method developed in statistics [25] is then employed to reconstruct the lighting function based on the photon map. Since a finite neighborhood is needed to collect a sufficient number of photons and to reconstruct the lighting function with an acceptable noise level, all density estimation methods are prone to a systematic error, so-called *proximity bias* [23,25] (due to a convolution of the original lighting function with the density estimation kernel). Photon mapping also suffers from other systematic errors: *boundary bias* (i.e., underestimation of illumination near object boundaries), *topological bias* [8, 23] (i.e., wrong estimation of the surface area on complex surfaces). See the examples in Fig. 1.1.

Recently, Lastra et al. [17] and Havran et al. [8] have shown that a viable alternative for the density estimation of photon hit points on the scene surfaces is an analogous operation performed directly for photon paths traveling in the proximity of these surfaces. To compute the irradiance value at a given surface point, its neighborhood is searched for photon rays that intersect a disc in the tangent plane, which is centered at this point (see Fig. 1.1c). The disc is extended until a minimum specified number of photon rays is found or a maximum disc radius is exceeded. This leads to elimination of boundary bias inherent to photon maps as well as a reduction of topological bias for convex surfaces, since density estimation is computed for a disc in the tangent plane and the real surface area (A in Fig. 1.1c) does not need to be estimated. The disadvantage of these methods is that they need complex and memory demanding data structures for nearest neighbor searches of rays. Their algorithms rely heavily on the coherence in the search queries

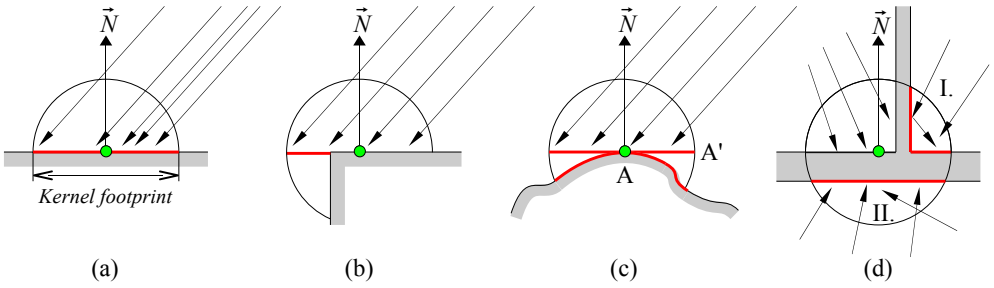


Figure 1.1: Bias sources in traditional photon density estimation: proximity bias (a), boundary bias (b), topological bias (c). Light leaking (d), due to wrong visibility assumptions, is a special case of proximity bias, which can be partially detected by back-face culling of incident photons (group II in (d)). Our algorithm eliminates boundary bias and topological bias.

and therefore only work for primary-ray hit points shot from the camera, which we will refer to as *eye samples*. Another drawback of these methods is that they cannot compute the correct tangent disc area for points on concave surfaces, for example, in corners where a disc is partially intersected (see for example Fig. 1.1d).

Direct rendering of lighting computed using either photon or ray mapping leads to poor image quality, which give only a vague idea of the rendered scene appearance and costly *final gathering* (the numerical integration of incoming radiance over the whole hemisphere) is performed for all pixels [11]. These costs can be reduced by using the *irradiance cache* data structure to interpolate irradiance samples sparsely in object space both for diffuse [34] and moderately glossy [13] surfaces.

The goal of this work is to provide a framework for quickly computing rendered previews of good quality, while also enabling the functionality of final gathering and irradiance caching if even higher quality and more robust results are needed. Our method shares all discussed benefits of ray density estimation, but neither requires the complex k-NN ray gathering as for the ray maps technique nor relies on the coherence of the eye sample positions. We replace the line density estimation in the tangent plane by energy splatting along photon rays to the eye samples. Our new density estimation metric is capable of handling illumination on complex geometric topology (e.g. wrinkled and bump mapped surfaces), which is not working in a direct visualization of photon maps. It also reduces the low-frequency noise, which is perceived as speckles in the final image (Fig. 7.3). Compared with standard photon density estimation, we obtain better image quality with the

same number of photons because more photons contribute to a pixel using the same kernel width, which reduces variance. Further combined with multi-stage filtering, our method leads to fast rendering of images with acceptable quality for low-frequency indirect illumination.

Additionally, we show how our method can be extended with state-of-the-art techniques in global illumination such as radiance caching and non-diffuse lighting on moderately glossy BRDFs. In the following section we review previous work concerning an operation central for our algorithm: photon energy splatting onto the image plane.

2 Previous Work

Photon mapping with direct photon splatting into the image plane has already been investigated by Lavignotte et al. [18]. They focus on off-line rendering and avoid the boundary bias by precise computation of the area covered by the splat footprint over each mesh element. On the other hand, they do not solve this problem for arbitrary topology. Since they only consider hit points of photons on meshes, their approach fails when it comes to estimation of the illumination on small surfaces. Furthermore, they allow only for diffuse scattering of light towards a camera. In order not to smear over object boundaries, they keep an item buffer with object IDs, which introduces further dependencies on a scene model.

Another interesting photon splatting approach has been subject in the research report of Bekaert et al. [1]. Their work aims at high-quality rendering of various kind of surface material and illumination conditions using photon density estimation. They achieve this by correcting the error in the neighborhood of a photon hit point, which is mainly caused by wrong visibility assumptions within the splatting footprint of the photon. However, their method needs to estimate the solid angle subtended by the splatting footprint at the photon's origin, which is generally difficult and therefore only coarsely approximated in their method.

Splatting has also been used in the context of different global illumination algorithms such as path tracing [6] and bidirectional path tracing [28], but there the main motivation was noise reduction. Suykens and Willems [28] propose an iterative procedure with adaptive filter-size control taking into account the density of samples and their energy contributions. This technique inspired us to control the splat size as a function of photon-path probability density including BRDF sampling density and number of photon bounces, which is neglected in gathering density estimation techniques only operating on a local neighborhood.

Splatting is also commonly used in recent GPU-based global illumination techniques, which are often designed for games and are usually limited to a

single bounce of indirect lighting for purely Lambertian environments. For example Dachsbacher and Stamminger [3] use an extended shadow map to deposit secondary light sources directly on lit scene surfaces and then splat energy from these sources to neighboring pixels without care of visibility in the indirect light. Splatting is also used to deposit lighting energy from reflections, refractions, and caustics [24, 29, 38]. In all those solutions the main goal is maximizing the rendering speed at the expense of reduced image quality.

Another class of algorithms are hybrid GPU-CPU off-line techniques that are designed for high quality rendering where the GPU role is merely to speed up some critical computation parts. Gautron et al. [7] use the GPU for the irradiance cache computation by rasterizing directly illuminated scene geometry (similar to [16]). In addition, the GPU is employed for splatting irradiance caches into the splat buffer (effectively the image plane).

Our method differs from previous splatting approaches in the way that a photon splats its energy along its path rather than in the neighborhood of its hit points only. This avoids boundary bias and reduces topological bias inherent to photon maps [11], which has been demonstrated in [8] and [17]. In contrast to [8] and [17] our photon ray splatting approach decouples the density estimation footprint entirely from the surface topology by applying a new density estimation metric over photon rays.

Our metric has also a similarity with *instant radiosity* [12] in the sense that a photon's energy contribution to a pixel is explicitly weighted by the cosine of the incoming photon direction. However, to be efficient, the remaining part of the geometric term including visibility and BRDF is stochastically sampled using a random photon walk with density estimation operating on a local neighborhood. This way we can still compute indirect glossy light transport (caustics) and handle a larger number of photons compared to instant radiosity.

3 Algorithm Overview

This section is intended to give an overview of the major processing steps in our photon-ray splatting architecture (refer also to Fig. 3.1). More detailed descriptions are provided in the following sections.

The basis of our method is a bidirectional path tracing algorithm combined with density estimation that samples eye and light paths (photons) whereby the eye paths are kept short to avoid the expensive final gathering through BRDF sampling. No indirect eye paths are sampled except for deterministic reflections (ideal specular surfaces). Instead, the main computation is carried out for the light paths via density estimation (photon splatting). The rendering algorithm consists of four passes:

Initialization: At the beginning of the algorithm the rendering and ray tracing system is initialized. This comprises scene parsing, construction of a ray tracing acceleration data structure, and optionally precomputing the coefficient tables of BRDF data in the spherical harmonics (SH) basis if the splatting is performed in the SH basis. For each BRDF the SH coefficients are precomputed for a constant number of discrete outgoing directions uniformly distributed over the hemisphere and stored in a coefficient lookup table [14].

Eye pass: After initialization primary rays are shot from the eye (camera) and *eye sample* records are stored at the hit points on the scene surfaces storing position, normal, BRDF index, incoming direction, pixel index, and weight for RGB components. In addition to the eye samples a *discontinuity buffer* [20, 32] for each pixel is maintained, which can be regarded as an extension to the classical z-buffer storing not only the nearest distance per pixel to the camera viewpoint but also the compressed normal in spherical coordinates. We use this buffer for discontinuity-preserving filtering in image space (Sections 6.2,6.3). Next, a kd-tree is constructed over the eye samples storing several eye samples per a leaf (see Section 5).

Light pass: After the eye pass the light pass starts with photon sampling. The photons are emitted from the light sources until the desired number of direct, caustics, and diffuse indirect photons are recorded. Since

the photon paths can be quite incoherent, all paths are stored and a kd-tree is constructed over the photon rays sorting the rays in spatial and directional domain (5D). Each photon ray is assigned a splatting kernel width, which is computed based on the entire photon path (Section 4.2).

Photon splatting: In the next phase all photons are splatted sequentially to neighboring eye samples in the vicinity of the photon rays (Section 4) using a density estimation kernel as described in Section 4.2. For an efficient nearest eye sample search along a photon ray the kd-tree over eye samples is traversed for a conical search domain (Section 5.2). In order to reconstruct glossy light transport at eye sample hit points, photon energy contributions are accumulated in an intermediate 4D data structure, which we call the *radiance map*, before being rendered to a final pixel. The radiance map consists of a number of *radiance images* with the same resolution as the final image, which represent spatial (pixel position) and directional (image index) information of incoming light.

As a comparison we have implemented two different methods for computing light transport towards pixels. In our first approach radiance is directionally discretized and accumulated in a directional histogram (*histogram splatting*) from which the pixel radiance is computed by BRDF importance sampling. In our second approach incoming radiance and BRDF are mapped onto the spherical harmonics (SH) basis (*SH splatting*). Except for pixels that cover specularly reflected eye samples, each pixel in the map corresponds to one camera-visible eye sample. In addition to the photon's energy contribution in the radiance map, a 2D image storing splat information per pixel is updated during the splatting phase. This image records the harmonic mean distance of incident photon rays and the sum of density-estimation kernel weights for each pixel and is used for determining the local filter size per pixel in each radiance image (Section 6.2) as well as the radiance cache spacing (Section 6.3).

Final integration with BRDF evaluation: When the light pass is finished, the radiance images can be prefiltered before the final image is composed through BRDF evaluation. The main purpose of the radiance image filtering is to speed up the algorithm and efficiently reduce noise in the radiance distribution due to photon splatting at the expense of increasing bias. Our filtering method is based on fast adaptive convolution in image space using a summed area table (SAT) that preserves discontinuities in the 2D filter footprint. The final image is then composed from the radiance images either in the spherical harmonics basis or in the primary domain via BRDF sampling. The main algorithm flow for the spherical harmonics splatting without radiance caching is illustrated in Fig. 3.1.

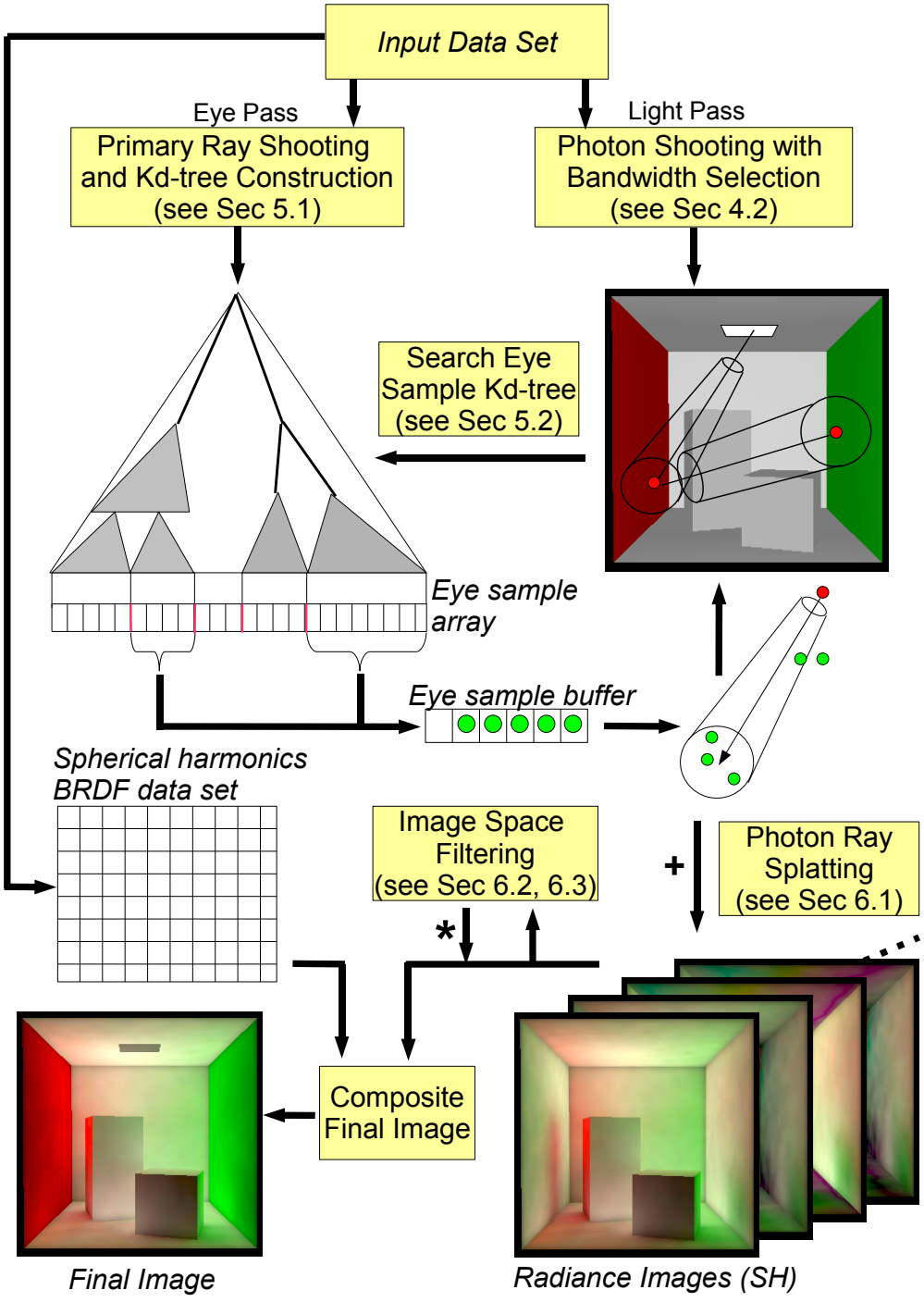


Figure 3.1: The processing flow in our photon splatting architecture (spherical harmonics ray splatting).

4 Photon Density Estimation in Ray Space

Standard photon-density estimation methods gather (or spread) photon energy in the neighborhood of a point inside a sphere using a normalized symmetric density estimation kernel [11, 25]. This ignores photon-path density changes (e.g. illumination gradients) in the neighborhood. Instead it is assumed to have an unbounded perfectly planar surface around the density estimation point y, which is only true in the limit for a differential surface area. To correct for this error, it is necessary to measure changes in path probability density (including visibility) with respect to corresponding photon hit point x_i. Such an approach has been proposed by Bekaert et al. [1]. However, their approach is computationally expensive and introduces a different sort of bias due to the approximation to the solid angle subtended by the density estimation footprint of a photon hit point. In our method, we partially correct the density estimation and still keep performance high. Let us first consider the probability density p_r for sampling the next photon hit point x_{i+1} from a particular photon location x_i. This pdf is proportional to

$$p_r(x_i \to x_{i+1}) \propto V(x_i, x_{i+1}) p_s^{\perp}(x_{i-1}, x_i, x_{i+1}) \frac{\cos\theta_{x_{i+1}}}{\|x_i - x_{i+1}\|^2}, \tag{4.1}$$

where $p_s^{\perp}(x_{i-1}, x_i, x_{i+1}) = p_s(x_{i-1}, x_i, x_{i+1}) \cdot \cos\theta'_{x_i}$ is the pdf for sampling the cosine weighted BRDF at point x_i. Refer to Table 1 for explanation of the symbols used in the paper.

Considering Eq.(4.1) we can draw some conclusions about the photon sampling density (i.e. flux density or irradiance) changes at hit point x_{i+1}. If we assume the density estimation footprint at hit point x_{i+1} to be relatively small with respect to the squared distance $\|x_i - x_{i+1}\|^2$ (such that visibility $V(x_i, x_{i+1}) = 1$ is likely within the footprint) and the BRDF sampling density p_s at x_i to be of low-frequency, then the change in photon sampling density in the neighborhood of x_{i+1} mainly depends on the change in surface orientation

$(\cos\theta_{i+1})$ in the vicinity of point x_{i+1}. In normal photon mapping this factor is simply ignored, assuming the density estimation domain is planar and continuous in the neighborhood of x_{i+1}, which results in visible bias in corners and on curved surfaces (see Fig. 1.1). We tackle this error by using a new density estimation metric that preserves the orientation when computing the contribution to each neighboring sample. Instead of computing density estimation for photon-ray intersections with surfaces as in photon mapping [11] and ray maps [8], we compute the density estimation in ray space.

Recall that radiance is a 5-dimensional quantity depending only on the position and direction in space. It is defined over differential projected area $dA^{\perp}_{\omega}(y)$ and differential solid angle $d\sigma(\omega)$

$$L(y,\omega) = \frac{d^2\Phi(y,\omega)}{dA^{\perp}_{\omega}(y)d\sigma(\omega)} = \frac{d^2\Phi(y,\omega)}{dA(y)\cos\theta\, d\sigma(\omega)}, \tag{4.2}$$

which is independent of surface orientation. However, it is measured on surfaces since radiant energy (photons) is absorbed and reflected on surfaces (our sensors). Inversely, in order to compute the radiance from the measured photon density on a surface with area $dA(y)$, the photon density is projected in the direction ω. This gives us the number of photons passing through a differential area $dA^{\perp}_{\omega}$ perpendicular to ω (see Fig. 4.1).

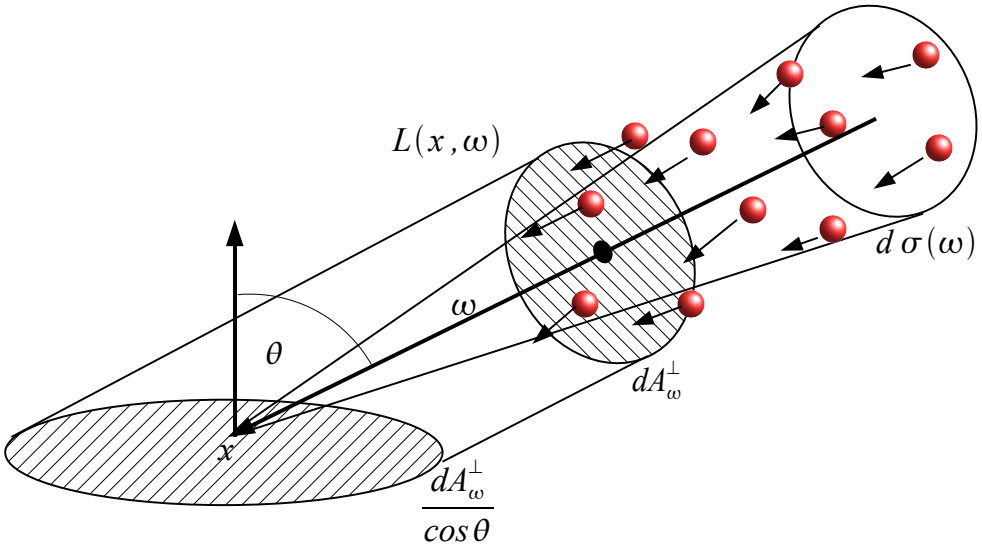


Figure 4.1: Radiance is defined as the incoming photon flux density per unit time in a projected differential area $dA^{\perp}_{\omega}$ and differential solid angle $d\sigma(\omega)$. Intuitively this can be understood as all the photons with direction in a certain solid angle crossing $dA^{\perp}_{\omega}$ per time unit.

Equivalently, if we know the radiance $L(y,\omega)$ for the differential area $dA^{\perp}_{\omega}$, we can compute its contribution to the irradiance of an arbitrarily oriented surface with area dA by multiplying $L(y,\omega)$ with the cosine of the angle θ between surface normal and ω (see Fig. 4.1).

To compute the irradiance $E(y)$ at a point y on a surface the incoming radiance is integrated over the upper hemisphere Ω^+ at y

$$E(y) = \int_{\Omega^+} L(y,\omega)\cos\theta\, d\sigma(\omega) = \int_{\Omega^+} \frac{d\Phi(y,\omega)}{dA(y)}. \tag{4.3}$$

Intuitively, this can be understood as accumulating the flux of all photons arriving in a small surface area around a point y, which leads us to the well-known photon mapping algorithm. In photon mapping we replace the differential quantities by finite ones and compute an approximation $\tilde{E}_1(y)$ to the real irradiance via density estimation

$$E(y) \approx \tilde{E}_1(y) = \sum_{i}^{K} \mathcal{K}_h(y,x_i)\frac{\Delta\Phi_i(x_i,\omega_i)}{\Delta A(y)}, \tag{4.4}$$

where $\mathcal{K}_h(x,y)$ is the density estimation kernel that satisfies $\int_S \mathcal{K}_h(x,y)dy = 1, \forall x$. Hence, in photon mapping we skip the computation of radiance by directly computing photon density on a surface. With this approach changes in surface orientation in the neighborhood of y are neglected in the density estimation.

To avoid this problem, we propose to estimate the photon density in ray space from the nearest photon rays and project the result onto the local surface to obtain the irradiance contribution. This requires only a small change to Eq.(4.3) to compute the irradiance as

$$\begin{aligned} E(y) &= \int_{\Omega^+} \frac{d\Phi(y,\omega)}{dA^{\perp}_{\omega}(y)}\cos\theta \\ &\approx \sum_{i}^{K} \mathcal{K}_h(y,x_i,\omega_i)\frac{\Delta\Phi_i(x_i,\omega_i)}{\Delta A^{\perp}_{\omega_i}(y)}\cos\theta_i, \end{aligned} \tag{4.5}$$

where K is the total number of photons arriving in the hemisphere centered at y. Combining Eq.(4.5) with a BRDF function to compute the reflected radiance $L(y,\omega')$ towards the camera leads to the more general description

$$L(y,\omega') \approx \sum_{i}^{K} f_s(\omega_i,y,\omega')\mathcal{K}_h(y,x_i,\omega_i)\frac{\Delta\Phi_i(x_i,\omega_i)\cos\theta_i}{\Delta A^{\perp}_{\omega_i}(y)},$$

where $f_s(\omega_i, y, \omega')$ is the BRDF at surface point y for incoming radiance direction ω_i and outgoing radiance direction ω'. $\mathcal{K}_h(y, x_i, \omega_i)$ is a normalized 2D Kernel function whose domain is oriented perpendicular to the direction ω_i. Hence, the kernel evaluates the distance r_i of point y to the ray (x_i, ω_i) rather than the distance to the ray-intersection x_i in the local tangent plane (see Fig. 4.2).

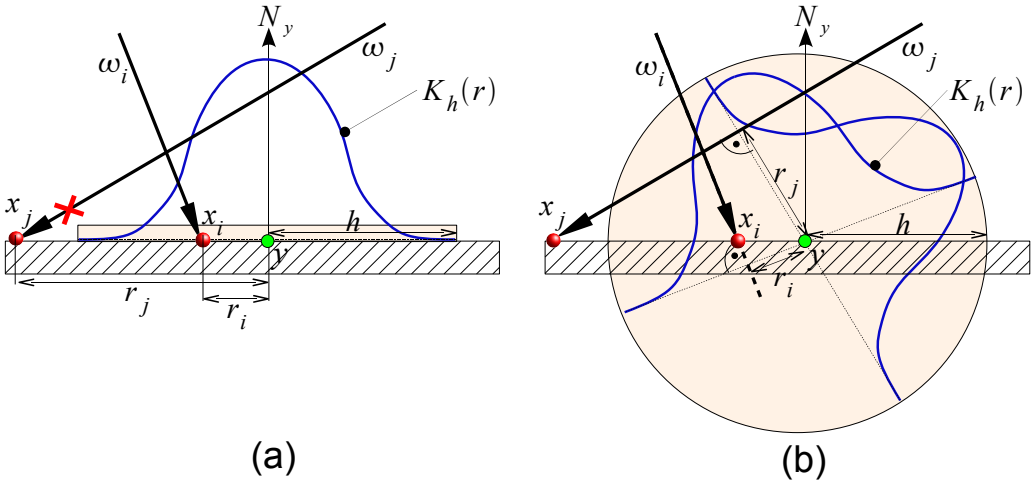


Figure 4.2: Photon density estimation via *ray gathering* visualized in 1D for two incoming photon rays: (a) when gathering in the tangent plane (ray disc intersection) only ray i contributes; (b) with our new metric, all rays intersecting the sphere with radius h centered around y contribute to the irradiance at y.

4.1 Photon Splatting Instead of Gathering

Instead of computing the photon density at an eye sample point y by *gathering* all neighboring photon rays, we can also utilize a *splatting* approach to estimate the photon density at all eye sample points. In splatting methods a photon computes its contribution weighted by a normalized kernel to a number of eye samples at once. This corresponds to kernel density estimation (KDE) in statistics [25].

Variable KDE with adaptive kernel width is preferable over k-nearest neighbors (K-NN) density estimation [25]. This is intuitively clear if we consider a large density gradient (e.g. shadow boundary): in the case of K-NN density estimation the filter kernel will expand to a large neighborhood in the low density region (shadow) "stealing" energy from the high

density region, which results in blurred slowly vanishing illumination. In case of variable KDE with a kernel width proportional to the local photon density, photons in low density regions spread their energy in a wider kernel increasing the error in the high density regions (which is however relatively small). Conversely, the high density photons spread their energy in a narrow kernel preventing strong light leaking into the shadow region, which results in "sharper" gradients particularly noticeable in density estimation for direct illumination and caustics.

If we consider now a constant bandwidth for kernel $\mathcal{K}$, then searching at each eye sample point all the photons that intersect the bounding sphere (see Fig. 4.2b) is equivalent to searching for each photon all eye sample points in the cylinder centered along the photon's ray (x_i, ω_i). However, instead of a cylinder we use a conical frustum as search domain because it is better suited for our bandwidth selection scheme proposed in Section 4.2. Each photon splats its energy weighted by a 2D kernel, which is aligned with its ray direction, to the found eye samples (see Fig. 4.3).

In practice the difficulty in this approach is to define where to end the splatting traversal, in particular for rays arriving at a grazing angle. We propose a simple heuristic: for eye samples located beyond the photon hit point the splatting footprint is reduced to a hemisphere (Fig. 4.3). This heuristic simplifies the search and prevents excessive light leakage as we do not evaluate the visibility for eye samples within the splatting footprint. Nonetheless, it can increase the noise since the splatting radius reduces gradually (potentially to zero) at the end of the cone. Therefore, the bandwidth of the density estimation kernel should stay above the minimum radius to avoid occasional noise artifacts. In practice, this noise is hardly visible in the indirect illumination as it only affects rays arriving at a grazing angle, which have low contribution.

Naturally, a photon can only contribute to eye samples that are oriented towards the photon ray (i.e. have a negative dot product of normal and ray direction).

More details of the photon ray splatting algorithm are presented in Algorithm 2. An example of the whole concept is visualized in Fig. 4.3.

4.2 Choice of Splat Kernel and Bandwidth Selection

According to statistical studies [25], the shape of the kernel $\mathcal{K}$ is rather unimportant for the bias reduction in density estimation. Therefore, we

have used a computationally efficient 2D kernel function: the *Epanechnikov* kernel [23, 25]:

$$\mathcal{K}(t) = \begin{cases} \frac{2}{\pi} \cdot (1 - t^2) & |t| < 1 \\ 0 & otherwise. \end{cases} \tag{4.6}$$

As an alternative we have also tested the *biweight* or quartic kernel [25], which is continuous at the boundary but increases the low-frequency noise.

A more important issue in density estimation is the choice of the *kernel width* or *bandwidth* (see [25]). The optimal width depends on the kernel function, the total number of samples, and the local fluctuations in the density we want to estimate (i.e. the second derivative of the irradiance function). The latter is difficult to estimate and consequently often replaced by various heuristics. However, in computer graphics, we are often more interested in computing results with low variance rather than noisy images. Therefore most photon mapping algorithms are based on k-nearest neighbors (K-NN) density estimation where the bandwidth is directly related to the local density of the samples. K-NN density estimation only attempts to reduce variance careless of introducing bias, which may result in strongly "blurred" images in particular for caustics.

For photon splatting, it is difficult to have a bandwidth selection proportional to the local sample density, which is not explicitly known during photon tracing. What we know is the path density and the contribution of individual photon paths. In case of perfect BRDF importance sampling ($p_s^\perp \propto f_s \cdot \cos\theta'$) and light source sampling proportional to its energy contribution ($p_e \propto L_e$), the photons are distributed according to the irradiance function and have all the same power. Intuitively, this means we sample more densely in the domain of the path space where the radiance contribution is high and sample more sparsely where it is low. In such a case, the photons are distributed according to the density

$$p(X) = p_e(x_0, x_1) g(x_0, x_1) \prod_{i=1}^{n-1} p_s^\perp(x_{i-1}, x_i, x_{i+1}) g(x_i, x_{i+1}), \tag{4.7}$$

where X denotes the full light path, x_i the i-th vertex of the path. The geometric density $g(x_i, x_{i+1}) = V(x_i, x_{i+1}) \frac{\cos\theta_{x_{i+1}}}{\|x_i - x_{i+1}\|^2}$ including visibility is inherently solved by the ray tracing operator.

Based on Eq.(4.7), we relate the bandwidth $h(x_i)$ to the path density of a photon, which has some desirable bias reduction properties. First, photons from a small number of bounces obtain a smaller bandwidth better preserving shadow boundaries and high illumination gradients while photons of multiple bounces spread their energy in a larger area, which reduces variance

(i.e. low-frequency noise). Second, caustic photon paths yield a high path density since BRDF sampling density is high resulting in a relatively small bandwidth. Suykens et al. [28] use a similar metric for computing the density estimation bandwidth for filtering samples in a bidirectional path tracer. They suggest using a bandwidth that is inversely proportional to the square root of the estimated function value at the sample location. To accommodate for different weights due to multiple importance sampling, the bandwidth is also scaled proportional to the square root of the sample weight, which is determined by the path sampling densities.

Since we do splatting in ray space with projected area measure, the path density is independent of the surface orientation at x_{i+1} and the cosine term $\cos\theta_{x_{i+1}}$ cancels out. Moreover, the path density $p(x_i|x_{i-1})$ can be arbitrarily small and arbitrarily large due to the distance term $\|x_i - x_{i+1}\|^2$ and we need to clamp it before being used in the bandwidth selection:

$$\tilde{p}(x_{i+1}|x_i) = \begin{cases} \frac{p_e(x_0,x_1)}{D(x_0,x_1)} & i = 0 \\ \tilde{p}(x_i|x_{i-1}) \cdot \frac{p_s^{\perp}(x_{i-1},x_i,x_{i+1})}{D(x_i,x_{i+1})} & i > 0, \end{cases} \tag{4.8}$$

where $D(x,y) = \max(\tilde{D}^2, \|x-y\|^2)$ is the squared length of the photon ray clamped at a scene dependent distance threshold $\tilde{D}^2$. Bounding the geometric term is also commonly applied, in a different context, to instant radiosity algorithms [12].

Using the bounded path probability density defined in Eq.(4.8), we compute the bandwidth $h(x_i)$ per photon ray by the following heuristic

$$h(x_i) = \frac{C}{\sqrt[6]{M}} \frac{w}{\sqrt{\tilde{p}(x_i|x_{i-1})^S}}, \forall i > 0, \tag{4.9}$$

where $h(x_0) = 0$, C is the user defined "smoothness" parameter, and $S \in]0..1]$ is the user defined bandwidth sensitivity controlling the variance of $h(x_i)$ (if S is set to 0, the bandwidth $h(x_i)$ is constant for all rays).

According to the optimal bandwidth for minimizing the mean integrated square error, $h(x_i)$ should be inversely proportional to the sixth root of the total number of samples M [25]. In our implementation the number of stored photons M is set for direct, indirect diffuse, and caustics photons separately. The square root comes from the fact that $\tilde{p}$ is related to the area rather than the radius of the splat footprint. The normalization coefficient w is automatically precomputed in an initial pilot shooting phase, which estimates the mean $\bar{r}$ of the term $r = 1/\sqrt{\tilde{p}(x_i|x_{i-1})^S}$. Coefficient w is then computed as

$$w = m_0 \frac{\mu_C}{\bar{r}}, \tag{4.10}$$

where the scene size dependent parameter

$$\mu_C = a \cdot \bar{D}(x, y) \tag{4.11}$$

is computed from the average path segment lengths $\bar{D}(x, y)$ also estimated in the pilot shooting phase. We set the constant $a = 0.2$, and $m_0 = \sqrt[6]{10^5} \approx 6.8$ functions as a calibration factor for $h(x_i)$ such that the mean bandwidth $\bar{h} = C \cdot \mu_C$ for $M = 10^5$. For the sake of robustness, the resulting bandwidth $h(x_i)$ is clamped at a minimum and maximum boundary value derived from a user defined maximum bandwidth scaling $R \in]0..1]$ such that for all rays $\frac{C}{\sqrt[6]{M}} \cdot \mu_C \cdot R \leq h(x_i) \leq \frac{C}{\sqrt[6]{M}} \cdot \mu_C / R$ holds.

Each photon ray $(x_{i-1} \rightarrow x_i)$ stores the initial bandwidth $h_0(x_i) = min\{h(x_i), h(x_{i-1})\}$ and the differential bandwidth per ray length $dh(x_i) = max\{0, h(x_i) - h(x_{i-1})\}/\|x_i - x_{i-1}\|$, which determines the angle of the conical frustum. Note that the bandwidth selection with all its parameters is defined in meters. For scenes defined in a different unit (e.g. feet, inches), we convert the ray length to meters and the bandwidth back to the scene unit. In Fig. 4.4, the precomputed bandwidth per photon splat is shown for 2 to 4 photon bounces (a - c) in a false-color mapping, where red corresponds to the smallest width and blue to the largest.

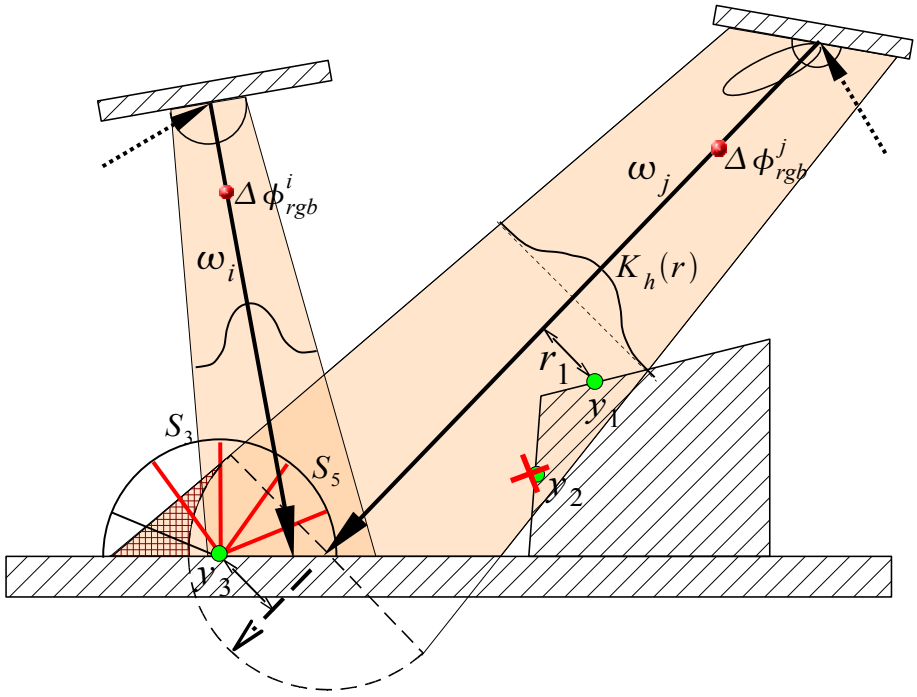
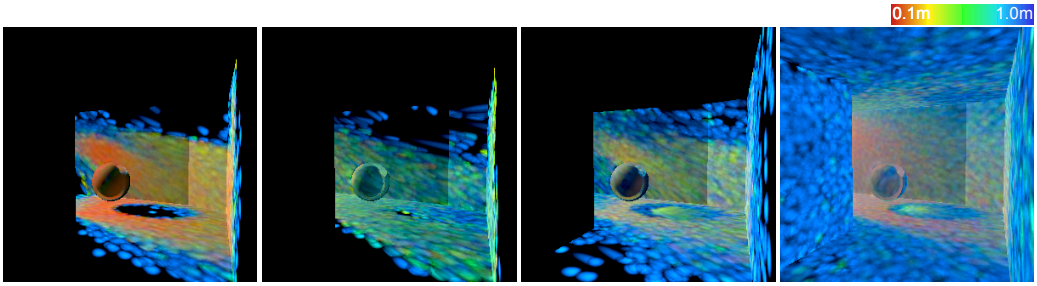


Figure 4.3: Density estimation via photon ray splatting to stored eye samples (y_1, y_2, y_3) in 1D. Photon j splats its energy within a certain kernel support (shaded area) of a function $\mathcal{K}(r)$ to y_1 and y_3, but not to back-facing y_2. The resulting energy contribution is either added to the spherical stratum whose discrete direction matches best the global incoming photon direction (histogram) or the contribution is added directly in the spherical harmonics basis to a number of radiance coefficients. In case of histogram splatting (see Section 6.1.1) photon ray i is mapped to stratum S_3 while photon j splats its energy to stratum S_5. Note that the splatting footprint (conical frustum) reduces to a hemisphere (dashed line) for eye samples beyond the photon hit point culling a part of the frustum (red cross-hatched region) as shown for ray j.



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

Figure 4.4: Color-coded splatting size (bandwidth) per photon in our simple test scene, red corresponds to minimum, blue to maximum bandwidth. (a) indirect photon-ray hits from one discrete direction for second bounce, (b) third bounce, (c) fourth bounce, and (d) all photon hits from all directions. For visualization purposes all photons splat their color-coded bandwidth in a small constant radius directly to the pixels. Note the small bandwidth associated to the indirect caustics photons passing through the glass sphere, which have a higher path probability density because of the two specular refractions in the glass sphere.

5 Efficient Nearest Neighbor Search Using a KD-Tree

All kernel density estimation methods require a search for the nearest neighbors at the sample points. Particularly in standard photon mapping [11], a search is performed for the k-nearest neighbor (K-NN) photon hit points in a sphere centered at each primary or secondary-ray hit point along the eye path. This is considered as the most time consuming operation in the illumination computation in particular for final gather rays. Therefore efficient hierarchical data structures were developed to query the nearest neighbors (NN) in sub-linear time. The probably most popular data structure for spatial searching of point data is the kd-tree. It enables searches for the K-NN in $\mathcal{O}(K + \log M)$ time complexity.

In the context of photon-ray splatting we face a similar problem. Previous approaches to photon-ray density estimation in the tangent plane are based on gathering the K-NN photon rays and need complex search data structures to manage the increased dimensionality of the ray data (5D) [8,17]. Since we utilize a splatting approach, we can still restrict our method to the classical and well-researched problem of searching point data.

We use a kd-tree over 3D points (eye samples), which can be constructed very efficiently. However, we need to modify the search for finding the nearest neighbor points in a volume associated with a photon ray. Rather than searching in a sphere as in normal photon mapping, we search the neighbor samples along the photon ray in a conical frustum with parameters depending on the photon's bandwidth as explained in Section 4.2.

5.1 The Splat KD-Tree Layout

We have chosen a standard axis-aligned kd-tree [9, 31] with splitting planes positioned at the spatial median of a node's associated bounding box or at

the sample point nearest to the spatial median if either half space is empty. The kd-tree is constructed from top to bottom until the termination criteria are met. The termination criteria are met if: the number of samples per node is smaller than 16 or the diagonal of the node's bounding box is smaller than a scene-size-dependent boundary threshold.

The kd-tree consists of four node types: *interior nodes*, *leaf nodes*, *empty nodes*, and *backface-culling nodes*. Each node uses 8 Bytes. The interior node encodes 1D splitting plane, offset to the right child node, node type and splitting axis. The left child node is always found at the next position $(index + 1)$ in the array of kd-tree nodes. In case of a leaf node, the 1D splitting plane encodes the number of elements and offset represents the index into the array of eye samples. Empty nodes (stopping nodes) are stored whenever the sub-tree does not contain any eye samples and must not be traversed any further. Additionally, we insert special nodes similar to [8] that we call backface-culling nodes. Such nodes allow for early culling of entire sub-trees containing infeasible backfacing eye samples with coherent normals. The difference to [8] is that we store not only the reference normal in the node but also the maximum angular deviation from the reference normal (see Fig. 5.1). This yields higher efficiency of successfully culling rays in particular on planar surfaces where the angular deviation of the normals is zero.

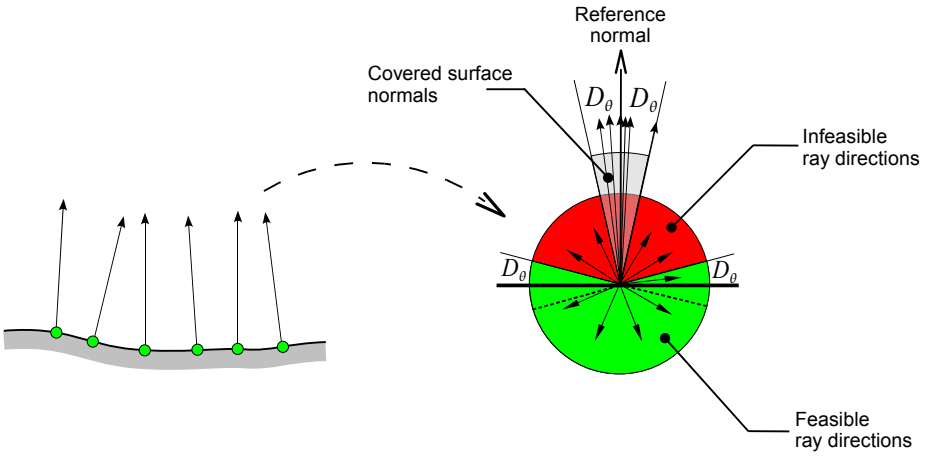


Figure 5.1: Clustering eye samples with coherent surface normals (left) by a directional node that stores the average normal and the angle D_θ of the normal bounding cone that contains all surface normals (right). All photon rays with a direction in the red range can be discarded conservatively since they are back-facing to all eye samples of the node.

We insert a backface-culling node into the tree if all eye samples in the current sub-tree have "similar" normals. Since testing for similarity during kd-tree construction is not for free, we only attempt to insert a backface-culling node if the number of samples per node is less than a maximum allowed threshold ($2^{0.5 \log N}$), and no backface-culling node has already been inserted above that sub-tree. If these criteria are met (then we have a high chance of finding coherent normals), we compute the average normal (the reference normal N_R) from all eye sample normals in the sub-tree and the maximum angular deviation D_θ from N_R (see Fig. 5.1). If D_θ is smaller than a constant threshold (15°), we insert a backface-culling node storing N_R (compressed) and the maximum feasible dot product ($C_{max} = \sin(D_\theta)$) of reference normal with incoming ray directions. The backface-culling node has only one child and does not subdivide space. Although using this node increases the kd-tree traversal depth by one and is also relatively expensive to traverse, we achieve a speedup of factor 1.2 to 1.3.

5.2 Photon Ray KD-Tree Traversal

The tree is traversed from top to bottom as in standard ray tracing algorithms with node traversal in 1D ray space. The difference is that we need to consider a volume associated with the ray. A kd-tree traversal algorithm for a similar problem, however in a different context, has been proposed by Dahmen [4]. He uses a kd-tree for accelerating the ray tracing of point data represented as oriented discs.

We start computing the minimum and maximum ray distance t_0 and t_1 by clipping the ray at the bounding box of the kd-tree extended by the ray's splat radius. Then we test t_0 and t_1 with the splitting plane of the current tree node. This plane is virtually moved to its left and right by the maximum splat radius R_1 of the photon ray. A child node needs to be traversed if a ray intersection with its virtual plane lies between t_0 and t_1. If the front-facing child node needs to be traversed, t_1 and R_1 are updated, respectively if the back-facing child node is scheduled for traversal, t_0 and R_0 are updated. This search algorithm is conservative but not optimal and can lead to unnecessary feasibility tests inside a leaf. For our tested scenes the average ratio of infeasible to feasible eye sample candidates found per ray traversal is between 27% and 52%, which depends on the ray's splat radius (the smaller the splat radius the more efficient becomes the search). Nevertheless, due to its simplicity the 1D-traversal algorithm performs better than accurate traversal algorithms in 2D [4]. The simplified pseudo code in Algorithm 1 describes the basic recursive version of the algorithm.

The complete traversal step of one interior node is shown in Fig. 5.2. There, the ray needs to visit both front and back child-nodes. Fig. 5.3 shows two examples where only the front respectively back half-space needs to be traversed.

Once the ray traverses a leaf node all eye samples associated with the node are tested for feasibility, i.e. distance to ray is smaller than splat radius, the normal of the eye sample is front facing, and its position is in front of the surface at the ray's origin. If feasible, the eye sample's weight is computed by the 2D kernel multiplied with the cosine between normal and ray direction according to Eq.(4.5).

In the proceeding splatting phase the photon ray splats its energy contribution to all pixels corresponding to the eye samples gathered during kd-tree traversal. The splatting is further described in Section 6.1

We also implemented a slightly different approach to the kd-tree search, where all traversed leaf nodes rather than individual eye samples are added to a queue of candidate leaves. The photon ray and its candidate queue are added to a shared buffer that is processed by a separate system thread which computes the actual density estimation testing each eye sample associated with a candidate leave for feasibility. This approach exploits better the modern computer architecture where the load can be shared between two CPU-cores. However, we achieved only a minor speedup ($\approx$ 15% to 20%) compared to the single threaded search since our search is relatively efficient compared to the time spent in photon splatting (Algorithm 2).

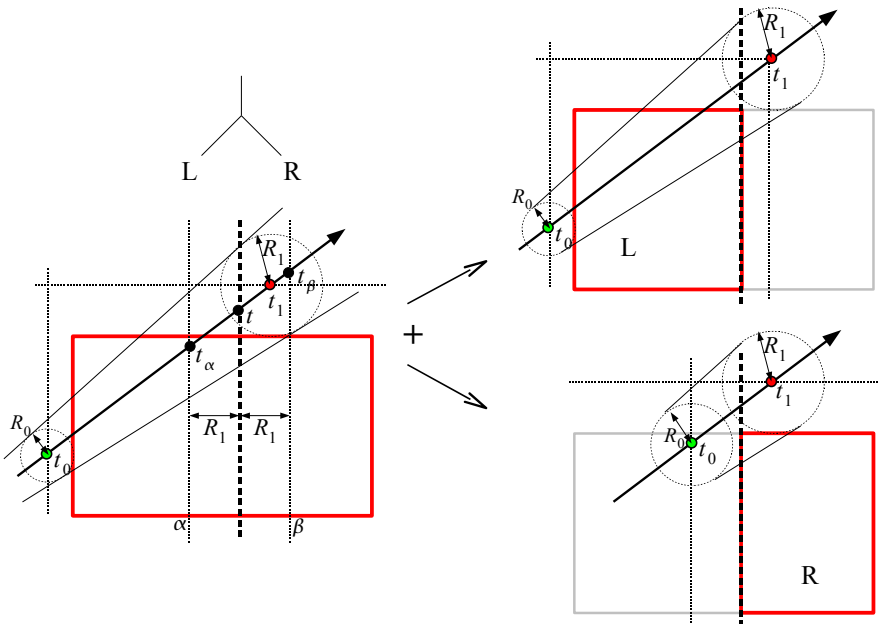


Figure 5.2: Traversal of a kd-tree node given the photon ray and its associated splat radius h. The dashed vertical line represents the 1D splitting plane and the thin dotted lines the virtual extensions of the node's corresponding voxel. The red frame depicts the currently traversed node. If $t_1 > t_\alpha$, we descend to the back child node. If $t_0 < t_\beta$, we traverse to the front child node.

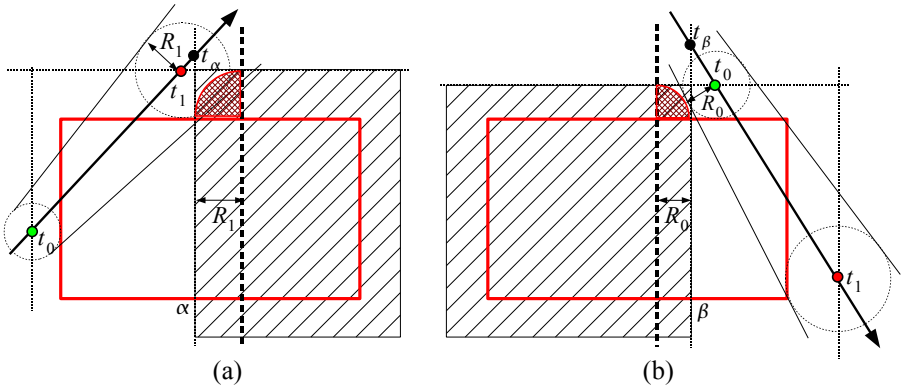


Figure 5.3: If $t_1 \leq t_\alpha$ only the front child node needs to be traversed (a). If $t_0 \geq t_\beta$ only the back child node needs to be traversed (b). Note that there is a small space near the corners where the ray traversal is not optimal. The ray only needs to traverse either half space if it intersects the indicated cross-hatched region. Due to the box approximation it may visit the front, back half-space respectively even if the conical frustum cannot intersect it.

Algorithm 1 KD-Tree Traversal

```
TraverseTree(ray, i_n, t_0, t_1)
  node := nodes[i_n]
  if (node.type = EMPTY) then
    return
  else if (node.type = LEAF) then
    test all samples in leaf and add to candidate list
    return
  else if (node.type = DIRCULL) then
    if (ray.dir • node.normal > node.C_max) then
      return
    else
      TraverseTree(ray, i_n + 1, t_0, t_1)
    end if
  else if (node.type = INTERIOR) then
    a := node.axis
    if (ray.dir[a] < 0) then
      invert order of traversal /* omitted here! */
    end if
    R_1 := ray.h_0 + ray.dh · t_1 /* compute splat radius at t_1 */
    /* compute ray length Δt between virtual plane and splitting plane */
    Δt := R_1 / ray.dir[a]
    t := (node.plane1D − ray.org[a]) / ray.dir[a]
    /* compute ray lengths t_α and t_β to virtual planes α and β */
    t_α := t − Δt
    t_β := t + Δt
    if t_0 < t_β then
      /* traverse front */ TraverseTree(ray, i_n + 1, t_0, min(t_β, t_1))
    end if
    if t_1 > t_α then
      /* traverse back */ TraverseTree(ray, node.offset, max(t_α, t0), t_1)
    end if
    return
  end if
```

6 Algorithmic Extensions

Like photon mapping [11], ray splatting is a very general method. It can be extended with many state-of-the-art techniques. Next we present some examples that we have implemented and tested.

6.1 Extension to the Directional Domain

First density estimation methods recorded photon flux in bins of a histogram, which has the advantage of low memory usage and fast rendering using graphics hardware. However, it is not capable of handling non-diffuse BRDFs. Jensen [10] showed that it is advantageous to keep the incoming direction of each individual photon in the *photon map*. With photon mapping it is possible to evaluate arbitrary BRDFs and render illumination on all kinds of surfaces with low-frequency BRDFs. Stürzlinger et al. [27] additionally combines the spatial density estimation kernel with a directional filter kernel to render moderately glossy illumination with photon density estimation.

In the spatial domain we use variable kernel density estimation as in standard photon splatting approaches [9, 18]. However, we consider not only the spatial domain of incoming photon flux but also the directional domain. For testing purposes we have implemented and practically evaluated two different methods for representing directional information of incoming light.

First method is based on a histogram approach. This means we accumulate flux on a surface in discrete directional strata with constant solid angle, which provides information about the average incoming radiance for a finite solid angle at a point on the eye path (see for example Fig. 4.3).

The second approach uses a different basis for representing incoming radiance in frequency space. We have chosen spherical harmonics (SH) [19, 21], since they are well suited for low-frequency signals over the sphere. First we describe the histogram approach and then the splatting in the SH basis.

6.1.1 Ray Histogram Splatting

To do density estimation in the spatial and directional domain even more photons are needed in order to have enough information for evaluating the BRDF at any point on a surface for a particular incoming direction. Since we only account for moderately glossy BRDFs, high angular frequencies in the incoming radiance are filtered by the BRDF [21]. Therefore a histogram of low resolution subdividing the incident sphere into P strata is sufficient if it does not undersample the BRDF.

We discretize the sphere to an icosahedron, which consists of 20 equally sized triangles. When subdividing it further to 80 triangles the efficiency of splatting decreases quickly while the memory consumption increases four times (80 images of screen resolution need to be stored).

Each stratum records incoming photon flux from a global discrete direction arriving in the neighborhood of the corresponding density estimation point (see Fig. 4.3). For postprocessing purposes the radiance for one global stratum is stored in one individual image for all eye samples such that each image corresponds to one discrete direction.

A directional filter kernel can be applied [27] such that each photon splats its energy to several neighboring strata with precomputed filter weights for P_D discrete photon directions, with $P_D \gg P$. However, this results in poor performance due to incoherent memory access. Therefore, we use a nearest neighbor approach where each photon contributes to only one stratum, the nearest neighbor stratum. The entire splatting algorithm for the general case with directional filtering is shown in Algorithm 2.

How do we benefit from the additional directional information? First, BRDF evaluation is simpler than for photon maps. We do not need to evaluate the BRDF (which can be expensive) for every photon with low contribution, but for the average accumulated radiance from a stratum of a discrete direction. Second, the filter kernel size for density estimation cannot only be adapted in spatial but *also* in the directional domain. Third, each radiance image can be adaptively filtered efficiently in 2D image space.

6.1.2 Ray Splatting in the Spherical Harmonics Basis

Using uniformly distributed strata over the hemisphere is efficient for computing the splatting, but on the other hand, it is not adaptive and can lead to aliasing artifacts if a BRDF contains too high frequencies. Therefore, we have also implemented a different approach using spherical harmonics (SH) basis functions to represent illumination as well as BRDFs by a small number of coefficients.

Algorithm 2 Photon Ray Splatting

SplatPhotonRay$(ray, candidates)$

$i_d :=$ *MapDirectionToStratum*$(ray.dir)$
$W :=$ *coeffTab*$[i_d]$
for all *eye samples (es) in candidates* ***do***
  $cos_\theta := ray.dir \bullet es.normal$
  if $(cos_\theta \geq 0)$ ***then***
    skip back facing sample
  end if
  $D_{es} := es.pos - ray.org$
  $z_n := D_{es} \bullet ray.normal$
  $z := D_{es} \bullet ray.dir$ */* compute projected distance along the ray */*
  $h := ray.h_0 + ray.dh \cdot z$ */* compute splat radius h (Section 4.2) */*
  $r_{es}^2 := \|D_{es} \times ray.dir\|^2$ */* compute squared distance to ray */*
  if $(z < 0)$ *or* $(z_n < 0)$ *or* $(z > h + ray.length)$ *or* $(r_{es}^2 > h^2)$ ***then***
    / outside the splat footprint → */ skip sample*
  end if
  if $(z > ray.length)$ ***then***
    $h^2 := h^2 - (z - ray.length)^2$
    if $(r_{es}^2 > h^2)$ ***then***
      skip sample
    end if
  end if
  $w := \frac{2}{\pi h^2} \cdot (1 - \frac{r_{es}^2}{h^2})$ */* evaluate Epanechnikov kernel */*
  $I := ray.flux * w \cdot \cos_\theta$ */* compute irradiance contribution */*
  / optionally compute irradiance gradient contribution, omitted here! */*
  $L := 0$
  / compute contribution to all directions using precomputed weights */*
  for $c = 0$ *to* P ***do***
    $L[c] := I * W[c]$
  end for
  / Add photon's contribution to radiance map */*
  UpdateRadianceMap$(es.pixel, L)$
  / Update harmonic mean distance and weights for filtering and radiance caching */*
  UpdateSplatImage$(es.pixel, w, 1/z)$
end for
return

The mapping onto the SH basis is entirely discretized. All BRDF data is initially precomputed for a number of discrete outgoing directions (θ'_o, ϕ'_o) mapped to SH coefficients, which are stored in a table [14]. The BRDF SH coefficients $c_l^m(\theta'_o, \phi'_o)$ for every outgoing direction are precomputed by evaluating the integral

$$c_l^m(\theta'_o, \phi'_o) = \int_{\Omega^+} f_s(\theta'_o, \phi'_o, \omega) \cdot Y_l^m(\omega)\, d\sigma(\omega). \tag{6.1}$$

Note that this can be simplified for isotropic BRDFs in particular for the Phong reflection model. Since we do the splatting directly in the SH basis, we also keep a table of precomputed real SH basis functions $Y_l^m(\omega_i)$ for P_D incoming ray directions ω_i. Since a photon contributes to many SH coefficients (and not only to the nearest-neighbor stratum in a histogram), we now keep one image storing the SH coefficients for the cosine-weighted incident radiance in each pixel. The radiance SH coefficients λ_l^m for a certain pixel are computed as

$$\begin{aligned}
\lambda_l^m &= \int_{\Omega^+} L(\omega) \cdot \cos\theta \cdot Y_l^m(\omega)\, d\sigma(\omega) \\
&= \int_{\Omega^+} \frac{d\Phi}{dA_\omega^\perp\, d\sigma(\omega)} \cdot \cos\theta \cdot Y_l^m(\omega)\, d\sigma(\omega) \\
&= \int_{\Omega^+} \frac{d\Phi}{dA_\omega^\perp} \cdot \cos\theta \cdot Y_l^m(\omega) \\
&\approx \sum_i^K \frac{\Delta\Phi_i}{\Delta A_{\omega_i}^\perp} \cdot \cos\theta_i \cdot Y_l^m(\omega_i),
\end{aligned} \tag{6.2}$$

where K is the number of neighboring photon splats contributing to the corresponding pixel. Note that we can also encode the cosine term $\cos\theta$ in the BRDF SH coefficients instead. However, then we would need at least 9 BRDF SH coefficients (3 bands) to represent diffuse cosine weighted BRDFs [21] with small error. For our strategy, only 1 coefficient (DC component) is needed for diffuse surfaces.

The photon ray splatting is then processed as follows. With each eye sample hit point we additionally store outgoing directions towards camera and normal in discrete spherical coordinates. Search and splatting is carried out as explained in Section 5. The difference is that the photon ray directly splats to the radiance SH coefficients of an eye sample. To do so the global incoming ray direction $\tilde{\omega}_i = (\tilde{\theta}_i, \tilde{\phi}_i)$ is rotated to local coordinates of the eye sample identified by its discretized normal (θ_N, ϕ_N). The real SH basis functions $Y_l^m(\theta_i, \phi_i)$ for the local incoming ray direction $(\theta_i, \phi_i) = Rot_{[\theta_N, \phi_N]}(\tilde{\theta}_i, \tilde{\phi}_i)$ are

looked up in the precomputed SH table for each eye sample included under the density estimation kernel. Each photon's radiance contribution is then scaled by the vector of SH basis functions and added to the radiance coefficients of the corresponding pixel in the SH radiance image

$$\lambda_l^m := \lambda_l^m + \mathcal{K}_{h_i}(y, x_i, \omega_i) \frac{\Delta\Phi_i(x_i, \theta_i, \phi_i) \cos\theta_i}{\Delta A_{\omega_i}^{\perp}} \cdot Y_l^m[\theta_i, \phi_i]. \tag{6.3}$$

The final pixel radiance is easily computed via a dot product of BRDF SH coefficients and radiance SH coefficients due to the orthogonality property of spherical harmonics.

$$\begin{aligned} L(\theta_o', \phi_o') &= \int_{\Omega^+} f_s(\theta_o', \phi_o', \omega) \cdot \cos\theta \cdot L(\omega) d\sigma(\omega) \\ &\approx \sum_{l=0}^{n} \sum_{m=-l}^{m=l} c_l^m(\theta_o', \phi_o') \cdot \lambda_l^m, \end{aligned} \tag{6.4}$$

where (θ_o', ϕ_o') is the local discrete outgoing direction.

6.2 Radiance Filtering in 2D Image Space

The ray splatting complexity depends linear on the search neighborhood and therefore on the size of the splat footprint. Instead of increasing the splat footprint, effectively the radius of the cone, we can also use a second pass filter in 2D image space to filter noise in the radiance images. To preserve discontinuities during filtering of the radiance images we only filter over geometrically continuous image region. In contrast to image processing approaches we have the geometric information behind all pixels available for free, which allows us to use a *discontinuity buffer* storing distance (length of primary ray shot from the camera) and normal per pixel [20, 32].

Even the simplest algorithm is still computationally expensive since it performs a convolution for a large number of pixels for all radiance images. The filter support can be large (up to 50×50 pixels). However, since we use a uniform kernel with rectangular support, we can significantly reduce the complexity of that algorithm by making use of a summed area table (SAT). One might argue that a filter with uniform kernel and rectangular filter support is too aggressive. However, first we do not filter visible radiance but indirect radiance for each direction or spherical harmonics band separately, which is only visible through BRDF modulation [21]. And second, the filter support for each pixel adapts to the number of photons that contributed to the pixel. Thus, the filter bandwidth is approximately inversely proportional

to the photon density per pixel similar as in the K-NN density estimation method.

A global SAT over the whole image does not suffice since we need to preserve discontinuities (e.g. edges). Thus, we construct local SATs over geometrically continuous image regions and separate those regions by applying a segmentation of the discontinuity buffer. We apply a seed-growing algorithm in image space for segmenting pixels that cover topologically continuous surfaces. Adjacent pixels are considered to be continuous if the second derivative (finite difference approximation) in primary-ray hit distance and the hit-normal deviation are below user-defined thresholds. Each segmented surface is assigned a unique ID which is stored in all its subtended pixels in the *discontinuity image*.

The algorithm has linear complexity and is efficiently computed. The outcome is the scalar discontinuity image I_{ID} where each pixel (i, j) is assigned an index k to the corresponding segment in the *segment list*. Each element in the segment list stores lower left corner (o_i, o_j), width, and height of the segment's bounding rectangle in image space.

In the next step we copy each pixel (i, j) of the currently processed radiance image to the corresponding *segment image* $I_S[k]$ at location $I_S[k](i - o_i, j - o_j)$ with index $k := I_{ID}(i, j)$. Pixels in the segment's bounding rectangle that do not belong to the segment are set to zero. Next we build the SAT over each segment image. The final filtering is then computed in constant time per pixel. This discontinuity segmentation is also exploited in the radiance caching scheme (Section 6.3).

6.3 Extension to Radiance Caching

In the following we show how our algorithm is extended to radiance caching in the spherical harmonics basis and exploits all benefits from the traditional caching scheme [7, 14, 34].

It is well known that (ir)radiance caching significantly speeds up computation of diffuse (glossy) indirect illumination computed with final gathering because the image plane is adaptively sampled [14, 34]. The cache sampling density adapts to an relative error estimate. The cache density in the original irradiance cache algorithm [34] adapts to the harmonic mean distance to the surrounding objects. The user provides a maximum error ϵ that determines the overall sampling density. However, choosing ϵ and the number of rays for final gathering is crucial and can lead to visible artifacts if set too relaxed. On the other hand, setting too conservative values may lead to long computation times with little progression in image quality.

Motivated by the radiance caching technique of Křivánek [14], we also perform the caching in the spherical harmonics basis. However, the difference lies in the illumination computation. While Křivánek [14] computes a high-quality solution of the incoming radiance by Monte Carlo final gathering of the incident hemisphere, we estimate the incoming radiance directly from neighboring photon-ray splats.

Our caching algorithm, initially intended to speed up the computation, inherently filters noisy photon splats as a by-product. In contrast to traditional (ir)radiance caching, reducing the cache error ϵ below a certain minimum error will not give any quality improvements since bias is already introduced in the cached samples, which are computed via photon-ray density estimation. On the other hand, since the irradiance is already low-pass-filtered and therefore well-suited for interpolation, we can reduce the complexity of the ray-splatting algorithm by sparse sampling the image plane and interpolating in-between pixels.

We utilize a multi-pass radiance caching algorithm that exploits the kd-tree build on top of the eye samples (see Section 5.1). Only a sparse number of eye samples from the lower levels of the kd-tree is cached and computed via photon ray splatting. The radiance caching and extrapolation is carried out in image space via cache splatting similar to [7]. However, the disc-shaped splat footprint of each cache record is computed in world space and projected to image space as visualized in Fig. 6.5b.

The initial world-space radius r of each cache splat is computed from the harmonic mean distance $R(y_c)$ [34], which we measure only from incoming photon rays (x_i, ω_i) during ray splatting.

$$r(y_c) = \epsilon \cdot R(y_c), \tag{6.5}$$

$$R(y_c) = \max\left\{R_-(y_c), \min\left\{R_+(y_c), \frac{1}{\sum_{i=1}^{K} \frac{1}{z_i}}\right\}\right\}, \tag{6.6}$$

where ϵ is the user-defined cache error [34], R_-, R_+ are respectively the minimum, maximum allowed harmonic mean distance, corresponding to 2 times, 50 times the projected width of the pixel that maps to the cache location y_c [13, 14, 30]. To update the harmonic mean distance at all cache records in the splat footprint of a single photon ray, we need to compute the euclidean distance l of the cache records y_c to the ray origin x_i. Because this is computationally intensive in particular for larger ray splat footprints, we approximate this distance by the projected distance $z_i = (y_c - x_i) \bullet \omega_i$ along the ray direction ω_i (see Fig. 6.1). This approximation is sufficient for our purposes and is computed as a by-product in the density estimation.

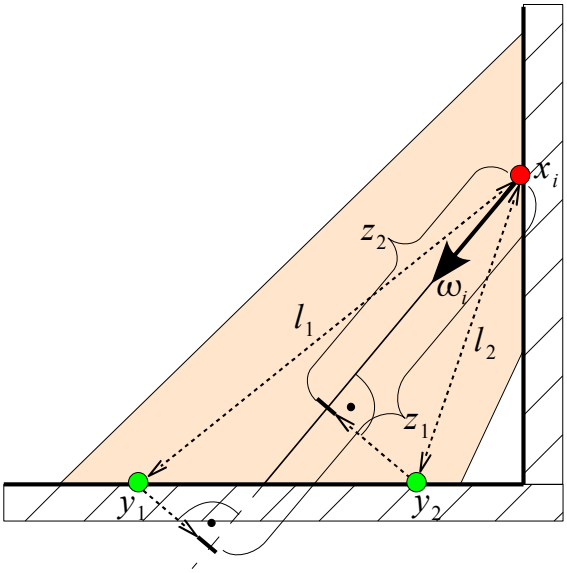


Figure 6.1: Approximating the harmonic mean distance (HMD) at two cache records (green dots) from the incoming photon rays (here shown for one ray in 2D only). Instead of computing the exact distances l_1 and l_2 to the neighboring cache records in the ray splatting footprint, the projected distances z_1 and z_2 are used to update the HMD, which are computed as a by-product of the density estimation.

Since one photon ray contributes to many neighboring cache records (see Fig. 6.1), short distances to nearby objects are less likely to be missed than for final gathering with hemisphere sampling where one ray contributes to one record only. This way, additional neighbor clamping [15] becomes obsolete. Nevertheless, due to the sparse number of photon rays in the vicinity of a cache record it is still possible to miss short distances to small objects. Therefore, we need to be more conservative in the cache error setting.

During cache splatting in image space a weight w_c is computed for each eye sample y that maps to the projected bounding rectangle of the cache footprint in image space. For filtering purposes we do not use Ward's original weighting function derived from the split-sphere model [34] since it is unbounded and has a singularity at the cache location (see Fig. 6.3), which creates spiky artifacts as shown on the top left in Fig. 6.2. Instead we have chosen a smooth filter function, which goes to zero at the maximum cache distance $\epsilon \cdot R_c$:

$$w_c(y) = \left(max \left\{ 1 - \frac{||y_c - y||^2}{\epsilon \cdot R(y_c)} \cdot \frac{1}{(N \bullet N_c)^4}, 0 \right\} \right)^2 . \tag{6.7}$$

The term $\frac{1}{(N \bullet N_c)^4}$ penalizes cache records with deviating surface normal N_c,

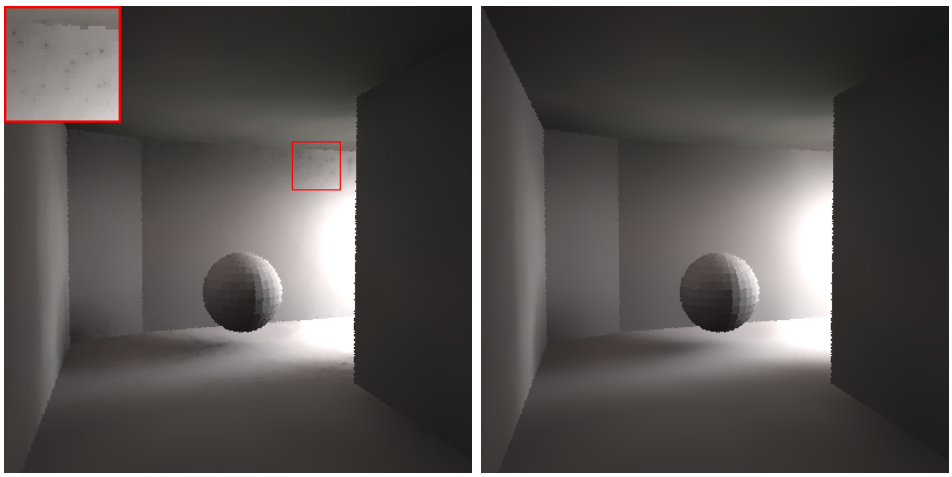


Figure 6.2: Irradiance cache results using Ward's weighting function (left) and our weighting function (right) used for cache interpolation. Note the artifacts in the left image due to unfiltered pixels corresponding to the cache locations. For both images approximately 2700 cache records are computed and at least 6 cache records contribute to a pixel.

where the exponent 4 was chosen for efficiency reasons. A comparison of Ward's weighting function with our weighting function is shown in Fig. 6.3 in 1D. If the weight $w_c(y)$ is greater than zero, the eye sample receives the weighted energy of the cache record, which is added to the *cache splat image* together with the weight.

After each cache splatting pass the *cache splat image*, which stores the sum of cache weights w and counts the number of contributing cache records per pixel, is processed in scanline order and every pixel is tested against a minimum required number of contributing caches N_c per pixel. For most scenes 4 to 10 contributing caches per pixel are sufficient. If a pixel has not yet accumulated enough weight (i.e. cache counter $< N_c$), its corresponding sub-tree of the kd-tree containing the pixel's eye sample is refined. One possible sampling pattern we have used for refinement is to pick every i-th eye sample from a sub-tree that needs to be refined and in the following pass every $i/2$ eye sample. The caching starts with one sample from each sub-tree that contains at most $\lfloor 2^{0.3 \cdot \log_2 N} \rfloor$ samples. This results in a sparse cache distribution with relatively low discrepancy in image space.

The previous steps are repeated for all newly generated cache records in each pass until no more cache records are created (i.e. all pixels have got sufficient weight and cache counts after scanlining the cache image). As an example of the iterative cache refinement in the SIBENIK scene see Fig. 6.4.

To improve the search for eye samples during photon ray splatting in repeating passes, we store references to the rays that traversed the initial

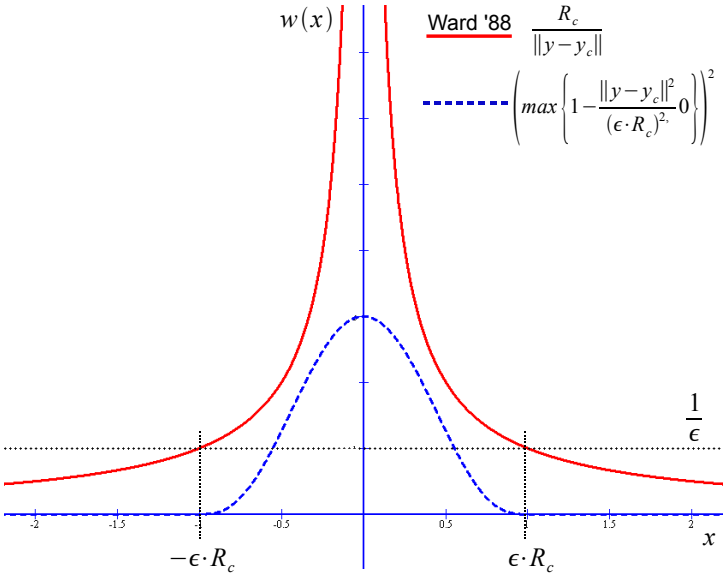


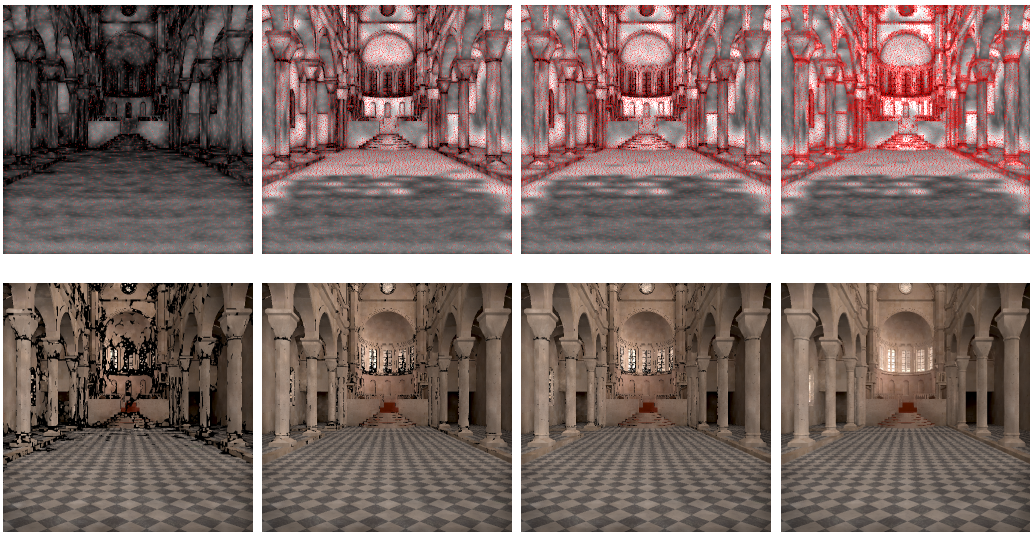
Figure 6.3: Comparing Ward's traditional cache-weighting function [34] (red solid curve) derived from the split sphere model with our continuous weighting function (blue dashed curve). Ward's weighting function has a singularity at the cache location and therefore does not allow filtering of the cache records, i.e. only one cache contributes at the cache location. Note that the weighting functions do not need to be normalized since the interpolated pixels are divided by the sum of weights afterwards.

nodes in the first pass. In following passes only the nodes' sub-trees need to be traversed and tested against the newly created cache records in the leafs and the upper search in the kd-tree is eliminated.

In the final pass, after all cache records have extrapolated their radiance SH coefficients to neighbor pixels, the SH coefficients λ_l^m in all pixel samples y are divided by the accumulated weight per pixel, which has been stored in the cache splat image:

$$\lambda_l^m(y) = \frac{\sum_{c\in\mathcal{C}_y} \lambda_l^m(y_c) * w_c(y)}{\sum_{c\in\mathcal{C}_y} w_c(y)}, \tag{6.8}$$

where the $\mathcal{C}(y) = \{c|w_c(y) > 0\}$ is the set of contributing cache records to pixel sample y. Note that we do not apply a rotation of the cached spherical harmonics (SH) coefficients [14] to align them with the local coordinate frame at y since the computational overhead of the SH rotation is too intensive compared with the computation of a new cache record in our framework. Therefore, we simply increase the cache density on curved surfaces (see Eq.(6.7)).



(I) (II) (III) (IV)

Figure 6.4: Iterative refinement of the radiance caching algorithm in the SIBENIK scene. In the initial pass (I) the cache records (red points) are uniformly distributed over the image plane. In the following passes (II – IV) the cache records are refined and splatted to neighboring pixels inside the projected splat footprint. The cache weights are accumulated in the cache image (top row). The cache weights depend on the estimated cache error computed from the harmonic mean distance and normal variation [34]. Hence the cache record density adapts to the cache error and more records are added to darker regions of the cache image shown in the top row. The bottom row shows the results after each pass (for demonstration purposes in form of pixel radiance which has been modulated with the BRDF SH coefficients).

(a) (b) (c)

Figure 6.5: Results of photon ray splatting (direct and indirect light) with radiance caching in spherical harmonics basis for the glossy APPARTMENT scene, (a) shows the noisy direct visualization resulting from a small splat size given as initial input to the radiance caching algorithm, (b) the radiance cache splats after first pass (for visualization purposes a smaller cache error was chosen), and (c) the final image after radiance caching with 3 passes. The time for computing photon ray splatting and cache splatting in 3 passes took 15 seconds for $500,000$ photons and 500×500 pixels.

6.3.1 Illumination Gradient Estimation

When using (ir)radiance caching the cache-record density adapts to the geometric gradient magnitude estimated from the split-sphere model [34], which is well established and commonly applied to most irradiance caching algorithms. However, this model is only valid for indirect illumination since it adapts to a geometric upper bound of the indirect gradient. Moreover, it often underestimates strong indirect illumination sources and overestimates near corners. As a remedy Ward et al. [35] proposed to compute the "real" irradiance gradient at each cache location in the local tangent frame as a by-product of final gathering (i.e. importance sampling the BRDF), which is then used for higher order cache interpolation. Furthermore, the irradiance gradient can also be used to control cache density [14]. If the magnitude of the irradiance gradient is larger than the estimated geometric gradient magnitude, the cache density is increased by reducing the cache's influence radius accordingly.

Our goals are similar. However, because we cannot estimate the gradient magnitude accurately enough, we do not want the noisy gradient to influence the cache interpolation since such interpolation [35] is very sensitive to the gradient. Instead we stick to our simple cache interpolation combined with additional filtering (see previous section). Nonetheless, the gradient helps to

control the cache density and therefore the filter width in order not to over-smooth illumination boundaries (e.g. shadow boundaries).[1] Since we do not perform view-dependent final gathering, we need a new approach for the gradient computation. Estimating the gradient $\nabla E(y)$ by central differences is cumbersome and too sensitive to noise. Fortunately, the gradient can be directly derived by differentiating the density estimation equation in Eq.(4.5):

$$\begin{aligned}\frac{\partial}{\partial u}E(y) \approx \sum_{i=1}^{K} \frac{\Delta\Phi_i(x_i,\omega_i)}{\Delta A^{\perp}_{\omega_i}} * [\frac{\partial}{\partial u}\{\mathcal{K}_h(y,x_i,\omega_i)\}\cdot\cos\theta_i + \\ \mathcal{K}_h(y,x_i,\omega_i)\cdot\frac{\partial}{\partial u}\{\cos\theta_i\}] \\ = \sum_{i=1}^{K} \frac{\Delta\Phi_i(x_i,\omega_i)}{\Delta A^{\perp}_{\omega_i}} * [\frac{\partial}{\partial u}\{\mathcal{K}_h(y,x_i,\omega_i)\}\cdot\cos\theta_i], \qquad (6.9)\end{aligned}$$

since $\frac{\partial}{\partial u}\{\cos\theta_i\} = 0$ in our metric, which assumes that the ray direction is constant within the density estimation footprint. The first derivative of the Epanechnikov kernel along direction u is a linear function in ray distance $||\vec{r}||$:

$$\begin{aligned}\frac{\partial}{\partial u}\mathcal{K}_h(y,x_i,\omega_i) &= \frac{\partial}{\partial u}\left\{2\cdot\left(1-\frac{||\vec{r}(y,x_i,\omega_i)||^2}{h_i^2}\right)\right\} \\ &= \begin{cases} -\frac{2}{h_i^2}(2r_u) & ||\vec{r}||^2 < h_i^2 \\ 0 & otherwise, \end{cases} \qquad (6.10)\end{aligned}$$

where $\vec{r}(y,x_i,\omega_i) = \{\vec{\omega}_i \times (y-x_i)\}\times\vec{\omega}_i$ is the distance vector to the photon ray and $r_u = \vec{r}\bullet\vec{U}$ is the projection into the local coordinate-frame axis $\vec{U}$ at y. $\vec{\omega}_i$ is the ray direction in euclidean coordinates. Note that the term $\frac{\Delta\Phi_i(x_i,\omega_i)}{\Delta A^{\perp}_{\omega_i}}$ as well as the bandwidth h_i are assumed to be independent of the density estimation point y and therefore do not change when displacing y along $\vec{U}$[2]. For a better understanding see the schematic draft in Fig. 6.6. And similarly for the V dimension:

$$\frac{\partial}{\partial v}E(y) \approx \sum_{i=1}^{K} \frac{\Delta\Phi_i(x_i,\omega_i)}{\Delta A^{\perp}_{\omega_i}} * [\frac{\partial}{\partial v}\{\mathcal{K}_h(y,x_i,\omega_i)\}\cdot\cos\theta_i], \qquad (6.11)$$

[1] Increasing the cache density according to the gradient magnitude and at the same time also using the gradients for higher order cache interpolation seems slightly redundant since the gradient based interpolation already compensates for linear changes in lighting. A better choice would be to compute the second derivative for steering the cache density and to use the gradient for interpolation.

[2] Actually h_i is only independent of y if the bandwidth is constant along the ray, i.e. cylindrical splatting footprint.

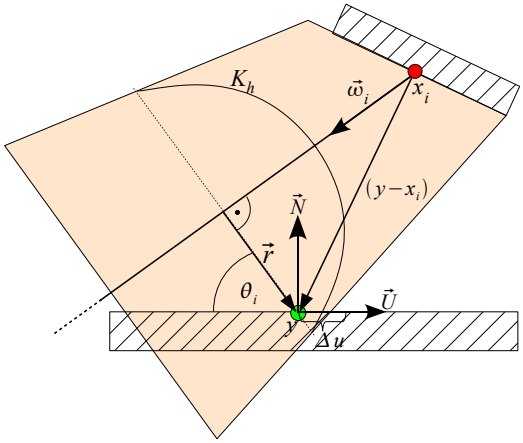


Figure 6.6: Quantities needed for computing the gradient along $\vec{U}$.

with $\frac{\partial}{\partial v}\mathcal{K}_h(y, x_i, \omega_i) = -\frac{2}{h_i^2}(2r_v)$, where $r_v = \vec{r} \bullet \vec{V}$.

Having computed the irradiance gradient $\nabla E(y) = \left[\frac{\partial}{\partial u}E(y), \frac{\partial}{\partial v}E(y)\right]$, we are able to steer the cache density by controlling the cache splat radius. Wherever the geometric gradient $||E(y)/R(y)||$ estimated from the harmonic mean distance $R(y)$ (see previous section) is less than $||\nabla E(y)||$, we decrease $R(y)$ by setting it to:

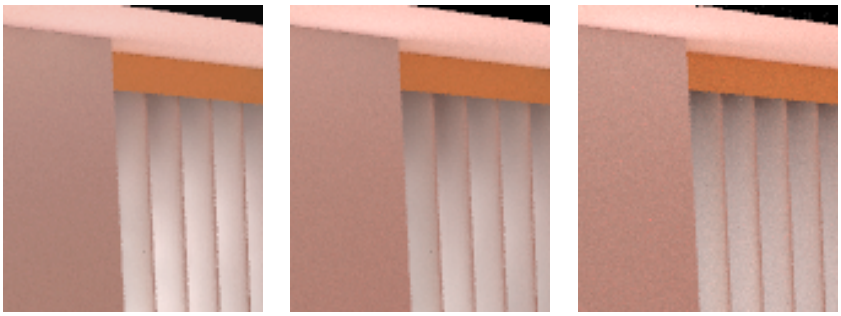
$$R(y) := \frac{E(y)}{||\nabla E(y)||}, \tag{6.12}$$

which consequently results in an increased cache density around y (see Fig. 6.8). The images in Fig. 6.8 show the results for the estimated irradiance with and without using the irradiance gradient for controlling the cache density. The overhead for computing the irradiance gradient during the ray splatting is negligible ($2-4\%$ increased computation time) but significantly improves the results in particular for direct lighting. A minor drawback of the proposed gradient computation in Eq.(6.9) is that the computed gradient might vanish for high-frequency illumination patterns where the gradients have similar direction but opposite orientation. A remedy is to estimate the gradient in a smaller neighborhood and then use 2×2 structure tensors [22] to filter the noisy gradients without cancellation effects [37].

6.4 High Quality Rendering with Final Gathering

In photon density estimation the visibility within the density estimation footprint is neglected and high frequency indirect lighting due to occlusion cannot

be reproduced and may lead to energy leaking for complex scenes. To address this problem, the global photon map is only queried for secondary eye rays, refered to as final gather rays (FGRs), generated using Monte Carlo sampling of the BRDF (final gathering) [11]. Final gathering balances the local error in the photon map estimate across all pixels and produces high quality indirect illumination (except for caustics, which are computed from a direct visualization of the caustics photon map). Nevertheless, final gathering with photon mapping has its shortcoming for concave surfaces where many FGRs hit the local neighborhood resulting in overestimated illumination, see Fig. 6.7(a). In such cases secondary final gathering is initiated drastically increasing the computation cost of the corresponding pixel. This is especially problematic for (ir)radiance caching [34] where the cache samples are concentrated near concave features such as corners. Combining our method with final gathering, we can mostly avoid secondary final gathering and also speed up the nearest neighbor search compared to photon mapping. This requires only a small change in the algorithm described in Section 3. Instead of primary ray hit points, we need to store all FGR hit points. To handle the increased memory demands, the image plane is rendered in tiles utilizing multiple splatting passes with the same photon ray distribution as proposed in [9]. Further, we do not use the radiance map (Section 6.1), which would be too memory consuming, but directly splat photon energy to the corresponding pixels weighted by the FGR contribution and the BRDF at the FGR hit point.



(a) (b) (c)

Figure 6.7: Comparing indirect lighting results in the conference scene for (a) photon mapping with 600 final gather rays per pixel, (b) ray splatting with 600 final gather rays per pixel, and (c) path tracing with 2000 paths per pixel. The full images are shown in Fig. 7.3(d).

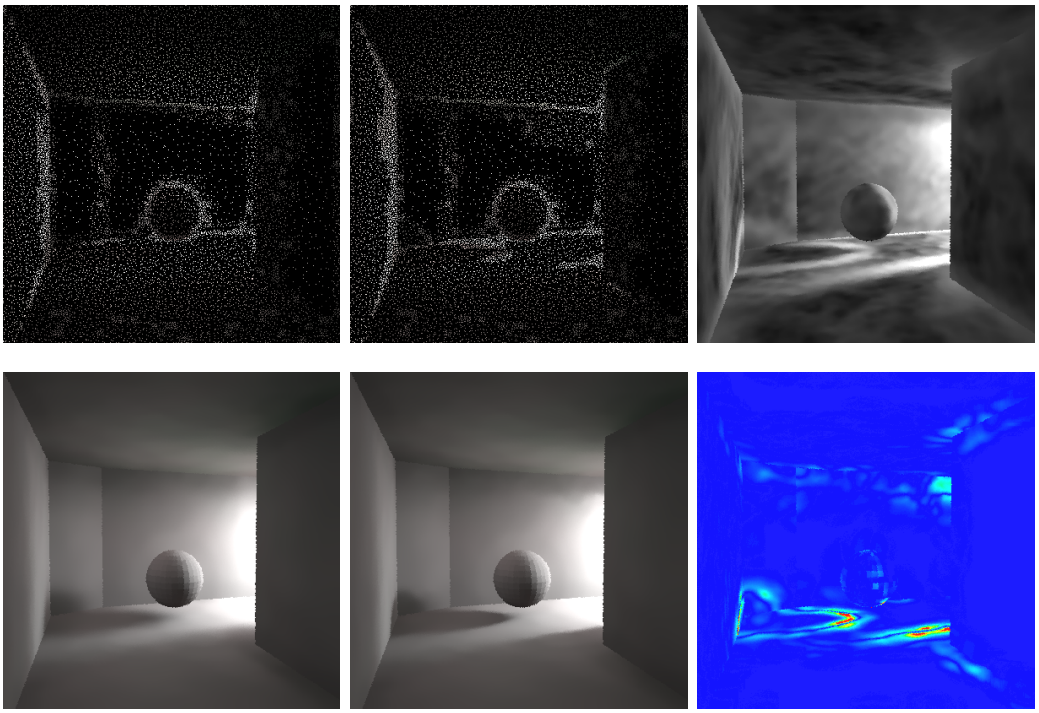


Figure 6.8: Irradiance caching results for sampling with geometric gradients (1. column) and sampling with our irradiance gradients computed during ray splatting (2. column). First row (from left to right): irradiance cache locations adapting to the geometric gradient [34], irradiance cache locations adapting to our irradiance gradient, our gradient magnitude (3 f-stops brighter). Cache sampling according to the geometric gradient results in $13,116$ caches records, while sampling according to our irradiance gradient yields $14,272$ cache records and preserves the shadow boundaries by reducing the filter bandwidth near the strong gradients according to Eq.(6.12). The rendering times are about 6 seconds for splatting $750,000$ photon rays and $14,000$ cache records. The bottom right image shows the color-coded difference between the bottom left and bottom middle image.

7 Results

We have evaluated our method for the scenes shown in Fig. 7.2 and Fig. 7.3. The scene APPARTMENT is copyrighted by Laurence Boissieux © INRIA 2005. All results were computed on a single PC (AMD Opteron 2.4 GHz) with a Linux operating system installed. Our algorithm has been implemented in C++ with STL on top of an existing rendering system. For compilation we used `g++ 3.4` with `-O2` optimization. No low-level code optimization has been applied.

We compared our method with standard K-NN density estimation using a direct visualization of the photon map [11]. The rendering times are given in Table 7.1. The times T_{init} and T_{ray} are the same for all methods. The times for photon tracing T_{light} and photon kd-tree construction T_{build} are slightly faster in photon mapping because for ray splatting T_{light} includes bandwidth selection and T_{build} comprises kd-tree construction over eye samples as well as kd-tree construction over photon rays. As there is no postprocessing in a direct visualization of the photon map, the times T_{cache} and T_{brdf} are zero. For a fair comparison we have used the same data structures and algorithms for sampling and searching for photon mapping as for ray splatting. To achieve the same level of noise in the result we had to set the number of k-nearest neighbors (K-NN) from at least 400 up to $1,200$ photons. In certain cases of glossy light-transport reconstruction from the photon maps [10], our ray splatting method outperformed the photon map since searching for a large number of K-NN and evaluating the BRDF for all K-NN photons becomes the bottleneck in photon mapping. This holds also for ray splatting if no radiance map is used (direct ray splatting). Moreover, the search for the exact K-NN photons can be quite time consuming for a large number of K, which is in particular problematic if the photon map queries are incoherent (e.g. for final gathering).

The ray splatting approach also scales well with image resolution and with number of photons. The time and memory dependencies on image resolution and photon number are shown in Fig. 7.1. The graphs show the measured

rendering times (columns a and b) and memory usage (column c) of four different methods for the CORNELL BOX (column a) and the APPARTMENT scene (column b): direct ray splatting (RS direct) to the image, direct visualization of the photon map with k-NN density estimation (PM k-NN), photon ray splatting to directional histogram (RS histogram), the ray splatting in spherical harmonics basis with radiance caching (RS+SH caching), and its cache computation time only (RS only). For fair comparison of our method with photon mapping, we also measured the time for direct ray splatting (red curve) to the image without the additional optimizations, i.e. no radiance map, no radiance caching, no filtering.

From the graphs one can observe that the behavior of ray splatting is similar to density estimation from the photon map in case of diffuse scenes. For glossy scenes the BRDF evaluation for all photon rays to eye sample candidates dominates the rendering time and the direct photon ray splatting becomes less efficient because of its larger density estimation footprint compared to photon mapping. However, the histogram splatting (blue curve) and the spherical harmonics splatting with radiance caching (black solid curve) significantly speedup the algorithm in particular for the non-diffuse APPARTMENT scene (column b) on the expense of increased memory utilization.

Due to the directional and spatial coherence in the photon ray splatting and the automatic bandwidth selection, the rendering time is sub-linear in the number of photons even though the photon rays are splatted sequentially. Ray splatting is faster than photon mapping if the number of photons is much smaller than the number of eye samples (pixel samples) as is usually the case for final gathering (see Table 7.1). It becomes less efficient if the number of photons increases because in ray splatting we search for each photon ray in a tree over eye samples whereas for photon mapping we search for all eye samples in a tree over photons (see [9] for more details).

Combining ray splatting with radiance caching (black curve), the pure splatting time dependency (black stippled curve) on image resolution is close to constant since the cache records are distributed in world space and are thus independent of the image. Moreover, the kd-tree traversal in the upper levels of the tree is eliminated for all subsequent cache passes because we keep references to the photon rays that traversed the initial kd-tree nodes in the first pass, which on the other hand boosts the memory requirements (column c).

The overall rendering times of our method range from about 30 seconds to 1 minute for a single image with a resolution of 500×500 pixels and $500,000$ photons. In this setting the memory requirements for splatting and radiance map are about 100 to 160 MBytes independent of the scene complexity. Note that the memory requirements can be reduced significantly if photon rays are

not explicitly stored during photon tracing but are progressively splatted to screen pixels.

The ray density estimation and search in a conical frustum (Section 5) is approximately 1.5 to 2 times slower than for photon density estimation in a spherical footprint with the same precomputed radius (i. e. without K-NN search). This holds also for a larger number of eye samples, for example when final gathering is used storing several hundred eye samples per pixel.

The bandwidth smoothing parameter C determines the noise level in the density estimation, while the sensitivity S controls the variance ($S = 0$ results in a constant bandwidth). Since we normalize the bandwidth term using an initial pilot estimate (Section 4.2), C is relatively independent of scene size and complexity. Therefore, C varies around 1.0. Parameter S depends on the lighting conditions. For direct lighting and caustics we set S around 0.5, while for indirect lighting values between 0.2 and 0.3 yield satisfying results. In case of final gathering (Section 6.4), C and S should be set to smaller values than the ones used for a direct visualization in order to keep performance high and reduce the overall bias.

In order to compute glossy light transport, we have implemented and tested three different approaches: the "naive" direct ray splatting with BRDF evaluation for every photon ray, the histogram splatting, and the splatting in the SH basis. For the histogram method we used 20 strata and for the SH method 16 coefficients per pixel, which yields similar results. The splatting to the histogram is more efficient. However, the final BRDF evaluation is more expensive since we apply BRDF sampling for all non-diffuse eye samples. For the SH method the final step reduces to a simple dot product of SH coefficients. The SH splatting method itself is computationally expensive but works well in combination with radiance caching.

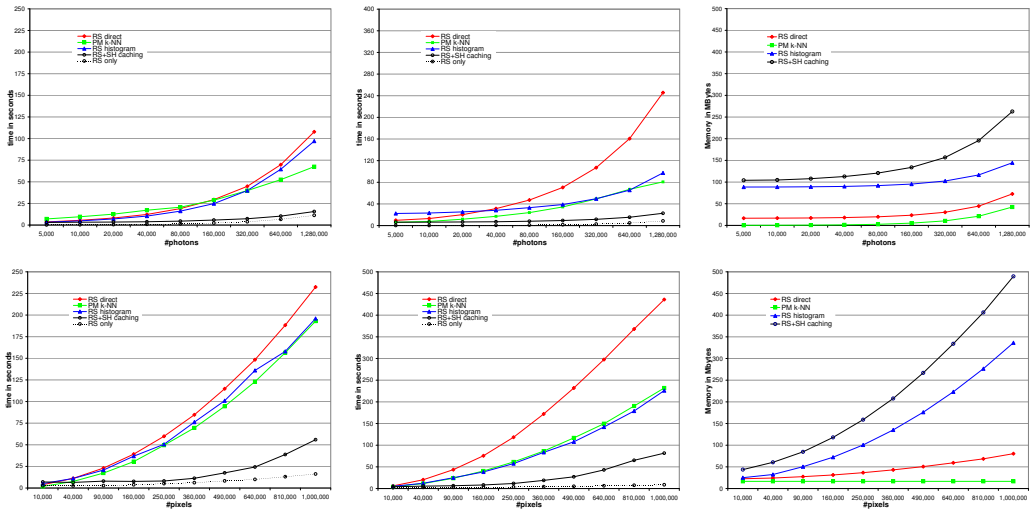
The additional radiance caching scheme further reduces the rendering time by one order of magnitude but requires to find a good combination of smoothness parameter C for splatting (Section 4.2) and cache error ϵ (Section 6.3). These two parameters are correlated. Noise is filtered when either choosing a large C and small ϵ or a small C and larger ϵ. However, the latter is more efficient and can, surprisingly, even enhance the visual quality of the results (see Fig. 7.2d). As a rule-of-thumb, when applying radiance caching, setting the value of C to its half leads to satisfying and fast results. For filtering purposes, the minimum number of extrapolated cache records contributing to a pixel was set to at least 4 to 7.

The optional radiance filtering in image space using the discontinuity segments takes 10 to 20 seconds for 20 radiance images, while for filtering the radiance SH coefficients the times are between 6 and 10 seconds. The segmentation of the discontinuity buffer is computed in approx. 50 ms for

an image of 500×500 pixels.

The proposed extensions: directional histogram, the SH radiance caching, and the radiance image filtering, can also be applied to standard photon mapping.

In Fig. 7.2 we compare our method quality-wise with photon mapping using k-nearest neighbor (K-NN) density estimation. The left image shows the reference solution obtained by final gathering with 1200 final gather rays per pixel, where the radiance along final gather rays is computed from the photon map. The second column (b) shows the solution from a direct photon map visualization with approximately 500 nearest neighbor photons per pixel. The third column (c) was rendered with our proposed photon ray splatting method in the spherical harmonics basis with the smoothness parameter C (see Section 4.2) chosen to have on average a similar density-estimation kernel width (splat radius) as in the photon map solution. The fourth column (d) shows the filtered radiance cache solution that is based on a noisy photon ray splatting input (see for example Fig. 6.5). For all solutions we have used the same photon sampling algorithm with $500,000$ stored photon samples in total whereby only those photon rays were stored that intersected the enlarged viewing frustum.

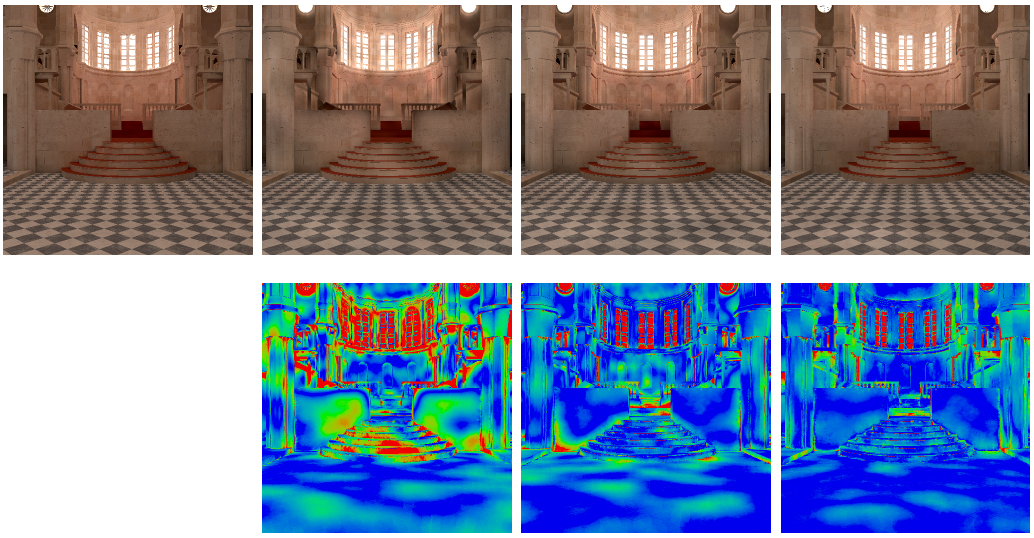


(a) (b) (c)

Figure 7.1: Scalability of photon ray splatting (red curve) compared to photon map K-NN density estimation (green curve) for the CORNELL BOX scene (column a) and the APPARTMENT scene © INRIA 2005 (column b). First row shows the rendering time in dependence on the number of stored photons for $x = 5,000$ to $x = 1,280,000$ photons with constant image resolution (500×500). The second row shows the rendering time depending on the image resolution ranging from $x = 100 \times 100$ to $x = 1,000 \times 1,000$ pixels with constant number of stored photons ($500,000$). The used memory for the APPARTMENT scene excluding geometry and ray tracing data structures is shown in column (c). The red curve represents the direct photon ray splatting (RS direct) to the image, i.e. evaluating BRDF for all eye samples in the ray's footprint, the blue curve (RS histogram) represents the directional histogram method, and the black solid curve represents the ray splatting in spherical harmonics basis with adaptive radiance caching. The stippled black curve shows only the fraction of time spent in computing the radiance cache records. Note that the scaling of the x-Axis is non-linear.

Scene	Method	T_{init}	T_{ray}	T_{light}	T_{build}	T_{solve}	T_{cache}	T_{brdf}	$\sum T$
CORNELL BOX	Photon Map (800 K-NN)	0.01	0.33	1.80	0.28	49.58	–	–	52.00
(100 % diffuse)	Histogram (C = 1.0, S = 0.4)	0.01	0.33	1.91	0.33	51.10	–	0.15	53.80
$N_{Prim} = 18$	SH Basis ($C = 1.0, S = 0.4$)	0.06	0.33	1.90	0.33	77.50	–	0.04	79.20
	SH Cache ($C = 0.6, S = 0.2$)	0.11	0.34	1.90	0.34	4.90	2.6	0.06	10.25
Scene settings	$500 \times 500 \times 1$; $M = 500,000$; $(0.3, 0.7, 0.0)$; $R = 0.2$								
CORNELL BOX	Photon Map (400 K-NN)	0.13	1.74	1.50	0.20	64.80	–	–	68.37
(WAVE)	Histogram ($C = 1.0, S = 0.4$)	0.14	1.75	1.60	0.25	55.30	–	3.20	62.24
(98 % diffuse)	SH Basis ($C = 1.0, S = 0.4$)	0.16	1.75	1.62	0.26	87.50	–	0.06	91.35
$N_{Prim} = 19,635$	SH Cache ($C = 0.6, S = 0.3$)	0.20	1.76	1.62	0.26	8.10	3.8	0.10	15.86
Scene settings	$500 \times 500 \times 4$; $M = 200,000$; $(0.3, 0.6, 0.1)$; $R = 0.2$								
CORNER ROOM	Photon Map (600 K-NN)	0.01	0.71	3.74	0.66	59.50	–	–	64.60
(0 % diffuse)	Histogram ($C = 1.2, S = 0.5$)	0.01	0.70	3.97	0.71	35.70	–	21.06	62.00
$N_{Prim} = 59$	SH Basis ($C = 1.2, S = 0.5$)	0.07	0.69	3.96	0.70	125.00	–	0.09	130.50
	SH Cache ($C = 0.7, S = 0.4$)	0.10	0.70	3.94	0.70	6.20	3.0	0.09	14.70
Scene settings	$500 \times 500 \times 1$; $M = 1000,000$; $(0.0, 0.95, 0.05)$; $R = 0.2$								
SIBENIK	Photon map (500 K-NN)	0.32	0.53	3.84	0.31	26.20	–	–	31.20
(99 % diffuse)	Histogram ($C = 0.9, S = 0.3$)	0.32	0.54	5.30	0.35	25.70	–	0.57	32.80
$N_{Prim} = 78,362$	SH Basis ($C = 0.9, S = 0.3$)	0.40	0.55	5.29	0.34	31.40	–	0.05	38.00
	SH Cache ($C = 0.5, S = 0.2$)	0.43	0.55	5.30	0.35	7.40	6.6	0.05	20.70
Scene settings	$500 \times 500 \times 1$; $M = 500,000$; $(0.0, 1.0, 0.0)$; $R = 0.2$								
APPARTMENT	Photon map (600 K-NN)	0.37	0.65	3.80	0.29	61.60	–	–	66.70
(47 % diffuse)	Histogram ($C = 0.9, S = 0.4$)	0.37	0.63	4.18	0.36	45.10	–	14.10	64.70
$N_{Prim} = 73,668$	SH Basis ($C = 0.9, S = 0.4$)	0.41	0.64	4.19	0.35	104.00	–	0.11	109.70
	SH Cache ($C = 0.5, S = 0.3$)	0.44	0.63	4.19	0.34	9.50	5.9	0.11	21.10
Scene settings	$500 \times 500 \times 1$; $M = 500,000$; $(0.0, 0.9, 0.1)$; $R = 0.2$								
CONFERENCE	Photon map (70 K-NN)	0.77	2.45	2.47	0.36	35.8	–	–	41.85
(86 % diffuse)	Ray Splat ($C = 0.6, S = 0.2$)	0.78	2.47	2.59	0.86	25.6	–	–	32.00
$N_{Prim} = 265,880$	PM-FG (600 FGRs)	0.78	991.00	2.49	0.16	4114.00	–	–	5108.00
	RS-FG (600 FGRs, 5×5 tiles)	0.83	997.00	2.58	111.00	2714.00	–	–	3825.00
Scene settings	$700 \times 700 \times 4$; $M = 160,000$; $(0.3, 0.6, 0.1)$; $R = 0.4$								

Table 7.1: Computation times for all rendering phases of our algorithm for 6 scenes using either a direct visualization of the photon map, the histogram, the spherical harmonics (SH) approach, the direct ray splatting (Ray Splat), photon mapping with final gathering (PM-FG), or direct ray splatting with final gathering (RS-FG) for computing the radiance at eye samples. The computed images are shown in Fig. 7.3. N_{Prim} is the number of primitives in the scenes. T_{init} is the time for preprocessing (e.g. kd-tree construction for ray tracing, discontinuity segmentation), T_{ray} is the time spend for casting primary rays and storing eye path samples, T_{light} is the time for tracing M photons, T_{build} is the kd-tree construction time for eye samples and time for presorting photon rays (5D-tree construction), T_{solve} is the time for photon ray splatting (including search), T_{cache} is the time for radiance cache splatting, T_{brdf} is the time for computing the outgoing pixel radiance at each eye sample, i.e. BRDF evaluation and eye sample weighting. And $\sum T$ is the total time spend to compute a single frame. In case of the photon map approach T_{build} corresponds to the construction of the kd-tree over photons and T_{solve} is the K-NN density estimation time including BRDF evaluation on non-lambertian surfaces. The scene settings are: image resolution $\times$ number of super-samples per pixel; total number of stored photons (M); the fraction of direct, indirect diffuse, caustics photons; the bandwidth clamping parameter R.



(a) 3950 s (b) 29 s (c) 35 s (d) 20 s

Figure 7.2: Comparing our splatting method with photon maps K-NN density estimation relative to a reference solution for indirect light in the diffuse SIBENIK scene. The first row shows the results of: (a) photon maps with 1200 final gather rays per pixel and 50 nearest neighbor photons per ray, (b) the direct visualization of photon maps with 500 nearest neighbors (c) our photon ray splatting in spherical harmonics basis (d) photon ray splatting with additional radiance caching and filtering. All methods use $500,000$ photon samples. The second row shows the relative error color-coded from blue ($< 5\%$ error) to green (15% error) to red ($\geq 30\%$ error) with respect to the reference image. Note the reduced bias near the boundaries and on curved surfaces for the ray splatting approach. Note also that the filtering due to radiance caching (image d) further reduces low-frequency noise and leads to better visual quality.

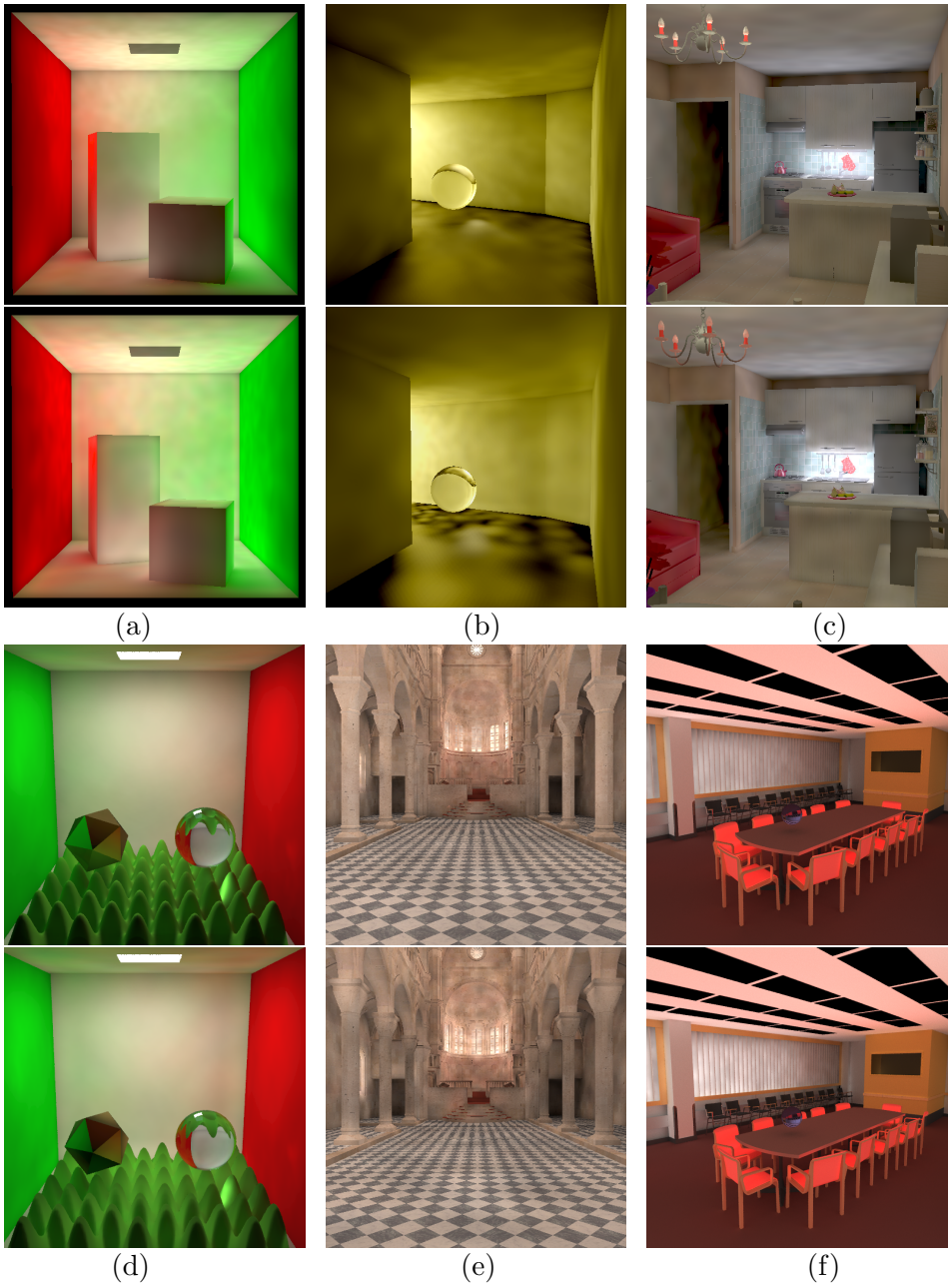


Figure 7.3: Comparing our method (top images) with k-NN photon density estimation in 6 scenes with different illumination conditions (the rendering times are given in Table 7.1). The number of k-NN photons was chosen to have on average a gather radius similar to the splat radius used for ray splatting. (a) classic Cornell Box (b) the glossy Cornerroom with difficult lighting conditions that generate indirect caustics on a slightly glossy floor, (c) the Appartment scene © INRIA 2005, which exhibits indirect diffuse and glossy light transport, (d) bumpy Cornell Box with full global illumination, (e) diffuse Sibenik scene, and (f) the indirect diffuse Conference scene rendered with final gathering using 600 rays per pixel.

8 Discussion and Future Work

Our proposed algorithm leaves space for further optimization and extensions of various kind. First, our radiance map implementation is not adaptive to the "glossiness" of a surface. Using a different basis with adaptive number of coefficients per pixel can increase the quality of glossy light reflections while decreasing computation time.

Second, we would like to extend our method into the temporal domain by reusing radiance information from previous frame(s). Inspired by [26] we believe that temporal coherence in the radiance cache distribution can give a speedup of one order of magnitude compared to single frame rendering and reduces flickering between consecutive frames. Together with adaptive (bilateral) filtering in the temporal domain [36], we could also reduce the low-frequency noise from photon sampling and increase the visual quality of the images.

Third, our method can also be used in a preprocessing step for computing an approximation to the real illumination in a finite element manner. The illumination on diffuse surfaces can be precomputed with photon ray splatting at the photon-ray hit points for second pass Monte-Carlo final gathering [2].

Our algorithm can also run in a distributed setup: the entire kd-tree and all eye samples can be instantiated on several clients whereas the photons are distributed among them. Because light is additive the resulting radiance images can be accumulated in a final pass.

Finally, the major drawback of our algorithm as in the photon mapping is the neglected visibility in the density estimation footprint. However, since our nearest neighbor search is along photon paths, we can gain more information about occlusion than only considering the local neighborhood of the photon-ray hit point. This could either be used as an instrument to control the bandwidth or even better to mask the kernel. Here we propose a simple yet efficient way to approximate the visibility during the kd-tree traversal.

8.1 Visibility Approximation

Photon ray splatting solves two major drawbacks of traditional photon density estimation, the boundary bias and the topological bias problem [8, 23]. However, as described so far in the previous sections the ray splatting does not improve the proximity bias because photon ray splatting does not account for visibility changes within the splatting footprint. Visibility changes yield occlusions (shadows) which impose the highest frequencies in the illumination especially for direct light. Therefore, ray splatting is not capable to reconstruct high-frequency signals in the lighting function if the visibility is neglected. Without explicit visibility testing, photon ray splatting and all other photon density estimation approaches only work satisfactory for smooth indirect illumination. Unfortunately, efficient visibility computation is one of the most difficult problems in computer graphics since making simplifying assumptions about the visibility is generally not possible. The brute-force solution would test each eye sample in the splatting footprint for occlusion with the photon ray's origin via ray-tracing. Such approach has been proposed in [1], where they stochastically decide whether to shoot a shadow ray or not. On the other hand, testing the eye sample's visibility explicitly ruins the advantage of photon density estimation, which computes the convolved visibility implicitly via density estimation.

We propose a simple visibility approximation exploiting the additional spatial information of the eye samples gathered during the ray traversal of the kd-tree. The gathered eye samples are "rasterized" into a 2D occlusion buffer, which functions as a stencil or mask for the density estimation kernel (see Fig. 8.1 b). Such approach only accounts for direct occlusions, i.e. occluders must be visible to the camera. However, indirect occlusions, although blurred, are implicitly handled by the density estimation. In order to prevent holes when masking the kernel, a "masking" density could be applied similarly as in point sample rendering techniques [39]. The masking density should depend on the eye sample density and perhaps other factors such as the distance and orientation relative to the camera for example. Therefore this approach still requires further investigation and remains as future work.

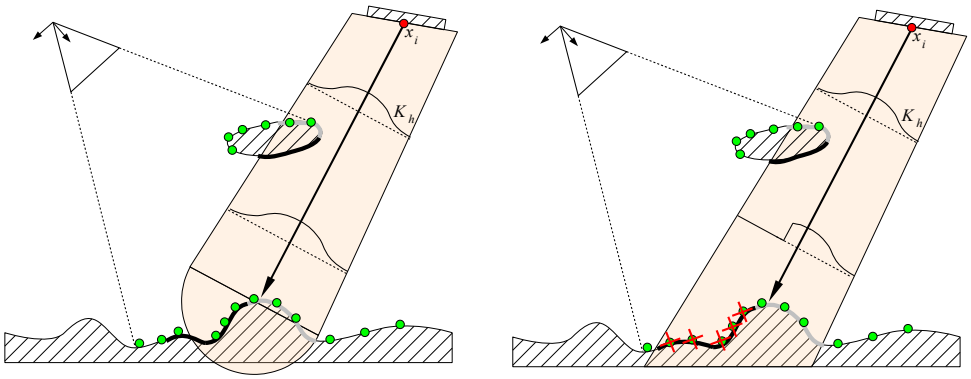


Figure 8.1: (a) Standard photon ray splatting neglects occlusions in the splatting footprint (thick black curves) whereas photon ray splatting with explicit visibility masking of the kernel function K_h using the gathered eye samples (green dots) inside the splatting footprint correctly discards occluded eye samples (b) and reproduces high-frequency shadows (see Fig. 8.2 second row).

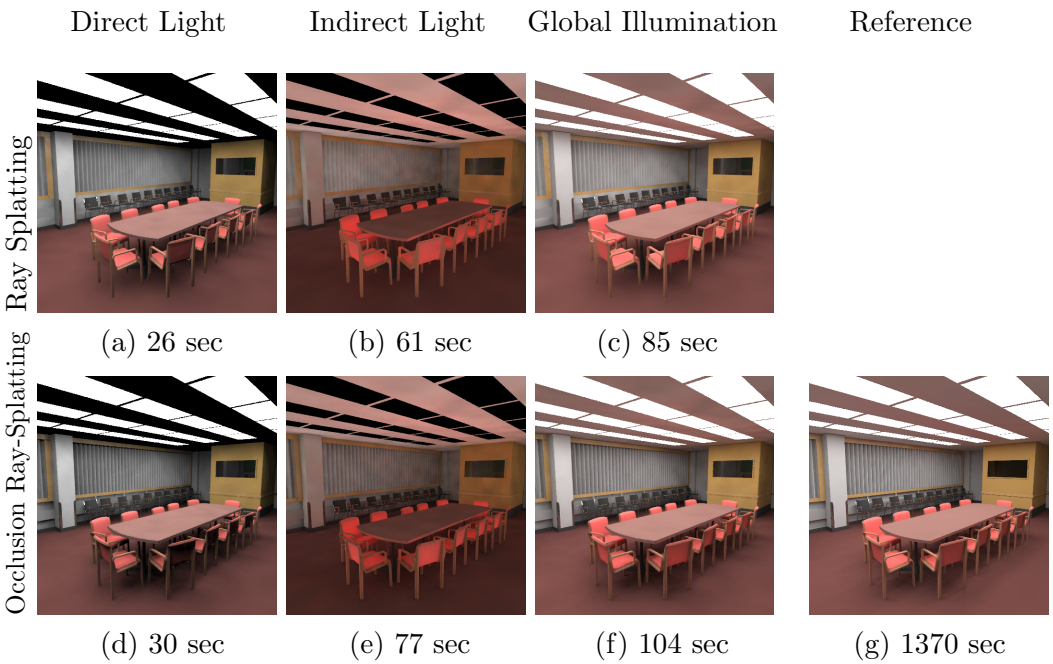


Figure 8.2: Including visibility information into the ray splatting significantly improves quality in particular for direct illumination and increases robustness against the sensitive bandwidth selection. First row shows the rendered images without explicit visibility computation. The results in the second row include visibility information. The first column shows only the direct light computed from $50,000$ photons, second column the indirect light (1f-stop brighter) computed from $200,000$ photons, third column the global illumination result ($250,000$ photons), and the last column (image g) is the reference computed with the *Lightcuts* algorithm [33] using an average of 339 shadow rays per pixel ($250,000$ virtual point light sources).

9 Conclusions

We proposed an algorithm that improves photon density estimation by exploiting further information acquired during photon generation and sampling phase like for example the photon ray direction and length, and its path probability density. Our method solves some of the problems inherent to photon density estimation and brings the quality closer to the expensive final gathering approaches.

First, we eliminate boundary bias due to the volumetric search along photon paths. This is especially noticeable on small unconnected surfaces where all hit-point density-estimation techniques fail. Since we do this via splatting instead of gathering, we avoid the use of complex and memory demanding data structures as in [8].

Second, our method does not suffer from discontinuities in surface orientation. Since we estimate the density in photon ray space, we decouple the density estimation from the surface area and obtain the convolved radiance in ray space. Therefore, we can compute the illumination from any direction on surfaces of complex topology where the actual surface area is difficult to estimate.

Third, we developed a simple and efficient bandwidth selection scheme for the photon splatting based on the photon-path probability density, which could also be used to speedup standard photon mapping [11] since the costly k-NN search can be avoided.

Fourth, we have shown how to adapt the classical irradiance caching algorithm [34] for use in a fast direct visualization of the photon density. We have replaced the expensive final gathering for estimating a cache record's irradiance and harmonic mean distance to the surrounding surfaces, which is needed to determine the cache spacing, by our photon density estimation. In addition, we also proposed a novel cache weighting-function, which enables to filter noisy cache records. We derived a simple and efficient irradiance gradient computed during ray splatting, which can further enhance the visual quality of the image computed with our radiance caching algorithm.

Although our algorithm often yields satisfying results, it also has some limitations. Like in all density estimation methods there are occasionally problems with light leakage due to the neglected visibility in the splat footprint. Only low-frequency lighting can be reconstructed with our method, but final gathering can be performed similar to photon mapping [9]. We are currently working on incorporating partial visibility information gathered during the photon ray traversal to the ray splatting, which further improves image quality. At present the method also requires tuning several parameters for the bandwidth selection: the smoothness coefficient C, bandwidth sensitivity S, and the bandwidth clamping parameter R, which is one parameter more than for photon mapping [11]. Also, if radiance image filtering is enabled, the user must provide two additional parameters, normal deviation and a distance threshold, for the segmentation of the discontinuity buffer.

We conclude that our algorithm has potential in fast rendering of low-frequency illumination, which could be either used for fast previewing or as a better input for high-quality Monte Carlo final gathering [2].

Bibliography

[1] P. Bekaert, S. Philipp, R. Cools, V. Havran, and H.-P. Seidel. A custom designed density estimation method for light transport. Research Report MPI-I-2003-4-004, Max-Planck-Institut für Informatik, September 2003.

[2] P. H. Christensen. Faster photon map global illumination. In *Journal of Graphics Tools*, volume 4, pages 1–10, 1999.

[3] C. Dachsbacher and M. Stamminger. Splatting indirect illumination. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 93 – 100, 2006.

[4] T. Dahmen. Multi resolution ray tracing of point data. Master's thesis, available at `http://graphics.cs.uni-sb.de/~morfiel/diplomarbeit.pdf`, Universität des Saarlandes, Computer Graphics Group, 2004.

[5] K. Dmitriev, S. Brabec, K. Myszkowski, and H.-P. Seidel. Interactive global illumination using selective photon tracing. In *Rendering Techniques 2002*, pages 25–36, 2002.

[6] Ph. Dutré, E.P. Lafortune, and Y.D. Willems. Monte carlo light tracing with direct computation of pixel intensities. In *Proceedings of Compugraphics '93*, pages 128–137, 1993.

[7] P. Gautron, J. Křivánek, K. Bouatoch, and S. Pattanaik. Radiance cache splatting: a gpu-friendly global illumination algorithm. In *Rendering Techniques 2005*, pages 55–64, 2005.

[8] V. Havran, J. Bittner, R. Herzog, and H.-P. Seidel. Ray maps for global illumination. In *Rendering Techniques 2005*, pages 43–54, Konstanz, Germany, June 2005.

[9] V. Havran, R. Herzog, and Hans-Peter Seidel. Fast final gathering via reverse photon mapping. In *Computer Graphics Forum*, volume 24, pages 323–333, 2005.

[10] H. W. Jensen. Rendering caustics on non-lambertian surfaces. In *Computer Graphics Forum*, volume 16, pages 57–64, March 1997.

[11] H. W. Jensen. Realistic image synthesis using photon mapping. AK, Peters, 2001.

[12] A. Keller. Instant radiosity. In *SIGGRAPH: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 49–56, 1997.

[13] J. Křivánek, P. Gautron, S. Pattanaik, K. Bouatouch. Radiance caching for efficient global illumination computation. In *IEEE Transactions on Visualization and Computer Graphics*, volume 11(5), pages 550–561, 2005.

[14] J. Křivánek. Radiance caching for global illumination computation on glossy surfaces. Ph.d. thesis, Université de Rennes 1 and Czech Technical University in Prague, December 2005.

[15] J. Křivánek, K. Bouatouch, S. N. Pattanaik, and J. Žára. Making radiance and irradiance caching practical: adaptive caching and neighbor clamping. In *Rendering Techniques 2006*, pages 127–138, Nicosia, Cyprus, June 2006.

[16] B. D. Larsen and N. J. Christensen. Simulating photon mapping for real-time applications. In *Rendering Techniques 2004*, pages 123–132, 2004.

[17] M. Lastra, C. Urena, J. Revelles, and R. Montes. A particle-path based method for monte carlo density estimation. In *In Poster Papers Proceeding of the 13th Eurographics Workshop on Rendering*, pages 33–40, June 2002.

[18] F. Lavignotte and M. Paulin. Scalable photon splatting for global illumination. In *GRAPHITE 2003*, pages 203–301, 2003.

[19] T. MacRobert. Spherical harmonics; an elementary treatise on harmonic functions, with applications. Dover Publications, 1948.

[20] M. D. McCool. Anisotropic diffusion for monte carlo noise reduction. In *ACM Transactions on Graphics*, volume 18(2), pages 171–194, 1999.

[21] R. Ramamoorthi and P. Hanrahan. A signal-processing framework for reflection. In *ACM Transactions on Graphics*, volume 23(4), pages 1004–1042, 2004.

[22] A. Ravishankar Rao and Brian G. Schunck. Computing oriented texture fields. In *CVGIP: Graphical Models and Image Processing*, volume 53(2), pages 157–185, 1991.

[23] R. Schregle. Bias compensation for photon maps. In *Computer Graphics Forum*, volume 22(4), pages 729–742, 2003.

[24] M. A. Shah, J. Konttinen, and S. N. Pattanaik. Caustics mapping: An image-space technique for real-time caustics. In *IEEE Transactions on Visualization and Computer Graphics*, volume 13(2), March 2007.

[25] B.W. Silverman. Density estimation for statistics and data analysis. Chapmann and Hall, London, 1985.

[26] M. Smyk, S. Kinuwaki, R. Durikovic, and K. Myszkowski. Temporally coherent irradiance caching for high quality animation rendering. In *Computer Graphics Forum*, volume 24, pages 401–412, 2005.

[27] W. Stürzlinger. Calculating global illumination for glossy surfaces. In *Computers & Graphics*, volume 22, pages 175–180, 1998.

[28] F. Suykens and Y. D. Willems. Adaptive filtering for progressive monte carlo image rendering. In *Proceedings of WSCG*, 2000.

[29] L. Szirmay-Kalos, B. Aszódi, I. Lazányi, and M. Premecz. Approximate ray-tracing on the gpu with distance impostors. In *Computer Graphics Forum*, volume 24(3), pages 695–704, 2005.

[30] E. Tabellion and A. Lamorlette. An approximate global illumination system for computer generated films. In *SIGGRAPH: Proceedings of the 23rd annual conference on computer graphics and interactive techniques*, pages 469–476, 2004.

[31] I. Wald, J. Günther, and P. Slusallek. Balancing considered harmful – faster photon mapping using the voxel volume heuristic. In *Computer Graphics Forum*, volume 23, pages 595–603, 2004.

[32] I. Wald, T. Kollig, C. Benthin, A. Keller, and P. Slusallek. Interactive global illumination using fast ray tracing. In *Rendering Techniques 2002*, pages 9–19, 2002.

[33] B. Walter, S. Fernandez, A. Arbree, K. Bala, M. Donikian, and D. Greenberg. Lightcuts: a scalable approach to illumination. In *Proceedings of SIGGRAPH '05*, pages 1098–1107, 2005.

[34] G. J. Ward, F. M. Rubinstein, and R. D. Clear. A ray tracing solution for diffuse interreflection. In *Proceedings of SIGGRAPH '88*, pages 85–92, 1988.

[35] G. J. Ward, and P. Heckbert. Irradiance gradients. In *Eurographics Symposium on Rendering*, pages 85–98, 1992.

[36] M. Weber, M. Milch, K. Myszkowski, K. Dmitriev, P. Rokita, and H.-P. Seidel. Spatio-temporal photon density estimation using bilateral filtering. In *Computer Graphics International*, pages 120–127, 2004.

[37] Joachim Weickert. Anisotropic diffusion in image processing. Teubner, Stuttgart, 1998.

[38] C. Wyman and S. Davis. Interactive image-space techniques for approximating caustics. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 153–160, 2006.

[39] M. Zwicker, H. Pfister, J. van Baar, and M. Gross, Surface splatting. In *SIGGRAPH: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, 371–378, 2001.

Below you find a list of the most recent technical reports of the Max-Planck-Institut für Informatik. They are available by anonymous ftp from `ftp.mpi-sb.mpg.de` under the directory `pub/papers/reports`. Most of the reports are also accessible via WWW using the URL `http://www.mpi-sb.mpg.de`. If you have any questions concerning ftp or WWW access, please contact `reports@mpi-sb.mpg.de`. Paper copies (which are not necessarily free of charge) can be ordered either by regular mail or by e-mail at the address below.

Max-Planck-Institut für Informatik
Library
attn. Anja Becker
Stuhlsatzenhausweg 85
66123 Saarbrücken
GERMANY
e-mail: `library@mpi-sb.mpg.de`

.

MPI-I-2007-RG1-002	T. Hillenbrand, C. Weidenbach	Superposition for Finite Domains
MPI-I-2007-5-002	K. Berberich, S. Bedathur, T. Neumann, G. Weikum	A Time Machine for Text Search
MPI-I-2007-5-001	G. Kasneci, F.M. Suchanek, G. Ifrim, M. Ramanath, G. Weikum	NAGA: Searching and Ranking Knowledge
MPI-I-2007-4-006	C. Dyken, G. Ziegler, C. Theobalt, H. Seidel	GPU Marching Cubes on Shader Model 3.0 and 4.0
MPI-I-2007-4-005	T. Schultz, J. Weickert, H. Seidel	A Higher-Order Structure Tensor
MPI-I-2007-4-004	C. Stoll	A Volumetric Approach to Interactive Shape Editing
MPI-I-2007-4-003	R. Bargmann, V. Blanz, H. Seidel	A Nonlinear Viseme Model for Triphone-Based Speech Synthesis
MPI-I-2007-4-002	T. Langer, H. Seidel	Construction of Smooth Maps with Mean Value Coordinates
MPI-I-2007-4-001	J. Gall, B. Rosenhahn, H. Seidel	Clustered Stochastic Optimization for Object Recognition and Pose Estimation
MPI-I-2007-2-001	A. Podelski, S. Wagner	A Method and a Tool for Automatic Veriication of Region Stability for Hybrid Systems
MPI-I-2007-1-002	E. Althaus, S. Canzar	A Lagrangian relaxation approach for the multiple sequence alignment problem
MPI-I-2007-1-001	E. Berberich, L. Kettner	Linear-Time Reordering in a Sweep-line Algorithm for Algebraic Curves Intersecting in a Common Point
MPI-I-2006-5-006	G. Kasnec, F.M. Suchanek, G. Weikum	Yago - A Core of Semantic Knowledge
MPI-I-2006-5-005	R. Angelova, S. Siersdorfer	A Neighborhood-Based Approach for Clustering of Linked Document Collections
MPI-I-2006-5-004	F. Suchanek, G. Ifrim, G. Weikum	Combining Linguistic and Statistical Analysis to Extract Relations from Web Documents
MPI-I-2006-5-003	V. Scholz, M. Magnor	Garment Texture Editing in Monocular Video Sequences based on Color-Coded Printing Patterns
MPI-I-2006-5-002	H. Bast, D. Majumdar, R. Schenkel, M. Theobald, G. Weikum	IO-Top-k: Index-access Optimized Top-k Query Processing
MPI-I-2006-5-001	M. Bender, S. Michel, G. Weikum, P. Triantafilou	Overlap-Aware Global df Estimation in Distributed Information Retrieval Systems
MPI-I-2006-4-010	A. Belyaev, T. Langer, H. Seidel	Mean Value Coordinates for Arbitrary Spherical Polygons and Polyhedra in $\mathbb{R}^3$
MPI-I-2006-4-009	J. Gall, J. Potthoff, B. Rosenhahn, C. Schnoerr, H. Seidel	Interacting and Annealing Particle Filters: Mathematics and a Recipe for Applications
MPI-I-2006-4-008	I. Albrecht, M. Kipp, M. Neff, H. Seidel	Gesture Modeling and Animation by Imitation
MPI-I-2006-4-007	O. Schall, A. Belyaev, H. Seidel	Feature-preserving Non-local Denoising of Static and Time-varying Range Data

MPI-I-2006-4-006	C. Theobalt, N. Ahmed, H. Lensch, M. Magnor, H. Seidel	Enhanced Dynamic Reflectometry for Relightable Free-Viewpoint Video
MPI-I-2006-4-005	A. Belyaev, H. Seidel, S. Yoshizawa	Skeleton-driven Laplacian Mesh Deformations
MPI-I-2006-4-004	V. Havran, R. Herzog, H. Seidel	On Fast Construction of Spatial Hierarchies for Ray Tracing
MPI-I-2006-4-003	E. de Aguiar, R. Zayer, C. Theobalt, M. Magnor, H. Seidel	A Framework for Natural Animation of Digitized Models
MPI-I-2006-4-002	G. Ziegler, A. Tevs, C. Theobalt, H. Seidel	GPU Point List Generation through Histogram Pyramids
MPI-I-2006-4-001	A. Efremov, R. Mantiuk, K. Myszkowski, H. Seidel	Design and Evaluation of Backward Compatible High Dynamic Range Video Compression
MPI-I-2006-2-001	T. Wies, V. Kuncak, K. Zee, A. Podelski, M. Rinard	On Verifying Complex Properties using Symbolic Shape Analysis
MPI-I-2006-1-007	H. Bast, I. Weber, C.W. Mortensen	Output-Sensitive Autocompletion Search
MPI-I-2006-1-006	M. Kerber	Division-Free Computation of Subresultants Using Bezout Matrices
MPI-I-2006-1-005	A. Eigenwillig, L. Kettner, N. Wolpert	Snap Rounding of Bézier Curves
MPI-I-2006-1-004	S. Funke, S. Laue, R. Naujoks, L. Zvi	Power Assignment Problems in Wireless Communication
MPI-I-2005-5-002	S. Siersdorfer, G. Weikum	Automated Retraining Methods for Document Classification and their Parameter Tuning
MPI-I-2005-4-006	C. Fuchs, M. Goesele, T. Chen, H. Seidel	An Emperical Model for Heterogeneous Translucent Objects
MPI-I-2005-4-005	G. Krawczyk, M. Goesele, H. Seidel	Photometric Calibration of High Dynamic Range Cameras
MPI-I-2005-4-004	C. Theobalt, N. Ahmed, E. De Aguiar, G. Ziegler, H. Lensch, M.A. Magnor, H. Seidel	Joint Motion and Reflectance Capture for Creating Relightable 3D Videos
MPI-I-2005-4-003	T. Langer, A.G. Belyaev, H. Seidel	Analysis and Design of Discrete Normals and Curvatures
MPI-I-2005-4-002	O. Schall, A. Belyaev, H. Seidel	Sparse Meshing of Uncertain and Noisy Surface Scattered Data
MPI-I-2005-4-001	M. Fuchs, V. Blanz, H. Lensch, H. Seidel	Reflectance from Images: A Model-Based Approach for Human Faces
MPI-I-2005-2-004	Y. Kazakov	A Framework of Refutational Theorem Proving for Saturation-Based Decision Procedures
MPI-I-2005-2-003	H.d. Nivelle	Using Resolution as a Decision Procedure
MPI-I-2005-2-002	P. Maier, W. Charatonik, L. Georgieva	Bounded Model Checking of Pointer Programs
MPI-I-2005-2-001	J. Hoffmann, C. Gomes, B. Selman	Bottleneck Behavior in CNF Formulas
MPI-I-2005-1-008	C. Gotsman, K. Kaligosi, K. Mehlhorn, D. Michail, E. Pyrga	Cycle Bases of Graphs and Sampled Manifolds
MPI-I-2005-1-007	I. Katriel, M. Kutz	A Faster Algorithm for Computing a Longest Common Increasing Subsequence
MPI-I-2005-1-003	S. Baswana, K. Telikepalli	Improved Algorithms for All-Pairs Approximate Shortest Paths in Weighted Graphs
MPI-I-2005-1-002	I. Katriel, M. Kutz, M. Skutella	Reachability Substitutes for Planar Digraphs
MPI-I-2005-1-001	D. Michail	Rank-Maximal through Maximum Weight Matchings
MPI-I-2004-NWG3-001	M. Magnor	Axisymmetric Reconstruction and 3D Visualization of Bipolar Planetary Nebulae
MPI-I-2004-NWG1-001	B. Blanchet	Automatic Proof of Strong Secrecy for Security Protocols
MPI-I-2004-5-001	S. Siersdorfer, S. Sizov, G. Weikum	Goal-oriented Methods and Meta Methods for Document Classification and their Parameter Tuning
MPI-I-2004-4-006	K. Dmitriev, V. Havran, H. Seidel	Faster Ray Tracing with SIMD Shaft Culling
MPI-I-2004-4-005	I.P. Ivrissimtzis, W.-. Jeong, S. Lee, Y.a. Lee, H.-. Seidel	Neural Meshes: Surface Reconstruction with a Learning Algorithm