Exploiting Coherence in Lighting and Shading Computations

Robert Herzog

Dissertation zur Erlangung des Grades des
Doktors der Ingenieurwissenschaften
der Naturwissenschaftlich-Technischen Fakultäten
der Universität des Saarlandes

MPI Informatik
Campus E1 4
66123 Saarbrücken

Saarbücken, 27. April 2010

Betreuender Hochschuldozent – Supervisor

Dr.-Ing. Habil. Karol Myszkowski
MPI Informatik, Saarbrücken, Germany

Prof. Dr. Hans-Peter Seidel
MPI Informatik, Saarbrücken, Germany

Gutachter – Reviewers

Dr.-Ing. Habil. Karol Myszkowski
MPI Informatik, Saarbrücken, Germany

Prof. Dr. Hans-Peter Seidel
MPI Informatik, Saarbrücken, Germany

Prof. Dr. Kadi Bouatouch
INRIA and University of Rennes, Rennes, France

Dekan – Dean

Prof. Dr. Holger Hermanns
Universität des Saarlandes, Saarbrücken, Germany

Mitglieder des Prüfungsausschusses – Members of the Examination Board

Prof. Dr. Hans-Peter Seidel
MPI Informatik, Saarbrücken, Germany

Prof. Dr. Kadi Bouatouch
INRIA and University of Rennes, Rennes, France

Prof. Dr. Christoph Weidenbach (Vorsitzender – Chairman)
Universität des Saarlandes, Saarbrücken, Germany

Dr.-Ing. Thorsten Thormälen (Berichterstatter – Reporter)
MPI Informatik, Saarbrücken, Germany

Tag des Kolloquiums – Date of Defense

1. Juli. 2010

Acknowledgments

As usual, research is only efficient with the help of other people. Therefore, this section is dedicated to those people who either actively helped me in my projects during my PhD or positively influenced the progress and final outcome with fruitful discussions and ideas. First of all, I would like to thank my supervisor Karol Myszkowski for his support and his great ideas during my research time at MPI. Second, I like to give credits to Hans Peter Seidel for employing me as a PhD student in his group. I really appreciated the travel opportunities and the flexible and creative working environment at the MPI. I'd also like to thank Kadi Bouatouch who kindly agreed in reviewing this thesis. But what is research without help and discussions with your colleagues and co-workers. I owe thanks to Vlastimil Havran, who has also been my former supervisor at the start of my PhD. We were a good team and finished several projects in a short time period from which I have gathered much experience. Besides, he also contributed to my first paper. I also thank the student Shinichi Kinuwaki for supporting me in the implementation of the render2MPEG project. And I would like to thank my co-author Elmar Eisemann for helping me with my last paper. Elmar gave me new insights into GPU programming and turned out to be a really fast and skillful writer. I also thank all anonymous reviewers and those people who made their source code publicly available, like for example Jaroslav Křivánek for his radiance caching and hemispherical harmonics implementation. I am deeply grateful to my family in particular my sister for her good advice in designing the layout of this thesis. And last but not least, I really appreciate the support of my girlfriend Verena not only for proof-reading parts of my thesis but also for taking care of me.

Abstract

Computing global illumination (GI) in virtual scenes becomes increasingly attractive even for real-time applications nowadays. GI delivers important cues in the perception of 3D virtual scenes, which is important for material and architectural design. Therefore, for photo-realistic rendering in the design and even the game industry, GI has become indispensable. While the computer simulation of realistic global lighting is well-studied and often considered as solved, computing it efficiently is not. Saving computation costs is therefore the main motivation of current research in GI. Efficient algorithms have to take various aspects into account, such as the algorithmic complexity and convergence, its mapping to parallel processing hardware, and the knowledge of certain lighting properties including the capabilities of the human visual system.

In this dissertation we exploit both low-level and high-level coherence in the practical design of GI algorithms for a variety of target applications ranging from high-quality production rendering to dynamic real-time rendering. We also focus on automatic rendering-accuracy control to approximate GI in such a way that the error is perceptually unified in the result images, thereby taking not only into account the limitations of the human visual system but also later video compression with an MPEG encoder. In addition, this dissertation provides many ideas and supplementary material, which complements published work and could be of practical relevance.

Kurzfassung

Die Berechnung *globaler Beleuchtung* in virtuellen 3D-Szenen ist heutzutage selbst in Echtzeit-Anwendungen kaum mehr wegzudenken. Mittels globaler Beleuchtung wird die Wahrnehmung von virtuellen Szenen und Materialien viel realistischer. Daher sind Algorithmen zur globalen Beleuchtungsberechnung in der Industrie sehr gefragt. Obwohl die Computersimulation von realistischer globaler Beleuchtung oft als gelöst betrachtet wird, sind die Berechnungen dafür immer noch sehr komplex und die Ergebnisse konvergieren langsam. Somit ist die Hauptmotivation aktueller Forschung, die Kosten dieser Berechnungen so gering wie möglich zu halten. Abgesehen von der Komplexität und Konvergenz eines Algorithmus zur globalen Beleuchtungsberechnung werden heute oft weitere Aspekte in Betracht gezogen. So spielen z.B. die Parallelisierung für verteiltes Rechnen auf mehreren (Grafik-) Prozessoren, die menschliche visuelle Wahrnehmung und insbesondere die Ausnutzung bestimmter Lichteigenschaften eine wichtige Rolle in der Entwicklung effizienter Algorithmen.

In dieser Dissertation nutzen wir die Kohärenz der Beleuchtung aus, um praktische Algorithmen für ein grosses Anwendungsspektrum, von dynamischen Echtzeitanwendungen bis zu Anwendungen, die hochqualitative Ergebnisse liefern, zu entwickeln. Dabei berücksichtigen wir nicht nur die Defizite in der menschlichen visuellen Wahrnehmung, um Berechnungsfehler in Bildern möglichst uniform zu verteilen, sondern auch die verlustreiche Videokompression von Einzelbildern durch einen MPEG-Kodierer. Ausserdem umfasst diese Dissertation praktisches Ergänzungsmaterial und viele Ideen, die den Inhalt der schon publizierten Arbeiten bereichern.

Summary

This thesis introduces new efficient global illumination and caching algorithms specifically designed for sparse sampling and interpolation of lighting computations. Further, we consider the problem of exploiting temporal coherence in global illumination computations with focus on the specialized applications: video streaming and superresolution for rendered frames of an animation sequence. The methods are separated into chapters according to publications, which we summarize below.

Photon Ray Splatting

Photon density estimation is a very general approach for computing global illumination in synthetic scenes and can simulate all possible lighting effects very efficiently. However, by its nature photon density estimation is only capable of producing low-pass filtered illumination and is biased. Therefore, it is mainly intended for fast previewing or as input for second pass Monte Carlo integration. The goal of our work in Chapter 4 is to improve the quality of photon density estimation in order to produce more reliable results, which are also more robust against complex geometry. We achieve this by using a novel density estimation metric that is better suited for photon density estimation as it exploits additional knowledge gathered during photon sampling. More specifically, the traditional point density estimation borrowed from statistics is replaced by density estimation over photon rays. In contrast to previous work our metric decouples the density estimation entirely from the surface topology and is able to compute the illumination for geometry where the actual surface area is small or difficult to estimate, e.g., thin objects and wrinkled or “bump-mapped” surfaces. To facilitate the necessary nearest-neighbor-search for photon rays, we reverse the density estimation, which transforms the problem into a simpler one: searching for point data in a conical ray frustum, which we simply refer to as *ray splatting*. 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.

Like for all photon density estimation techniques the main limitation of our ray splatting is the neglected visibility in the density estimation footprint, which may result in light-leaking and blurred shadow boundaries. In follow-up work we extend our ray splatting with a method that preserves visibility using approximate volumetric occlusion tests in a hierarchical, discrete scene representation, which we quickly obtain by voxelizing synthetic scenes using programmable rasterization hardware. This way, our photon density estimation becomes less sensitive to the bandwidth selection and can even produce direct lighting in moderately complex scenes with acceptable quality.

Anisotropic Radiance Cache Splatting

High-quality production rendering has high demands on the visual richness of the lighting computation. Although our ray splatting improves the quality of the lighting reconstruction with photon density estimation, for production rendering purposes its quality is still moderate. Low-frequency noise may be visible and besides, our ray splatting is, like all photon density estimation methods, view-independent and therefore not efficient when computing illumination on glossy surfaces. Therefore, Monte Carlo integration with view-dependent importance sampling of the radiance function is the method of choice. However, Monte Carlo methods converge very slowly and are too expensive for computing high-resolution images. On the other hand, the indirect lighting before scattering with the BRDF is smooth and well suited for interpolation. Therefore, dedicated caching algorithms compute indirect lighting on demand at sparse surface locations and interpolate. This technique is called *irradiance caching*. In order to make irradiance caching practical, several heuristics have been proposed making the algorithm complex and difficult to control even for an experienced user. Hence, usual settings for tuning irradiance caching are overly conservative. One problem arises from the fact that irradiance caching is very sensitive to noise in the Monte Carlo integration because during interpolation high-frequency noise is turned into lower frequencies, which eventually leads to visible artifacts. The cause for the varying noise is blind importance sampling in the Monte Carlo integration without accounting for the lighting distribution. Another downside is the lazy generation of the cache data structure, which has been proposed in times where memory resources were scarce and whose performance and reconstruction quality is highly depending on the pixel traversal order and the number of the pixels in the image. Hence, in Chapter 6 we propose a radiance caching system with convenient control over the computation error. On one hand, we achieve this with a deterministic and perceptually conservative algorithm based on hierarchical radiosity called *lightcuts*, which replaces the traditional view-dependent integration pass. Thereby, we extend the lightcuts method to be used for traditional radiance caching in the hemispherical harmonics basis and derive translational and rotational gradients needed for higher-order interpolation. And second, we propose an adaptive two-layered caching data structure, which computes direct and indirect lighting in parallel, while respecting perceptual error criteria. To make our system more robust, we further apply an iterative cache refinement inspired by previous work, which we improve with anisotropic cache interpolation that better preserves lighting discontinuities and reduces the number of cache records needed for convergence. Overall, the method is suitable for high-quality walk-through animations and generates frames in less than a minute.

Render to MPEG

Another aspect of rendering is how to process and store the resulting video frames of animations, which is also becoming more important for bandwidth-limited streaming applications. So far, rendering animation frames and subsequent video encoding are two individual processes, which poses two main problems: First, stochastic noise in the rendered images is often preserved in the video and consumes extra bandwidth, while subtle details in shading might be eliminated by the quantizer of a lossy video encoder. Second, video encoding wastes a lot of computational power by computing motion vectors for minimizing residuals in the block matching between frames, whereas motion vectors can easily be computed precisely in the preceding rendering step. Therefore, in Chapter 7 we propose a technique, which couples a state-of-the-art global illumination renderer with an MPEG-2 encoder. Thereby, we exploit knowledge from the video encoder to steer the rendering. More precisely, we relax the rendering for pixels where an estimated upper error bound becomes smaller than the quantization error such that further details in the signal would be removed anyway. Our rendering is based on the *lightcuts* algorithm, which we have already utilized for efficient radiance caching in Chapter 6. However, here we exploit another useful property of lightcuts: the explicit control over the pixel-error. Apart from the quantization error, we also incorporate a perceptual error metric. This metric is directly computed in the frequency domain effectively using the discrete cosine transform inherent to MPEG encoding. Since all perceptual error metrics require knowledge of the lighting but the lighting computation is controlled by the error thresholds, previous perceptual rendering algorithms rely on too conservative assumptions, e.g., only consider the direct illumination. We show how to obtain the error thresholds before the actual lighting computation by motion-compensated warping of the thresholds derived from previous frame.

Spatio-Temporal Upsampling for Superresolution

For saving expensive pixel computations, in offline rendering it is common practice to employ adaptive caching algorithms that exploit coherence in the shading. However, regarding coherence in animations is not only important for offline rendering but can also help to speedup real-time rendering on graphics hardware. Recent advances in programmable graphics hardware facilitate real-time computations of complex shading and lighting algorithms. Unfortunately, the integration of a sophisticated adaptive caching algorithm like the one we presented in Chapter 6 is not suited for parallel processing on graphics hardware. Consequently, current GPU methods only marginally exploit spatial and temporal coherence. Furthermore, so far there exists no practical GPU algorithm which caches in space and time simultaneously. Hence, the goal of the method described in Chapter 8 is to provide a GPU-friendly architecture for speeding up costly real-time shading and lighting computations. For that we propose a framework for upsampling dynamic low-resolution results to high-resolution. The method runs in a single geometry pass entirely on the GPU and poses only a small constant overhead since no adaptive sampling nor any hole-filling step as for previous work is required. We achieve this by mixing a temporal and spatial strategy combined in a unified spatio-temporal upsampling algorithm. The first ingredient is a temporally interleaved sampling scheme, which ensures that frames are converging if shading is static. To compensate for dynamic scene motion, efficient reprojection techniques are applied. Second, an edge-preserving bilateral filter is used for spatially upsamping low resolution frames. The resulting spatio-temporal bilateral filter adapts to temporal gradients and geometric edges and provides fast response for dynamic shading. As a side-effect our method allows for temporal filtering of noisy signals. In addition, we propose a few extensions, such as antialiasing and a multi-scale approach, where low-frequency shading terms can be computed at a coarser scale.

Contents

List of Figures

List of Tables

Introduction

This thesis proposes several methods for accelerating intensive lighting and shading computations, which are applicable to various target applications reaching from high-quality off-line rendering to real-time rendering on graphics hardware. In this chapter, we motivate our accomplished research and give the reader a quick overview of the thesis and its contributions. At the end, we outline the structure of the whole thesis.

1 Motivation

Photo-realistic rendering has been and is one of the driving forces in computer graphics since the beginning. The probably most involved part in photo-realistic image synthesis is *global illumination* (GI). The word "global" refers to the fact that the illumination of an object depends on the light it receives from all other (non-occluded) objects. In this sense, computing the indirect illumination even at a single point requires consideration of the entire scene model. Even worse, this computation is of recursive nature since GI, in fact, relates to an infinite-dimensional integration problem. Despite of its complexity, solving the "complete" illumination is very attractive as it is one of the important perceptual cues of our visual system that gives us an impression of "photo-realism". For example, without indirect illumination surface-materials cannot be faithfully reproduced [NKGR06] and may even be misinterpreted as the correct visualization of a different material [K0̌5a]. Perhaps the crudest approximation that was introduced in image synthesis is to ignore the indirect illumination and only solve the direct illumination, which is usually much simpler to compute. While for certain applications pure direct illumination can be sufficient, it is easy to construct a scene where indirect lighting is the dominant or even the only light contribution to a virtual camera. Therefore, much research has been devoted to the topic of approximating the indirect illumination in a perceptually plausible manner.

On the other side, the computational hardware has evolved quickly over the last decade. Nowadays GI, although only coarsely approximated or precomputed, is even considered in real-time applications like games, which are the driving force of photo-realistic rendering and graphics hardware development. Because of the urgent desire for real-time GI in the past, many GI algorithms were designed based on crude approximations and heuristics solely for game-like applications in the spirit of “looks-good”. There the main objective is to get the *best possible quality in real-time* by exploiting parallel processing on modern graphics hardware. Nevertheless, computing GI in real-time remains challenging and cannot be compromised with simple heuristics. Therefore, a large gap between interactive but approximative GI techniques and high-quality photo-realistic methods had emerged in the past. This gap is nowadays slowly closing with faster and faster graphics hardware allowing computer games to take over advanced techniques, which have previously been dedicated to slow off-line rendering.

The contributions in this thesis are situated somewhere in-between high-quality production and real-time rendering covering a wider range of applications. We propose techniques that can converge to a high-quality solution as required by engineering applications and also give approximative but satisfactory results in short rendering times. We therefore aim at closing the gap between those extremes at a rather high level while still keeping future extensions in mind for adaptation to real-time rendering on parallel graphics hardware.

So far, in the movie and design industry the computation of single frames is in the order of minutes to hours rather than milliseconds because high-quality or predictive rendering is still very brute-force based on a vast amount of samples to compute the illumination exactly. On the other hand, just for visual entertainment unbiased results are not needed and one can simplify computations by making use of certain lighting properties. The key observation is: indirect lighting is spatially smooth and correlates with the local geometry, i.e. 3D points with similar location and orientation are likely to have similar illumination. This in turn allows us to compute illumination sparsely and interpolate afterwards. The computational savings, at least theoretical, are usually immense (two to three orders of magnitude). For example, for the synthetic scenes that we used in this dissertation, the indirect illumination can be reconstructed with high quality from only 2,000–20,000 samples as opposed to millions for high-resolution images. Therefore, this technique referred to as *irradiance caching* has become the de-facto standard and is widely used in the movie industry [TL04, CB04]. Although not directly applicable to parallel processing in computer games, the idea of sparse sampling and smart interpolation techniques is slowly entering the game industry, which is confronted with increasing display resolutions. In contrast, here adaptive “optimal” sampling is traded with parallelism for coherent processing.

Despite the recent advances in rendering algorithms, several issues have gained little attention so far. For example direct lighting has mainly been considered as inexpensive enough to be computed in a brute-force, exact manner. This is not true for complex scenes with many area lights. Second, the indirect lighting is not only spatially coherent but also temporally. But temporal smoothness in the lighting function is little exploited in most approximative GI algorithms. This is particularly surprising as temporal coherence has been heavily exploited in video compression. By taking advantage of temporal and spatial coherence we can achieve much more robust results even leveraging the lack of spatial adaptivity in games since in most situations lighting is either spatially or temporally coherent. Therefore, one aspect of this thesis is how to exploit temporal *and* spatial coherence in lighting computation. Another often neglected topic for irradiance caching is how to make irradiance caching practical and robust. For most algorithms the computation of the cache samples is too conservative and parameters are manually controlled by an experienced user. Overly conservative cache settings may quickly outweigh the initial performance gains by exhaustive cache searching and interpolation with non-trivial weighting functions. Thus, one objective of this thesis is to reconcile these differing goals: speedup, generality, and simplified user-control of caching techniques.

The contributions in this thesis are two-fold: first, we propose a novel technique for approximating the indirect illumination based on better radiance estimation with photon maps that exploits additional information gathered during photon sampling. The second contribution consists of novel approaches for the reduction of pixel computations, either by efficient space and time caching with sophisticated perceptually-motivated interpolation or by exploiting the limits of the human visual system and video compression. In this context we propose two caching techniques designed either for real-time or off-line rendering.

2 Main Contributions

This section gives a list of all my publications and their contributions. The work in this thesis is based on these five publications: [HHK*07a, HS07, HKMS08, HMS09, HEMS10], whose contributions are summarized below.

The work in [HHK*07a] has been published in the proceedings of the *Eurographics '07* conference and improves traditional photon mapping. More specifically, it introduces a new density estimation metric tailored for photon density estimation. Additionally, the paper proposes several new extensions, which are also applicable to traditional photon mapping. In summary we proposed:

- A **new photon density estimation metric** that leads to **reduced bias** in the radiance estimate in particular for **complex 3D scenes**.
- A novel way for **efficient bandwidth selection for photon splatting** approaches, which is the most critical part of any density estimation.
- An **acceleration data structure** for **searching points in a conical**

frustum.

- Post-process image-filtering to suppress noise in the radiance estimate while preserving gradients.

Overall, the method is very general as it is based on photon density estimation and it is well-suited for generating reliable previews within a few seconds for arbitrary shading in complex scenes. Additionally, in a follow-up paper [HS07] published in *Pacific Graphics '07*, we extended the method with a light-weight approximation of the visibility term that is hidden in the density estimation kernel. This extension allows for reproducing sharp boundaries in the illumination with fewer photons on the expense of a slightly more complex photon traversal.

The work published in the *Eurographics '09* proceedings [HMS09] is intended to speedup the computation of global illumination in high-quality production-style rendering. It is based on two well-known algorithms, *radiance caching* [KGPB05, WRC88] and *lightcuts* [WFA*05], which speedup the global illumination computation. Although both algorithms seem to be orthogonal, their combination is not trivial and has not been considered before. How to practically integrate lightcuts into radiance caching and achieve substantial performance gains is therefore the main contribution of [HMS09], which is further described in Chapter 6. In summary, the contributions are:

- Novel combination of **radiance caching with lightcuts** (*hierarchical instant radiosity*), which leads to a significant speedup and reduces the space of user-parameters compared to traditional radiance caching algorithms.
- An improved radiance cache interpolation with **anisotropic reconstruction** and **translational and rotational radiance gradients for lightcuts**, which better adapts to the lighting function and avoids blurring across edges (e.g., indirect shadows).
- **Two-layered radiance caching** for simultaneous computation of direct and indirect lighting, where the two layers mutually steer the cache refinement by incorporating **perceptual masking thresholds**.

The focus in [HKMS08] is perceptual rendering in the context of animation sequences. The paper has been published in the proceedings of *Eurographics '08* conference. Here the novelty is a perceptual error metric that takes advantage of the lossy video compression of rendered animation frames. This way we are able to adapt the rendering quality and convergence of a global illumination algorithm not only to the human perception but also to the compression level of standard MPEG encoders, which has not been considered so far. The method is described in Chapter 7 and its contributions are:

- A novel **perceptual error-metric based on the discrete cosine transform**, which takes into account the **quantization of an MPEG encoder**.
- A strategy for the **prediction of perceptual error thresholds** that steer

the global illumination computation of a frame, which are based on the thresholds computed from previous frames in an animation sequence.

- Adaptation of the lightcuts algorithm to dynamic scenes.

The method described in Chapter 8 is specifically designed for real-time rendering on the GPU and is based on my recent publication [HEMS10] in *ACM Siggraph Symposium on Interactive 3D Graphics and Games 2010*. According to feedback from the game industry the method will have some impact in the future game development. Its contributions are:

- An **efficient upsampling** scheme for dynamic low-resolution frames, which **exploits both temporal and spatial coherence**.
- **Lightweight single-pass upsampling** well-suited for **realtime GPU shading**.
- A **novel GPU-suitable interleaving pattern** for sparse sampling over space and time.
- A proposal for **adapting the exponential filter weights**, used for reprojection caching, to temporal pixel gradients for **faster response** and **attenuation of noise**.

3 Thesis Outline

The basis of the thesis are my published papers and their content is separated into individual chapters, which are organized in two main categories: global illumination algorithms and sparse sampling and reconstruction techniques. As some chapters describe extensions to methods introduced in previous chapters we may eventually cross-refer.

In addition to my published work, there are several new sections with novel contributions making the individual methods more comprehensible and profound. In particular Chapters 4, 6 and 8 extend the originally published material significantly.

The thesis is organized as follows. In the next chapter we will review the background knowledge in realistic image synthesis and visual perception, which is needed in order to understand the proposed methods in this thesis. We will then outline related work in Chapter 3. The actual contributions of this thesis begin with algorithms targeted at efficient global illumination rendering. In Chapter 4 we present an approximative global illumination method, *photon ray splatting*, which is then extended with approximate indirect visibility queries described in Chapter 5. In Chapter 6 we propose a method aiming at high-quality and robust global illumination rendering based on the well-known *lightcuts* algorithm [WFA*05], which we extended for use with *radiance caching*. As opposed to previous chapters, in Chapter 7 we focus on perceptual issues of lighting computations and how to adapt the rendering quality to the lossy video compression. At last, in Chapter 8 we present a different approach that exploits spatial and in particular temporal coherence of shading in dynamic 3D scenes, which maps well to current parallel graphics hardware. Finally, we conclude in Chapter 9 and collect ideas for future research.

4 Naming Convention

In the following chapters we will use the symbols and naming convention shown below.

$x, y, z, \ldots$	Scalar values
$\mathbf{x}, \mathbf{y}, \ldots$	Points
$\vec{\mathbf{x}}, \vec{\mathbf{u}}, \ldots$	Vectors
$\vec{\mathbf{u}}, \vec{\mathbf{v}}, \vec{\mathbf{n}}$	local tangent frame $\vec{\mathbf{u}}, \vec{\mathbf{v}}$ and surface normal vector $\vec{\mathbf{n}}$
$\mathbf{x}_0, \mathbf{x}_1, \ldots$	Light path vertices (e.g. photon hitpoints, $\mathbf{x}_0$ origin on light source)
$\boldsymbol{\omega}_o, \boldsymbol{\omega}_i$	Outgoing and incoming light directions, respectively
θ, ϕ	Polar angle and azimuthal angle relative to surface normal
$\Phi(\mathbf{x}, \boldsymbol{\omega})$	Incoming flux (photon) at $\mathbf{x}$ from direction $\boldsymbol{\omega}$
$L(\mathbf{x}, \boldsymbol{\omega}_o), L(\mathbf{x}, \boldsymbol{\omega}_i)$	Radiance in direction $\boldsymbol{\omega}$ leaving $\mathbf{x}$ and arriving at $\mathbf{x}$, respectively
$E(\mathbf{x})$	Irradiance at $\mathbf{x}$
$\mathcal{K}_h(\mathbf{x}, \mathbf{y})$	Normalized 2D Kernel function at $\mathbf{x}$ with bandwidth h: $\mathcal{K}_h(\mathbf{x}, \mathbf{y}) = \frac{1}{h}\mathcal{K}\left(\frac{\|\mathbf{x}-\mathbf{y}\|}{h}\right)$
$\mathcal{K}_h(\mathbf{x}, \mathbf{y}, \boldsymbol{\omega})$	Normalized 2D kernel function $\mathcal{K}_h(\mathbf{x}, \mathbf{y})$ with domain oriented perpendicular to $\boldsymbol{\omega}$
$d\boldsymbol{\omega}$	Differential solid angle for direction $\boldsymbol{\omega}$
$dA, \Delta A$	Differential surface area, finite surface area
$dA_\theta^\perp$	Projected differential surface area, $dA_\theta^\perp = dA \cdot \cos\theta$
$V(\mathbf{x}, \mathbf{y})$	Visibility predicate, if $\mathbf{x}$, $\mathbf{y}$ are mutually visible then $V(\mathbf{x}, \mathbf{y}) = 1$, else $V(\mathbf{x}, \mathbf{y}) = 0$
$f_s(\mathbf{x}_{i-1}, \mathbf{x}_i, \mathbf{x}_{i+1})$	BRDF describing scattering of light: $\mathbf{x}_{i-1} \rightarrow \mathbf{x}_i \rightarrow \mathbf{x}_{i+1}$
$f_s(\mathbf{x}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o)$	BRDF at $\mathbf{x}$ for scattering light from incident direction $\boldsymbol{\omega}_i$ to outgoing direction $\boldsymbol{\omega}_o$
$p(x)$	Probability density function (pdf)
$p(\mathbf{y}\|\mathbf{x})$	Conditional pdf for sampling light path vertex $\mathbf{y}$ given vertex $\mathbf{x}$
$Y_l^m(\theta, \phi)$	Spherical harmonics (SH) basis function of degree m, band l, with index $i = l(l+1) + m$
$f_l^m(\theta, \phi)$	BRDF SH coefficients for the outgoing direction $\boldsymbol{\omega}_o = (\theta, \phi)$
λ_l^m	SH coefficients representing incoming radiance

Background

1 Radiometry and Photometry

Light is a complicated phenomenon and it is still not completely understood. Since the earliest days human beings have been fascinated by the perception of light because it carries the richest source of information to the human's most powerful sense. Many theories explaining *light* have been proposed over the centuries. In former days the Greeks believed that "light" starts in our eye which is emanating rays that touch the environment we see. Interestingly, this phenomenon is mathematically valid and is heavily exploited in Computer Graphics as the *importance* or *potential* transport, which flows in opposite direction of radiant energy. It took about 1000 years before the scientist Alhazen (A.D. 956-1038) described the light by straight rays and modeled the eye with a pinhole camera called *camera obscura*. More than 600 years later, Christiaan Huygens and Isaac Newton came up with two different novel theories. Huygens demonstrated the wave optics of light whereas Newton supported his light particle theory and argued heavily against Huygens wave model. From the 19th century on, researchers like Augustin Fresnel and Thomas Young studied effects such as polarization and diffraction. James Maxwell further described the properties of electromagnetic waves and in the 20th century several physicians, Max Planck, Albert Einstein, Werner Heisenberg amongst others manifested the light quantum theory, the cradle of the *photon*. At that time it became clear that light could not be explained by either theory. This formalized the *dualism of light* established by the field of quantum mechanics. Therein, light is classified into ray/wave optics, and photon (particle) optics. For most computer graphics applications, like the ones presented in this thesis, we are mostly concerned with a highly abstract model based on ray optics, even for *photon mapping* [Jen01]. We further neglect important properties of light, e.g., the time-propagation of light. Nevertheless, even with such simplified models we can simulate most light phenomena that we are able to see in our daily life in a plausible way.

1.1 Light Terminology

In order to describe light and the transport of energy, basic terminologies must be introduced. There are two different definitions, the *radiometry* which is physically based and the *photometry*, which takes into account the human eye's sensitivity to various wavelength. We will first introduce the radiometric quantities together with their corresponding units, which is more accepted in science and generally used in computer graphics. The smallest quantity in lighting is the photon. The energy of a photon with wavelength λ is $e_\lambda = \frac{h \cdot c}{\lambda}$ where h is the Planck's constant $h \approx 6.63 \cdot 10^{-34} J \cdot s$ and c is the speed of light.

- *Radiant energy* Q is the total energy gathered over time from all n_λ photons for each wavelengths λ measured in watts W per second s

$$Q = \int_0^\infty n_\lambda e_\lambda d\lambda \ \ [W \cdot s]. \tag{2.1}$$

- *Radiant flux* Φ is defined as radiant energy per time

$$\Phi = \frac{dQ}{dt} \ \ [W] \tag{2.2}$$

  It represents the time rate of flow of radiant energy, which is captured by photons.

- *Irradiance/ Radiosity*: the general terminology *radiant flux area density* is usually separated into *irradiance* E and *radiosity* B, also known as radiant exitance. It represents the flux density on a surface which is defined as radiant flux per differential area

$$E(\mathbf{x}) = \frac{d\Phi}{dA} \ \ \left[\frac{W}{m^2}\right]. \tag{2.3}$$

  The difference between irradiance and radiosity is that irradiance corresponds to the flux arriving on a surface and radiosity corresponds to the scattered flux leaving the surface.

- *Radiant intensity* I represents the directional density of flux (i.e., the flux coming from a certain direction) and is defined as flux per differential solid angle $d\omega$

$$I(d\omega) = \frac{d\Phi}{d\omega} \ \ \left[\frac{W}{sr}\right] \tag{2.4}$$

  The solid angle is explained in Section 1.2.

- *Radiance* L can be considered as a product of directional density and area density of flux which is dependent on the surface orientation (therefore the division by the cosine of the polar angle θ). It is probably the most important quantity in computer graphics because this is what we measure with *ray tracing*. It is defined as radiant flux per differential solid angle per differential projected area.

$$L(\mathbf{x},\omega) = \frac{dE}{\cos\theta \, d\omega} = \frac{d^2\Phi}{\cos\theta \, d\omega \, dA} \quad \left[\frac{W}{m^2 \cdot sr}\right] \tag{2.5}$$

Informally, it can be understood as the number of photons per time arriving on a surface with area dA and from a certain direction ω with small angular deviation defined by the solid angle $d\omega$. Therefore, radiance is a five-dimensional quantity. Another important aspect is that radiance stays constant along a line in vacuum. We can also refine the radiance definition to include the wavelength, which gives us the *spectral radiance* L_λ, the radiance for a certain differential wavelength.

1.1.1 The Relationship between Radiometric Quantities

From the definitions of the radiometric quantities, the following relationship can be derived:

$$\Phi = \int_A E(\mathbf{x})dA = \int_A \int_\Omega L(\mathbf{x},\omega)(\omega \cdot \vec{n})d\omega \, dA, \tag{2.6}$$

where $\vec{n}$ is the surface normal at $\mathbf{x}$ and $\omega \cdot \vec{n} = \cos\theta$. If the complete radiance field on a surface is available then the irradiance/radiosity can be computed by integrating the incident/exitant radiance field over all directions. The incident/exitant flux is computed by integrating the irradiance/radiosity over the area.

1.1.2 Photometry

The difference between radiometry and photometry is that photometric quantities include the visual response of the average observer. For example, luminous flux Φ_v is the visual response to radiant flux

$$\Phi_v = \int_\Lambda \Phi_\lambda V(\lambda)d\lambda, \tag{2.7}$$

where $V(\lambda)$ is the *luminous efficiency function*, which measures the visual response of a standard observer, see Fig. 2.1. And Λ is the visible spectrum ($\approx$ 380 nm – 780 nm). *Illuminance*, E_v, is the counterpart to irradiance and *luminous exitance*, M_v, to radiosity or radiant exitance, respectively. *Luminous intensity*, I_v, is the photometric quantity that corresponds to radiant flux and *luminance* is the photometric equivalent of radiance. The radiometric quantities are often preferred in global illumination algorithms but the visual response plays an important role when rendered images are post-processed before display via *tone mapping* (see Section 4.4).

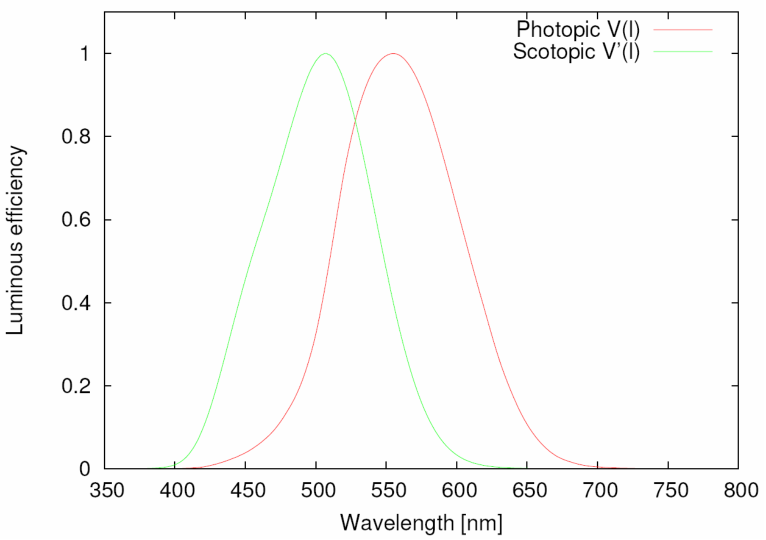


Fig. 2.1 – Luminous efficiency function

CIE spectral luminous efficiency curve for photopic (day light) and scotopic (night) vision. Data downloaded from http://www.cvrl.org/.

1.2 Solid Angle

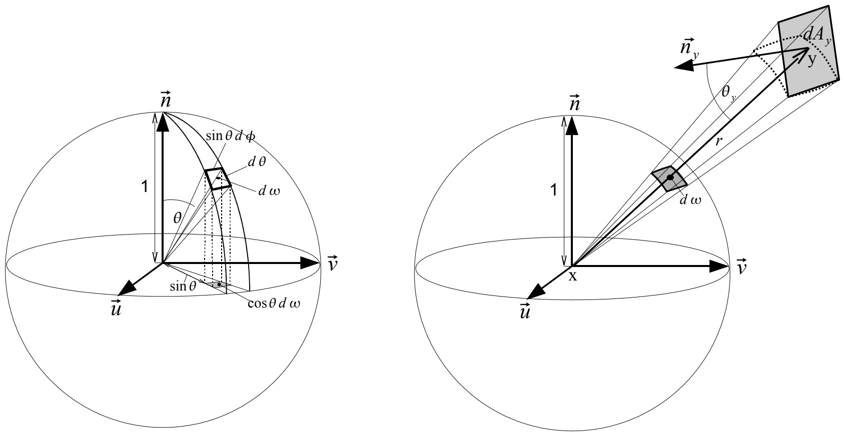
In order to integrate functions over the hemisphere which becomes necessary for computing radiometric quantities, a specific measure defined on the hemisphere is needed. This is the *solid angle*. The finite solid angle Ω corresponds to the projection of a surface onto the unit sphere

$$\Omega = \frac{A}{r^2}, \tag{2.8}$$

where A is the projected area on the sphere with radius r. Scientifically, the solid angle is dimensionless but for convenience it is expressed in *steradians* [sr]. The differential solid angle $d\omega$ can be thought of as the angular size of an infinitesimal thin beam. When parametrized in spherical coordinates, the beam direction ω is expressed as (θ, ϕ), corresponding to the polar (longitude) and azimuthal (latitude) angle, respectively. And its size is defined as the infinitesimal area on the unit sphere computed as the product of the longitude arc $d\theta$ and the latitude arc $\sin\theta\, d\phi$ (see Fig. 2.2 (a)). The transformation from a differential surface with differential area dA_y, distance r_y and orientation θ_y to a differential solid angle $d\omega$ is given by

$$d\omega = \frac{\cos\theta_y dA_y}{r_y^2}. \tag{2.9}$$

This relationship is visualized in Fig. 2.2 (b).



(a) (b)

Fig. 2.2 – A geometric interpretation of the solid angle

(a) The area subtended by a differential solid angle $d\omega$ *is the product of the differential length of the longitude arc* $d\theta$ *and the latitude arc* $\sin\theta\ d\phi$*. (b) The differential solid angle* $d\omega$ *subtended by a differential area* dA_y *is equal to* $dA_y \cos\theta_y / r^2$*, where* θ_y *is the angle between* dA_y*'s surface normal and the vector to the point* $\boldsymbol{x}$ *and* r *is the distance from* $\boldsymbol{x}$ *to* $\boldsymbol{y}$*. In other words,* $d\omega$ *is the projection of the differential area* dA_y *onto the unit sphere around point* $\boldsymbol{x}$*.*

2 Light Interaction with Surfaces

Light alone is not very valuable for us, but the interaction with the scene model (the illumination) shows the variety of our environment. Looking at arbitrary real surfaces reveals the complexity of such light interaction. Instead of physically simulating the light scattering inside a surface's material, in computer graphics light-surface-interactions at *micro-scopic scale* are treated as a "black-box" and only the measured radiance as a result of the scattering event is observed. However, in real world light is reflected and transmitted in numerous ways and the general light-surface interaction is a 9-dimensional problem (ignoring time-dependency), depending on the wavelength, the incident and reflected direction, the surface location of the incident and even the location of the exitant radiance. To describe this kind of scattering by a mathematical function suitable for simulation with a computer is generally difficult because of the high-dimensionality of the problem. Therefore, one first reduces the dimension to 8 by neglecting wavelength dependencies and considering only a few (usually three) constant basis functions of the continuous color spectrum. Functions of this general type are still inappropriate for efficient representation and simulation by a computer. Besides many materials, especially artificial ones, do not change across the surface and few materials exhibit noticeable sub-surface scattering effects. In case a material is homogeneous we can reduce the scattering to 6 dimensions discarding the spatial variation. The type of functions is called *bidirectional scattering surface reflectance distribution function* or short *BSSRDF*. On the other hand, if we assume that a material varies across a surface but does not show visible subsurface scattering effects, there is no need to consider the surface neighborhood around a point of a light scattering event but assume that incident and exitant location of the light are the same. This type is classified as *bidirectional texture function BTF* [Dis98, MMS*05]. Although, the acquisition and storage of such 6D functions is still expensive and time consuming, BTFs convey much more realism and are therefore often used for predictive and photo-realistic rendering in the movie and design industry. The measured data can become very large (several GBytes) for a single material but is quite redundant, which imposes various compression schemes. Note that measured BTFs inherently include sub-surface scattering effects.

The BSSRDF accounts for all sub-surface scattering effects of light such as scattering in translucent materials (e.g., milk, skin, snow, alabaster) excluding time- (e.g. phosphorescence) and wavelength-dependent effects (e.g. fluorescence). Subsurface scattering is the dominant effect in many organic materials [NKGR06]. However, similar to BTFs, BSSRDFs are still too complex for efficient interactive simulations and are mostly addressed by off-line computations.

For interactive or real-time applications the complex light scattering in materials is further simplified. In many cases, the materials are invariant with respect to sub-surface scattering and the two spatial degrees of freedom in the BTF can be resolved by a surface subdivision to piecewise constant materials. In this case the scattering function is restricted to 4 dimensions and a new name is assigned for this representation: *bidirectional scattering distribution function* (BSDF). In a lighting simulation it is a common practice to separate reflecting (upper hemisphere) and transmitting properties (lower hemisphere of directions) of a material. This is why BSDFs of reflected and transmitted light are usually considered separately and are called *BRDF* (*Bi-directional Reflectance Distribution Function*) and *BTDF* (*Bi-directional Transmittance Distribution Function*) respectively. However, for simplicity, we will not regard transmitting properties of a material here.

2.1 The BRDF

The *BRDF*, denoted by f_s, describes the interaction of light with a surface neglecting sub-surface scattering events – we simply assume the light is reflected at the same location it is arriving. The BRDF is defined as

$$f_s(\mathbf{x},\omega_i,\omega_o) = \frac{dL(\mathbf{x},\omega_o)}{dE(\mathbf{x},\omega_i)} = \frac{dL(\mathbf{x},\omega_o)}{L(\mathbf{x},\omega_i)\cos\theta_i d\omega_i} \quad \left[\frac{1}{\mathrm{sr}}\right] \tag{2.10}$$

and tells us what fraction of the differential irradiance dE coming from direction ω_i with infinitesimal small solid angle $d\omega_i$ is reflected at surface point $\mathbf{x}$ towards direction ω_o. A BRDF can yield any positive value even approaching infinity in case of a perfect (however idealized) mirror where all but *one infinitesimal small* direction are zero resulting in a *dirac impulse*. Intuitively, the BRDF can be understood as a probability density function which describes the probability of a photon coming from direction ω_i to be reflected in direction ω_o. However, a BRDF does not need to integrate to one over the hemisphere of directions. In fact, it integrates to something lower than one, which is referred to as *energy conservation* and states that the reflected radiant energy can not exceed the incident radiant energy, i.e.,

$$\int\limits_{\Omega} f_s(\mathbf{x},\omega_i,\omega_o)\cos\theta_i d\omega_i \leq 1. \tag{2.11}$$

Having the full (hemispherical) incident radiance field available, we can build a local illumination model: the reflected radiance in all directions ω_o:

$$L_r(\mathbf{x},\omega_o) = \int\limits_{\Omega_i} f_s(\mathbf{x},\omega_i,\omega_o) dE(\mathbf{x},\omega_i) = \int\limits_{\Omega_i} f_s(\mathbf{x},\omega_i,\omega_o) L_i(\mathbf{x},\omega_i)\cos\theta_i d\omega_i, \tag{2.12}$$

where the cosine term is computed as $\cos\theta_i = \vec{n}_x \cdot \omega_i$, the dot product of the normal at surface point $\mathbf{x}$ and the incoming light direction ω_i.

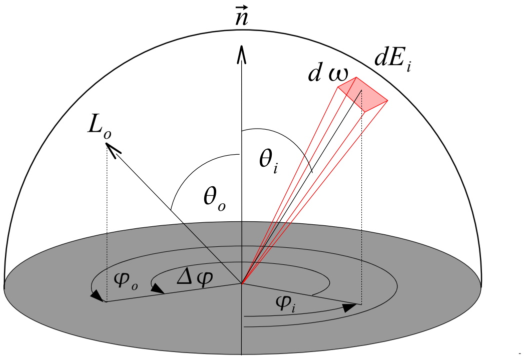


Fig. 2.3 – A geometric interpretation of the BRDF

A geometric interpretation of the BRDF for a point on a surface. The BRDF function gives the ratio of radiance L_o reflected in direction (θ_o,ϕ_o) to the differential irradiance dE_i of differential solid angle $d\omega$ from direction (θ_i,ϕ_i). In case of an isotropic BRDF, only the difference $\Delta\phi = \phi_o - \phi_i$ in the latitude angles ϕ_i and ϕ_o is regarded.

The BRDF has four dimensions as it depends on four angles $(\omega_i,\omega_o) = (\theta_i,\phi_i,\theta_o,\phi_o)$ giving 4 degrees of freedom. However, for most materials the rotation around the surface normal is often redundant. In such case of *isotropic* materials only the relative latitude angle of incident to reflected direction $(\phi_i - \phi_o)$ is important (see Fig. 2.3). Therefore, the BRDF of isotropic materials can be reduced to three dimensions. Another simplification can be made for diffuse surfaces because they are hardly view-dependent. We assume that all incident light is equally distributed in all directions resulting in a constant BRDF at a surface point, which can easily be modulated by a texture. Such a surface is called *Lambertian*. Most walls and wallpapers of indoor environments are very close to be Lambertian.

Physically correct BRDFs obey the *Helmholtz law of reciprocity*, which states that the BRDF is independent of the light-flow direction:

$$f_s(\mathbf{x}, \omega_i, \omega_o) = f_s(\mathbf{x}, \omega_o, \omega_i) \tag{2.13}$$

This is a fundamental law most global illumination algorithms make use of, since it enables to trace light paths in both directions: from the eye and from the light source.

2.2 Shading Models

Usually in rendering, material properties are only dealt with statistically at microscopic scale (i.e., material structure is beyond the light wavelength), which is represented by the BRDF introduced in the previous section. Because of the high-dimensionality of the BRDF (4D), a lookup table of real measured data can become very large and inconvenient for efficient rendering. On the other hand, many materials consist of low frequency BRDFs with simple shapes that can be efficiently compressed or even modeled with analytical models called *shading models*, which can be fitted to the measured data. Typically, only angular material-variations are modeled with analytic shading models, while the spatial variation is attained separately by a modulation of the model's parameters with so-called *textures*. Such texture modulation can either be represented with procedural functions (procedural textures) or with sampled images parametrized and glued to 2D surfaces, or 3D objects in case of 3D textures. Shading models are very popular in rendering and have the advantage of low storage costs, efficient computation, and easier importance sampling of the BRDF (see Section 3.5). Therefore, we will not regard measured and compressed BRDFs but rather use shading models in the sequel.

Various shading models of different complexity have been proposed and a comprehensive introduction is out of the scope of this thesis. Shading models can be classified into empirical models, often non-physically correct (e.g., Phong [Pho75], Blinn [Bli77]) and physically-based models (e.g., Cook-Torrance [CT81], Ward [War92], Schlick [Sch94]).

A common assumption of all popular shading models is the decomposition into a diffuse (*Lambertian*) and glossy component. The diffuse part is constant and hence invariant with the viewing direction. The more complex glossy part is often modeled by one [Pho75, Bli77] or several specular lobes [LFTG97] usually oriented close to the mirror direction of the view point. This principle eases also the computation of the reflected radiance and is exploited in many global illumination algorithms, e.g., *radiosity*-based finite-element algorithms, *photon mapping* [Jen96].

3 Rendering and Image Formation

Rendering is a specific field in computer graphics dealing with the synthesis of images and can be divided into *non-photo-realistic* and *photo-realistic* rendering. We will focus on photo-realistic rendering, which is the synthesis of naturalistic-looking images. It can be regarded as the process of simulating the physical light transport within the limits of perception in a virtual three-dimensional scene that is observed through a virtual camera. Thereby, physical processes and phenomena of light are adapted to our perceptual and device specific limits. Although often obeying physical laws, the execution steps of rendering algorithms in particular the order of the computation does not need to be physically related. What matters is the final outcome and speed/convergence of its computation. The outcome is usually either a single image or in case of an animation a *frame* in a video sequence. The main focus of photo-realistic rendering is the *efficiency* of the simulation of various real-world lighting and shading effects. This is different to other fields in computer graphic dealing with measured data because in rendering we can nearly always obtain ground-truth results by simply applying unbiased algorithms. Hence, what matters is the quality-time tradeoff in the computation. Rendering algorithms are classified as interactive or real-time if the time to compute *one* displayable frame is within a fraction of a second or without any noticeable delay, respectively. Algorithms, which take several seconds, minutes or even hours and more to compute a frame are referred to as off-line.

3.1 Digital Images

Images are perhaps the most common data structure in rendering and represent a visual signal on a 2D sensor like a camera CCD or the human retina. They are often used as input, e.g., *textures*, and are usually the output of the computation that can directly be visualized. More formally we interpret an image of width w and height h as a mapping

$$I : [0,w) \times [0,h) \to \mathbb{R}. \tag{2.14}$$

This definition in Eq. 2.14 is for a continuous image. In practice, we have to deal with digital images, discretized to a certain resolution $w \times h$ sampled on a regular

2D grid at integer positions, which are named *picture elements* or short *pixels*. Those pixels form a matrix of scalar (= grey values) or vector (= color) data.

Such discretization boils down to a 2D sampling problem. According to sampling theory, if we assume the image signal is band-limited, we need to sample it at least at twice the highest frequency, the *Nyquist frequency*, to avoid *aliasing*. In practice this is not always feasible as an image can consist of infinitely many frequencies. Therefore, the input image signal needs to be properly *low-pass filtered* (blurred) *before* sampling it at lower frequencies. This process is referred to as *anti-aliasing*. While for real image acquisition with optical camera devices a blur filter can be sufficient to band-limit the image signal, in synthetic image rendering the image signal is largely unknown before sampling it. Therefore, to reduce aliasing it is necessary to compute the image at a higher resolution than the final image resolution, which is called *super-sampling*. However, simply computing the image regularly at higher and higher resolutions, does not solve the problem in general as it can still suffer from aliasing at even higher-frequencies. Hence, a common approach is to transform aliasing into stochastic noise by employing *irregular* sampling patterns to distribute sub-pixel samples. Common methods are *Poisson disc*, *stratified random*, *pseudo-random* sampling [PH04].

Different image representations like *Wavelets*, *Fourier basis*, *Discrete cosine transform* (DCT), *Hough space*, analytic 2D vector graphics were developed each one better suited for specific tasks like for example compression, filtering and processing, manipulation, or object recognition. In terms of synthetic image rendering the digital image is usually formed by a projection of 3D data to a virtual screen, the *digital imaging sensor*.

3.2 3D Scene Representation

Realistic image synthesis is the process of computing a virtual camera view in a three-dimensional scene. For this purpose a geometric model of a 3D scene is necessary, which can have very different representations (e.g., points, polygons, implicit surfaces, splines surfaces, etc.). Most popular and efficient in terms of rendering are the 3D representations based on simple primitives as listed below.

3.2.1 Image-Based Rendering

The traditional approach of computer graphics is to create a geometric model in 3D and project it onto a two-dimensional image. Conversely, in *image-based modeling and rendering* (IBMR) we do not know the (exact) 3D model of the scene but only know a set of two-dimensional images representing different camera views in this scene. Knowing the camera position and orientation corresponding to those images, we can directly compute novel views by interpolating the nearest images. In IBMR we skip the manual 3D modeling stage and can deal with real-world scenes independent of their complexity. On the other hand, unless a huge amount of images is used, artifact-free blending of the images requires to estimate depth values, which is a slow and non-trivial task. Further, view-dependent glossy materials and dynamic scenes are difficult to handle and require many images in order to faithfully reproduce new images. IBMR is not only a useful and appealing approach when the scene geometry is unknown. It is becoming also more and more interesting in real-time rendering of synthetic 3D scenes where the exact depth information is known for free. However, here the goals are slightly different: saving computation costs by either performing 3D computations in 2D image space thereby exploiting the implicit grid structure of the images [RGS09, BS08, KBW06, RSC87] or image-based warping and caching for boosting the frame-rate [YNS*09, SaLY*08a, NSL*07]. The latter concept has been also exploited in Chapter 8.

3.2.2 Point Representation

3D points are the most simple and fundamental geometry-defining entities. Points as display primitives were already considered in [LW85]. They are more general than an image-based pixel representation discussed previously. Based on their fundamental simplicity, points are applicable in a variety of research topics in computer graphics. First of all, since points are the raw output of 3D scanners, they are important in geometry processing, modeling and visualization. Points are a simple and attractive multi-resolution representation for smooth and detailed objects as often found in nature. In terms of rendering the major challenge of point-based rendering (PBR) algorithms is to achieve a continuous interpolation between discrete point samples that are irregularly distributed on smooth surfaces. However, in case of synthetic, artificial objects they become less appealing since sharp features require a higher density of points and hybrid polygon-point representations are preferable. Nevertheless, in rendering they are often useful when exact geometry is not needed, like for example in low-frequency lighting computations [REG*09, RGK*08, LZT*08].

3.2.3 Voxel Representation

A *voxel* (volume element) is the 3D pendant to a pixel in 2D. A voxel represents a value in a regular grid in three dimensional space. In contrast to points and polygons, voxels (and pixels) themselves do not have their position explicitly encoded along with their values. Instead, the position of a voxel is inferred based upon its position relative to other voxels. Voxels are a natural represention of regularly-sampled spaces that are non-homogeneously filled and are frequently used in the visualization and analysis of medical and scientific data.

In real-time rendering the voxel representation is becoming increasingly attractive [CNL*09, ED06, DCB*04] due to the ability of efficient 3D rasterization and real-time volume rendering with well-defined level-of-detail on current GPUs. The downside is the increased memory consumption, which quickly reaches the order of Gigabytes instead of Megabytes as for 2D images (e.g., a voxel grid of resolution 2048^3 requires 32 Gigabytes of memory for 4-Byte-voxels and still 1 Gigabyte for binary voxels). The voxel representation has also been exploited in Chapter 5 as a means of approximate but fast visibility queries of "volumetric rays".

3.2.4 Polygonal 3D Scenes

Polygons are the common choice for representing and approximating surfaces of 3D objects. Often 3D polygonal objects are modeled by artists and engineers using specialized CAD software. The basic object used in polygonal modeling is a *vertex*, a point in three dimensional space. Two vertices connected by a straight line define an *edge* and three vertices, connected to each other by three edges form a *triangle*, which is the simplest polygon in Euclidean space. More complex polygons can be created out of multiple triangles. Four sided polygons, referred to as *quads*, and triangles are the most common shapes used in rendering and modeling. A group of polygons which are connected together by shared vertices is referred to as a *mesh*.

Polygons are well suited for *rasterization* and therefore the method of choice for real-time computer graphics. The main disadvantage is that polygons are incapable of accurately representing curved surfaces, so a large number of them must be used to approximate curves in a visually pleasing manner. Polygonal meshes may also become inefficient when triangles become smaller than a pixel during rasterization. This can be avoided with the help of level-of-detail adaptation but this is generally more difficult for polygonal meshes than for points or voxels and usually, for the same object, several meshes of different complexity are precomputed offline and kept in memory.

3.3 Determining Visibility

Computing the visibility between any two surfaces is the crucial part of any (global) illumination algorithms. Visibility causes discontinuities and is in general not conservatively predictable and therefore requires explicit evaluation via *rasterization* or *raytracing*. Thus, the efficiency of a global illumination algorithm is mainly determined by the number and coherence of its visibility queries.

3.3.1 Rasterization

Rasterization is perhaps the most common technique for generating views of three-dimensional polygonal scenes. Individual polygons are projected into screen-space and for every screen pixel covered by the projected polygon, the visibility to the camera is determined by the z-buffer algorithm such that only those pixels are kept, which are closest to the camera. This discretization of polygons to pixels is called rasterization. Basically, the entire scene, after projection to normalized device coordinates, is sampled at regular intervals (rasterized) forming a uniform grid representing the image. The advantage of rasterization methods is the coherent processing of neighoring pixels, which fits well to parallel processing hardware. It is most efficient if all polygons are visible and if the polygon count is small. On the other hand, the major limitation of rasterization methods is the lack of local adaptation. Usually, the entire scene is rasterized even though only sparse point-to-point visibility is needed.

3.3.2 Raytracing

Raytracing is a more natural and general way of computing synthetic views of three-dimensional scenes. It is not limited to polygonal scenes and, as the name states, better resembles the ray-based nature of light. Therefore, it is more convenient in terms of physical simulation and software implementation since each “light-ray” has atomic nature representing radiance from a single direction and point in space (ignoring volumetric effects), which can be determined by computing the nearest intersection with all surfaces in the 3D scene model. However, simply testing every surface for possible intersection with each ray sequentially is computationally intensive and not well scalable. Therefore, efficient search data structures often referred to as raytracing acceleration structures that adapt to scene complexity and enable sub-linear intersection queries become very important and are still a hot topic in current research. Raytracing is the method of choice when it comes to simulating complex camera lenses and all kind of lighting effects.

This flexibility comes at a high price: the efficiency for computing the surface intersections is strongly depended on the coherence of the traced rays. While in reality “ray-tracing” is inherently parallel, in a computer simulation millions of rays are often traced sequentially benefiting little from the coherence of rays. Therefore, for long, raytracing has been doomed to be an off-line algorithm. In recent years, thanks to the quickly developing parallel computing hardware, raytracing is becoming more and more attractive for interactive and even real-time rendering. However, so far, coherence is mainly explored at the level of "direct rays" cast for example from a pine-hole camera or a point light source, which are easily parallelizable on a multi-processor machine.

3.4 The Rendering Equation

The *rendering equation* introduced by Kajiya [Kaj86] is the basis for all global illumination methods and will appear frequently in the thesis. It forms the basis for computing the light transport in a scene model without participating media. More specifically, it relates outgoing radiance L_o leaving a differential surface-patch to the local incident radiance L_i. Thereby, L_o is interpreted as the sum of self-emitted radiance L_e and the reflected radiance L_r

$$\begin{aligned} L_o(\mathbf{x},\omega_o) &= L_e(\mathbf{x},\omega_o) + L_r(\mathbf{x},\omega_o) \\ &= L_e(\mathbf{x},\omega_o) + \int\limits_{\Omega} f_s(\mathbf{x},\omega_i,\omega_o) L_i(\mathbf{x},\omega_i)(\omega_i \cdot \vec{n}) d\omega_i, \end{aligned} \tag{2.15}$$

where L_i is the incident radiance from a certain direction in the hemisphere, which in general is not explicitly known as it is the outgoing radiance L_o reflected from another surface.Unless only the *direct illumination* from the emitting light sources is computed, we cannot reformulate this integral to a smaller integration domain Ω, since, at least for closed environments, we have to consider all direction ω_i. For *indirect illumination* Eq. 2.15 cannot be directly evaluated and recursively expands since the radiance value is on both sides of the equation. Therefore, it is often more convenient to write Eq. 2.15 in the form

$$L = L_e + L_r = L_e + TL, \tag{2.16}$$

where T is the integral operator in Eq. 2.15, which can be recursively expanded to

$$L = L_e + TL_e + T^2 L_e + \ldots = \sum_{i=0}^{\infty} T^i L_e. \tag{2.17}$$

This expansion is called *Neumann series* and sums light contributions reflecting $0, 1, 2, \ldots$ times.

3.5 Monte Carlo Light Transport

Since analytical solutions are generally not feasible because of the hidden visibility term in Eq. 2.15, we need numerical solutions. A natural way is the numerical quadrature formula (e.g. trapezoidal-rule, Simpson, Gaussian quadrature). However, as one can easily see, the rendering equation is a recursive function because the radiance value is on both sides of the equation and is therefore high-dimensional. And those numerical quadrature techniques become quickly less efficient for higher dimensions as the computational complexity is exponential with regard to the dimension of the integration domain. Therefore, the common solution to avoid such *dimensional explosion* of numerical quadrature formulae is *Monte Carlo sampling* using an appropriate hemispherical *probability density function* (pdf) $p(\vec{\Psi})$ to generate N random directions $\vec{\Psi}_i$. This results in an estimator:

$$L_o(\mathbf{x},\omega_o) = L_e(\mathbf{x},\omega_o) + \frac{1}{N}\sum_{i=1}^{N} \frac{f_s(\mathbf{x},\vec{\Psi}_i,\omega_o) L_i(\mathbf{x},\vec{\Psi}_i)(\vec{\Psi}_i \cdot \vec{n})}{p(\vec{\Psi}_i)}. \tag{2.18}$$

In order to compute this estimator, N directions are generated from the pdf $p(\vec{\Psi})$. The BRDF f_s and cosine term $\cos\theta_i = \vec{\Psi}_i \cdot \vec{n}$ also need to be evaluated. A ray is traced in direction $\vec{\Psi}_i$ where we compute the outgoing radiance in direction $-\vec{\Psi}_i$ from the closest ray intersection point.

3.6 Variance Reduction

Blind Monte Carlo integration with uniform sampling is prone to noise, which vanishes slowly. To improve the convergence of the estimate, we must reduce variance. The basic strategy for variance reduction is to exploit some knowledge about our function we wish to integrate.

3.6.1 Importance Sampling

In terms of *variance reduction* it is crucial to use pdfs that are as close as possible to the incident hemispherical radiance function L_i or even better the BRDF-modulated incident radiance function $f_s \cdot L_i$. Such sampling from non-uniform pdfs is called *importance sampling*. Conceptually, the idea of importance sampling is to evaluate the integral denser in regions of the domain where the function we wish to integrate is large. Ideally, the sampling density should be proportional to the function itself, i.e., $p \sim f_s L_i \cos\theta_i$. Therefore, importance sampling of the indirect illumination is a very difficult problem as we do not know the incident lighting we wish to integrate. The common assumption is that incident radiance is relatively uniform. This reduces the problem to a local computation: BRDF importance sampling – the adaptation to the local BRDF, which can be sampled optimally for many analytical BRDF models. Much research has been devoted to the topic of importance sampling and we will not go into detail about advanced Monte Carlo techniques here but refer the interested reader to profound literature: [BGH05, DBB03, KW00, Vea97, SWZ96, KW86].

It would be inefficient to compute the full reflected radiance via blind (i.e., uniform) *Monte Carlo ray tracing*, since the incident radiance field as well as the BRDF contains high frequencies, which results in stochastic noise. Therefore, equation Eq. 2.15 is commonly split into a sum of disjoint light contributions for different BRDF components that are solved individually and summed at the end. This is possible because light is additive and most BRDF models are separable into diffuse and glossy parts.

One important contribution to the reflected radiance is the direct illumination from visible light sources. Since only a few directions in the estimator Eq. 2.18 result in a nonzero contribution to the direct light, L_r is split into direct and indirect light from which only the latter one is computed by the integral over hemispherical directions. However, for the direct light we better transform the integral in equation Eq. 2.15 to an integral over surface locations by replacing the differential solid angle

$$d\omega_i(\mathbf{x}_i) = \frac{(\omega_i \cdot \vec{n}_i)dA_i}{\|\mathbf{x} - \mathbf{x}_i\|^2}, \tag{2.19}$$

where the index i denotes another surface with differential area dA_i, normal $\vec{n}_i$, and position $\mathbf{x}_i$. Remember the solid angle is the projection of the surface area onto the unit hemisphere. This leads to the rendering equation computed as the integral over all surface points

$$L_o(\mathbf{x}, \omega_o) = L_e(\mathbf{x}, \omega_o) + \int_S f_s(\mathbf{x}, \mathbf{x}_i \to \mathbf{x}, \omega_o) L_o(\mathbf{x}_i \to \mathbf{x}) V(\mathbf{x}, \mathbf{x}_i) G(\mathbf{x}, \mathbf{x}_i) dA_i, \tag{2.20}$$

where $V(\mathbf{x}, \mathbf{x}_i)$ is the visibility function

$$V(\mathbf{x}, \mathbf{x}_i) = \begin{cases} 1, & \mathbf{x}_i \text{ and } \mathbf{x} \text{ are mutually visible} \\ 0, & \text{otherwise,} \end{cases} \tag{2.21}$$

$G(\mathbf{x}, \mathbf{x}_i)$ is called the geometry term comprising distance and orientation of two differential surfaces

$$G(\mathbf{x}, \mathbf{x}_i) = \frac{-(\omega_i \cdot \vec{n}_i) \cdot (\omega_i \cdot \vec{n})}{\|\mathbf{x} - \mathbf{x}_i\|^2}, \tag{2.22}$$

and S is the set of all surface points from which we want to compute the radiance contribution to point $\mathbf{x}$. In case of the direct light computation equation 2.20 is more appropriate since the area of all light sources is usually relatively small and covers only a small solid angle on the hemisphere above $\mathbf{x}$.

3.6.2 Random Number Generation

Besides designing a "good" pdf for importance sampling, the rate of convergence of Eq. 2.18 depends also on the properties of the multi-dimensional uniform random-number sequence. The goal is to reduce the *discrepancy* of the sampling sequence. In other words, we want to avoid the clumping of random samples, which is particularly problematic for higher dimensions. The two main techniques that achieve this are: *stratified sampling* and *Quasi-Monte Carlo sampling*. Stratified sampling divides the domain into M disjoint sub-domains called strata each evaluating the integral separately with one or more uniformly distributed samples. Quasi-Monte Carlo techniques replace randomness entirely by deterministic sequences called *low-discrepancy sequences* (LDS) in order to minimize the *discrepancy* on the expense of introducing correlation between samples. Many LDS based on Quasi-Monte Carlo techniques have been developed. Popular ones are *Halton*, *Hammersley*, *Niederreiter*, and *Sobol*. The results converge as fast as for stratified sampling without the dependence on the total number of samples. Furthermore, we inherently get stratification over several dimensions. On the other hand, Quasi-Monte Carlo sampling can lead to visible artifacts due to deterministic sampling patterns. Therefore, a combination of randomness and LDS, which is called *randomized Quasi-Monte Carlo* (RQMC) is sometimes preferable. More details about the Quasi-Monte Carlo method can be found in [Nie92, Kel96].

4 Human Visual Perception

Correctly simulating every lighting effect within some physically imposed error bounds is computationally demanding. However, many effects are not even perceivable because our human visual system (HVS) is also limited and adapted to "important" visual signals as a result of millions of years of evolution. For example the human eye is very limited with respect to color vision and much more sensitive to the light intensity, or even more to contrast of a lighting signal. Hence, what is important is the *perceived* error of a lighting simulation not the physical error, which can be very different. Understanding the HVS and deriving a perceptual error metric is quite complex and a research topic on its own. It depends on many parameters such as adaption states of the eye, background luminance, color, the signal's velocity across image plane, and so on. Although the early stages of the HVS – from retina to primary visual cortex – have been relatively well understood, the cognitive aspects, the "post-processing of the transmitted signals" in the brain, are still hardly comprehend.

We will only introduce the visual aspects of the human perception here in order to understand the relation between radiometry, described previously, and photometry, which we will need later in this thesis.

4.1 Color

The notion of color has rather a perceptual than physical meaning since it is strongly related to the limited human visual perception. Colors like "red" and "green" do not have a unique relation in the physical world. The physical light consists of a continuous spectrum of wavelengths and is described by the spectral power distribution $\phi(\lambda)$. On the other hand, the human color perception is determined by three types of cones in the fovea of the eye: L, M, and S and their sensitivity to a particular wavelength in this power spectrum. Hence, mathematically speaking the perceived color is just a projection of an infinite-dimensional continuous power-spectrum to the spectral sensitivity functions $\mathcal{C}$ of the three types of cones in our eye

$$R = \int_\lambda \phi(\lambda)\mathcal{C}_L(\lambda)d\lambda, \quad G = \int_\lambda \phi(\lambda)\mathcal{C}_M(\lambda)d\lambda, \quad B = \int_\lambda \phi(\lambda)\mathcal{C}_S(\lambda)d\lambda. \tag{2.23}$$

Therefore, different spectral power distributions can have the same apparent color response to a human observer. The spectral response of the standard human observer to various wavelengths has been acquired by the CIE in as early as 1931. Since the spectral response of the cones was not known at this time the CIE initiated a color matching experiment in which human observers had to match a particular wavelength with a linear mixture of three monochromatic primary colors (435.6 nm, 546.1 nm, 700 nm) representing blue, green, and red, respectively. The resulting matching curves are shown in Fig. 2.4(left). The negative part for the red primary is due to the fact that the human subjects were allowed to add monochromatic primary light to the reference color to be matched (as we can not have negative light) whenever the reference color could not be reproduced with the given three primaries.

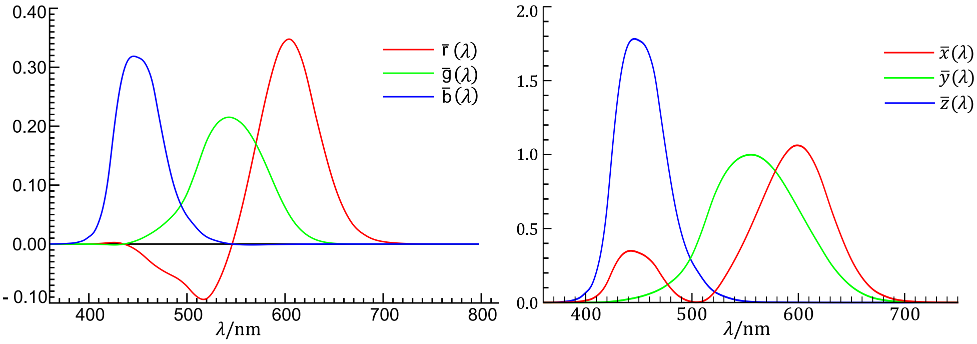


Fig. 2.4 – CIE RGB and XYZ color matching functions

(left) CIE color matching functions for the stimuli red (R), green (G), blue (B) of a standard observer http://www.cvrl.org*, (right) CIE XYZ color matching functions. Note that the Y primary (green) was chosen to represent luminance.*

New CIE matching functions, $\bar{x}(\lambda)$,$\bar{y}(\lambda)$,$\bar{z}(\lambda)$, were derived from these RGB functions in such a way that all function values are positive in the visible color spectrum (i.e. the whole visible color gamut is covered) and $\bar{y}(\lambda)$ corresponds to the luminous efficiency function, see Fig. 2.1. The new set of primaries, referred to as XYZ, is shown on the right of Fig. 2.4. Although very old, most applications still use the CIE XYZ standards as reference.

A plot of all visible colors is three-dimensional. However, for convenience we can separate the visible colors into *chromaticity* and *luminance* (brightness). Thus, the XYZ *tristimulus* values are normalized to obtain 2D chromaticity coordinates $(x,y) = (X,Y)/(X+Y+Z)$, $(z = 1-x-y)$ and luminance Y. By using the x and y chromaticity coordinates, this derived color space, known as the CIE xyY color space, better resembles our visual system. But Euclidean distances between pairs of points in this space do not match the expected perceived differences. Therefore, better color spaces have been derived from the XYZ color space, which achieve visual uniformity and also incorporate the luminance dependence of perceived color differences (e.g. CIE 1976 $L^*a^*b^*$, CIE 1976 $L^*u^*v^*$). Furthermore, it is important to note that color information in natural images is highly correlated. Therefore, efficient color spaces (e.g. $L^*a^*b^*$) use opponent color-pairs, i.e. color is encoded as differences: red-green and blue-yellow, which is also the way the visual signal is encoded in the optical nerves leaving the eye-ball.

Note that although representing color with just three primaries is perceptually justified as it measures how light interacts with our eye, it may be insufficient for a global illumination rendering system where light interacts with surface materials in a complex way before it is finally observed by the simplified human color vision. Only the directly visible output of such a simulation can actually be processed in a perceptual tristimulus color space without significant distortion [WEV02]. Nevertheless, due to limited computational resources most computer graphics application like the one's presented in this thesis rely on simple tristimulus values.

4.2 Luminance Sensitivity

Luminance is the quantity that our visual system is most sensitive to. Photoreceptors in the human eye convert light energy to neural signals in a non-linear way. The response depends on the current luminance adaptation state. Therefore, the eye is more sensitive to relative luminance levels with respect to the background than absolute luminance. This means that an absolute luminance threshold becomes less visible for high luminance conditions. This phenomenon is referred to as *luminance masking*. The photo-receptor's response to luminance is often modelled as an S-shaped curve in a log-linear plot (see Fig. 2.5) where the middle part of the curve ($10-500\ cd/m^2$) is most interesting to us as current CRT and LCD displays fall into this range of brightness.

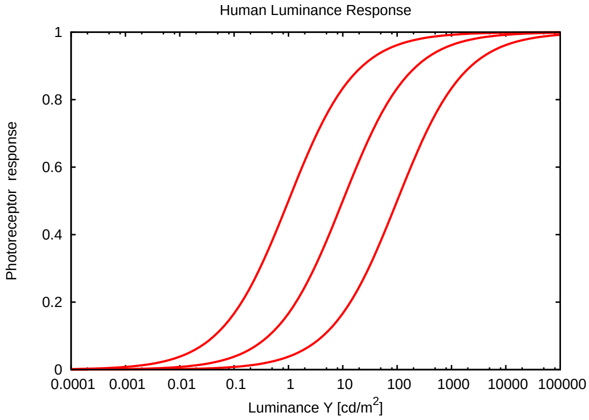


Fig. 2.5 – Luminance sensitivity function

Humans photoreceptor response curves (shown for 3 different adaptation levels) according to the Michaelis-Menten equation.

This range (photopic vision) can be modeled by a logarithmic function and the Weber-Fechner law, which states that the contrast detection threshold is roughly 1% of the luminance adaptation level (Weber fraction), can be applied[1]. However, in the low to medium luminance range (mesopic vision) of modern LCDs a power function model is a better choice. Note that the non-linearity of most

[1] The Weber-Fechner law is a coarse simplification and does not match well realistic measurements of the human vision for small luminance values, $Y < 10cd/m^2$. However, most computer graphics applications limited to a small range of luminance it can be regarded as a conservative estimate and for $Y > 100cd/m^2$ even matches almost perfectly the visual response.

displays equals a power function which is nearly inverse to the human's luminance perception (see Section 4.2.1), which causes the display's response to be roughly perceptual uniform.

4.2.1 Relation to Gamma Correction

The visual output of a display device is not a linear function of the applied signal. Instead, a device has a power-law response to voltage: the luminance produced at the face of the display Y is approximately proportional to the applied voltage V_{in} shifted by the black-level offset ε raised to a power known as gamma γ

$$Y \approx (V_{in} + \varepsilon)^{\gamma}. \tag{2.24}$$

For CRT displays a common gamma value is 2.2 while for LCDs the gamma value can deviate slightly. In order to linearize the luminance of the output, we must consider this gamma function when storing our colors. Such non-linearity must also be taken care of when using image textures for rendering where we assume linear data because the (RGB) values of a texture are commonly stored as a gamma converted version (e.g. sRGB color space). Interestingly, the reason for this is widely miss-understood: It is not intended to "correct" the gamma of a display device, which in fact is in favor of our human's luminance response (see Section 4.2), but to better utilize the number of bits during image compression [Poy98] (e.g. JPEG, PNG). Ideally, the images need to be perceptually uniform before compression[2].

4.3 Spatial and Temporal Contrast Sensitivity

The sensitivity of the human visual system (HVS) to contrast is described by the *contrast sensitivity function* (CSF) as shown in Fig. 2.6.

The sensitivity to contrast varies with spatial and temporal frequency, orientation, wavelength, and luminance. Interestingly, this contrast sensitivity function shown in Fig. 2.6 is in favor with Monte Carlo rendering algorithms, where the visual error manifests in form of high-frequency noise. Such highest frequency image-content is what the human eye is least sensitive to. This effect becomes even stronger in the presence of other high-frequency "masking" signals, e.g. textures, overlaying the generated noise, which is described in Section 4.3.1. Therefore, in terms of rendering it is usually advisable to turn (low-frequency) errors into high-frequency noise, which is also less objectionable.

[2] If a low dynamic range RGB image is linearized, approximately 11 bits are necessary to achieve high-quality. With non-linear (gamma-corrected) coding only 8 bits are sufficient [Poy98].

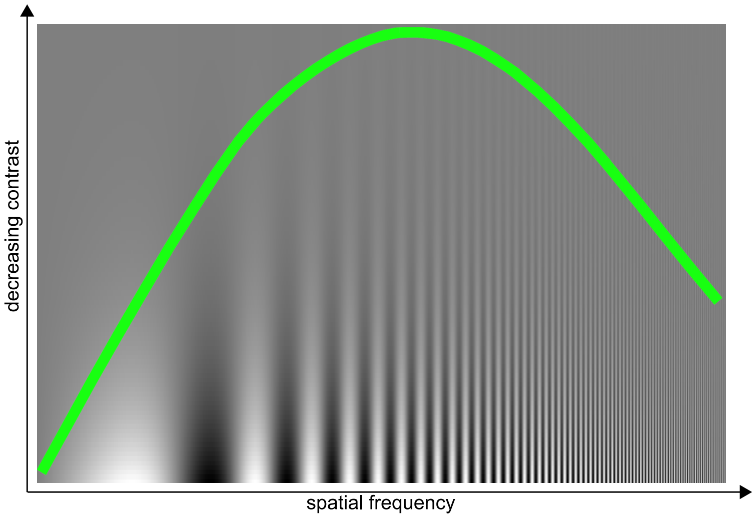


Fig. 2.6 – Contrast sensitivity function

Contrast sensitivity function (green). Depending on the luminance level, the contrast amplitudes above the green curve are invisible to a human observer.

4.3.1 Contrast Masking

Natural images consist of complex signals with many frequencies with different phase and orientations. It is well understood that a signal is less perceptible in the presence of other signals with similar frequency and orientation [FPSG97, Dal93]. For an example see Fig. 2.7. This phenomenon is called *contrast masking* or *visual masking* since one signal masks the other. Contrast masking is one of the strongest visual effects that can be exploited in rendering. The effect is strongest if both, masking signal and masked signal, have same orientation, color and frequency. The detection threshold of a signal increases non-linearly with the contrast of the masker signal.

Considering above observations one can deduce that errors in computer generated images become less perceptible the complexer the image content. Consequently, the effect of contrast masking plays an important role and has been widely employed in various fields of rendering [WPG02, MTAS01, RPG99, BM98, FPSG97] mainly for reducing per pixel computation costs where image content is dominated by high-frequency textures and geometry. The effect of contrast sensitivity and visual masking has also been exploited in this work, see Chapter 7, Section 4 and in Chapter 6, Section 6.5.

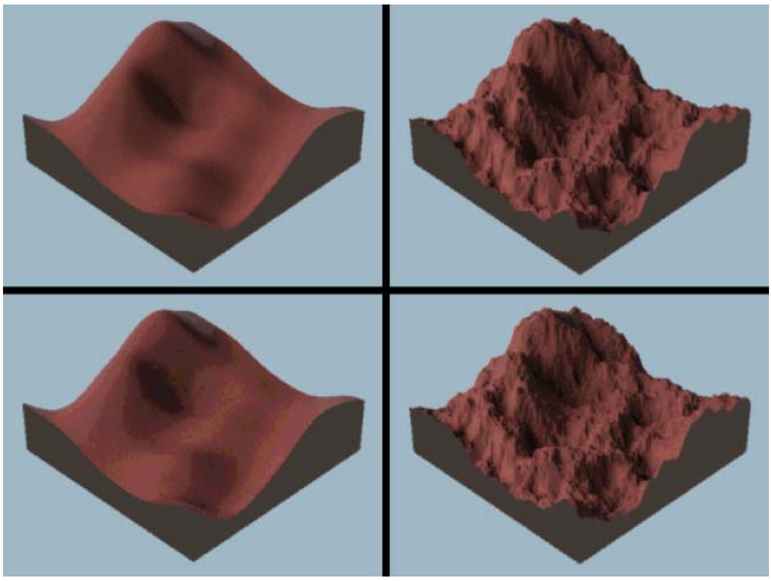


Fig. 2.7 – Example of visual masking effects in computer graphics *[image courtesy of M. Bolin and G. Meyer 1998]*

Visual masking (contrast masking) in computer graphics [BM98]: The upper pair of images is quantized to 8 bits whereas the lower pair is quantized to 4 bits. Banding is clearly visible in the smooth surface for the 4 bit image on the left but not for the rough surface in the right image, where the 4 bit quantization errors are masked by the high-frequency base content.

4.4 Tone Reproduction

A computer-based lighting simulation yields physical radiance values possibly even spectral radiance values. Such radiance values can be of arbitrary value greater than zero. While we, as humans, can see 14 orders of magnitude of luminance range (in cd/m^2) and simultaneously up to five orders of magnitude, most current display devices and image file formats have a very limited range (usually 2–3 orders of magnitude, i.e., 8–10 bits) of representing luminance values. The luminance range of standard displays is referred to as low-dynamic range (LDR), while higher ranges close to the human vision are classified as high-dynamic range (HDR) (however with a rather fuzzy transition). Hence, the physically-based rendering output needs to be prepared for LDR display and storage. This process is called *tonemapping* and is generally lossy as real pixel values are clamped and quantized to 8 bits. This way, too small values are clamped to zero (black), the lowest displayable shade, and too large values are clamped to white, the brightest displayable shade. Since our human visual system perceives color in a non-linear way, it is a bad approach to linearly scale and clamp the HDR values. Therefore, tonemapping tries to compress HDR images in such a way that the compression to LDR is perceptually linearized. Much research has been devoted to the topic of tonemapping HDR images [RWPD05], which we silently ignore at this point. A simple, yet effective tonemapping operator, is to use a gamma curve (see Section 4.2.1), which compresses higher values more aggressively while preserving smaller values. Effectively all display and camera devices perform such a gamma mapping implicit or explicitly. For almost all HDR images in this thesis we use a simple gamma based tonemapping operator.

Related Work

In this chapter we briefly summarize and discuss the most related work, which is split into different sections covering approximate global illumination, irradiance and radiance caching techniques, perceptual rendering, and real-time spatial upsampling. And at the end we highlight a few most recent publications that are based on the work in this thesis.

1 Approximate Global Illumination

This section summarizes the most relevant work in the area of global illumination. We will not discuss finite-element radiosity-based methods and precomputed radiance transfer (PRT) since these are not in the scope of this dissertation.

1.1 Photon Density Estimation

An efficient way for computing the approximate illumination at any surface point in the 3D scene model is *photon density estimation* [Jen01], where photons correspond to a sample from the radiant energy distribution in equilibrium state. Although well-studied in the field of radiometry, the concept of photon density estimation has first been brought to computer graphics by Henrik Wann Jensen in 1996 [Jen96]. Jensen proposed a two-pass method where photons are first emitted from the light sources and their hit-points with the scene's surfaces are stored in the *photon map* search data-structure giving the method its name: *photon mapping*. In the second pass the photon map can be queried to estimate the photon density, and hence the irradiance, at any surface point in the scene model.

While the first pass of photon mapping is unbiased, the second pass is not since a finite neighborhood is needed to collect a sufficient number of photons for reducing the noise in the density estimation [Sil85, Sch03]. Conceptually, photon density estimation can only be made *consistent* by increasing the photon count to infinity and at the same time letting the density estimation radius go to zero.

This way photon mapping can converge towards the correct solution, which has been shown empirically in *progressive photon mapping* [HOJ08]. Unlike statistical 2D density estimation, photon density estimation also suffers from geometrically incurred bias as the density estimation domain is generally three-dimensional and bounded [HBHS05, Sch03]. Lastra et al. [LURM02] and Havran et al. [HBHS05] replaced the nearest-neighbor search for photon hit-points on the scene surfaces with a nearest-neighbor search for photon rays traveling in the proximity of these surfaces and this way could avoid bias introduced at the boundary of surfaces.

Various statistical methods [Sil85, ST87, SBS94, WJ95] can directly be applied to reduce the bias and noise in the estimate of the 2D point density corresponding to the photon's hit-point records. The two most prominent approaches are: *k-nearest neighbor* (k-NN) density estimation and variable *kernel density estimation* [Sil85], in the rendering community often referred to as *gathering* and *splatting*, respectively. Whereas in statistics variable kernel-density estimation is known to be superior to k-NN density estimation [Sil85], in rendering the latter is still often preferred for its property of being able to directly evaluate the photon density at arbitrary points in any order in the scene, which is needed for view-dependent random queries as in Monte Carlo ray tracing [Jen01].

Nevertheless, photon mapping with direct photon splatting onto the image plane has already been investigated. Lavignotte et al. [LP03] 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 the 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.

A strong advantage of photon splatting is that we do not need to store the photons but directly splat their energy successively onto the image plane, which has been exploited in progressive photon mapping [HOJ08], where the memory complexity is constant with respect to the photon count allowing to use billions of photons.

Havran et al. [HHS05] first introduced "indirect" photon splatting for final gathering similar to the photon mapping algorithm, which they called *reverse photon mapping*. They showed that splatting is algorithmically superior than gathering when the number of pixel samples (i.e., number of final gather rays times pixels) is larger than the number of photons. They proposed a three-pass algorithm where in several iterations first, all final gather rays are traced and their hit-points with scene surfaces are stored in a search data structure. In the second pass, the photons are sampled as in traditional photon mapping [Jen01] possibly guided through importance sampling using the initially stored final-gather rays. In the final pass, all photons indirectly splat their energy to the pixels corresponding to final gather rays.

Another interesting photon splatting approach has been subject in the work by Bekaert et al. [BPC*03]. They aim at high-quality rendering of various kind of surface material and illumination conditions using photon density estimation. They show that, at least theoretically, photon density estimation can be made unbiased. 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 [DLW93] and bidirectional path tracing [SW00], but there the main motivation was noise reduction. Suykens and Willems [SW00] 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 [DS06] 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 transport lighting energy from reflections, refractions, and caustics [SKALP05, SKP06, WD06]. In all those solutions the main goal is maximizing the rendering speed at the expense of reduced image quality.

1.2 Instant Global Illumination

Instant radiosity also referred to as *instant global illumination* (IGI) is another very popular two-pass method introduced by Keller [Kel97] in 1997, which is related to bidirectional path tracing [LW93] and photon mapping [Jen96]. Similar to photon mapping, IGI caches and reuses lighting on scene surfaces to suppress noise at the expense of bias. The main difference to photon density estimation is the view-dependent gathering pass. The idea is simple though effective. Instead of directly computing the irradiance from the local photon density, the indirect diffuse illumination is computed by direct illumination treating each photons as an oriented virtual point light source (VPL). Hence, IGI integrates the incident radiance over the whole scene area while explicitly evaluating the geometric term Eq. 2.22 and visibility in the rendering equation. Therefore, it accurately reproduces high-frequency lighting features like sharp shadow boundaries.

On the other hand, its brute-force nature allows only to use a small set of VPLs, which is insufficient for rendering glossy light transport such as caustics and indirect illumination on glossy surfaces. Therefore, a “good” distribution (low discrepancy) of the VPLs is very important, which can be achieved with Quasi-Monte Carlo sampling [Kel96]. IGI also suffers from singularities near corners due to the squared distance term in the denominator of the geometric term, which can be compensated for when mixing it with solid-angle based Monte Carlo integration [KK04]. It is therefore only well-suited for distant illumination from view-independent diffuse surfaces and near-field illumination needs to be clamped to suppress noise.

The main advantage of IGI is that it can be efficiently implemented on graphics hardware for less complex scenes using extended shadow maps [RGK*08, DS05].

Few extensions to IGI have been developed to overcome the limitation to diffuse light transport [HKWB09, WFA*05]. Further, most approaches treat each VPL the same during final gathering although VPLs can have very different energy contributions depending on their distance and orientation. *Lightcuts* [WFA*05] is an algorithm which adaptively chooses a small subset from a large number of VPLs depending on their relative location and orientation in the scene, which we will further describe in Chapter 6.

1.3 Voxelization and Visibility Approximation

Computing the indirect visibility is the main bottleneck of global illumination algorithms. Therefore, opting for efficient and perceptually plausible approximations is very desirable. However, this is a very broad topic and we will only regard a few related methods here mainly based on solid voxelization.

Using a voxel representation of a polygonal scene is not new but has regained attention due to powerful, programmable graphics hardware with integer arithmetic. Solid voxelization is becoming increasingly attractive in real-time rendering. Recently, even giga-voxel grids can be processed at real-time or interactive rates [CNL*09]. In [HW02] the authors present a robust method to generate an octree hierarchy of a solid voxelization of arbitrary polygonal scenes, which is still created in a slow offline process. Then, in [DCB*04, ED06] it was shown how to efficiently exploit the rasterization pipeline of graphics hardware to generate a voxelized grid of a polygonal scene in real-time. This allowed for complex effects in real-time applications, such as the computation of translucency, constructive solid geometry (CSG) operations, particle collision detection, etc. (see [ED08] for example). The basic principle is that surfaces of the scene are rasterized to binary grid slices in 3D in a few render passes [DCB*04] or even in a single pass on newer hardware [ED06, ED08]. In Chapter 5 an approach based on the work by [DCB*04, ED06] is used although targeted on efficient hierarchical data structures with further processing on the CPU.

Perhaps one of the first who has approximated the visibility in global illumination computations using a voxel-based representation is R. Malgouyres, who used it in a discrete radiosity solver [Mal02].

An alternative for voxelization of sparse scenes is *depth peeling* [Eve01], where the scene surfaces are "peeled" to form successive visibility layers with respect to a certain view point. In each rasterization pass the visible surfaces of the previous pass are "ignored" (by offseting the depth buffer), revealing the surfaces behind, which appear in the next slice. Depth peeling has the advantage that each slice has full depth precision and can store more information (e.g., normals) but requires arbitrarily many passes depending on scene and view point. It has mainly been used for indirect visibility computations performed on the GPU [Hac05, SKP98].

A similar concept are *layered depth images* [SGwHS98], which extend the standard z-Buffer to store many depth values per pixel, which are constructed using ray tracing or image warping in a preprocess. They allow for fast image based rendering using efficient splatting methods.

Besides regular grid-like data structures, other 3D scene representations have been successfully used in global illumination computations. Ritschel et al. [RGK*08] employed a point-representation of a polygonal scene for cheap visibility evaluation for the indirect illumination computed with instant radiosity [Kel97]. They only construct very coarse shadow maps, referred to as *imperfect shadow maps*, but for thousands of indirect light sources (VPLs) in real-time.

Deep shadow maps [LV00] store for each sample ray a one-dimensional visibility function accounting for the light intensity changes along the ray. This additional information allows not only to render volumetric effects but also to have better shadow filtering especially for fine geometry such as hair.

2 Exploiting Spatial/Temporal Coherence

2.1 Radiance Caching

Caching lighting information is the fundamental idea behind all efficient global illumination algorithms. Indirect lighting, in particular diffuse indirect lighting, is known to be very spatially coherent and computing it densely at every pixel sample is very costly. Therefore, it pays off to sample the indirect lighting sparsely and interpolate in-between values. Such scheme known as *irradiance caching* was introduced to computer graphics by [WRC88]. Irradiance caching makes global illumination practical and has been widely used ever since. In [WH92] the authors improved the cache interpolation by using irradiance gradients. The original irradiance cache sampling assumes purely diffuse environments when determining an upper bound for the cache sampling density. Smyk and Myszkowski [SM02] also exploits the irradiance gradients to increase the cache density where the actual gradient magnitude is larger. Tabellion et al. [TL04] propose several small practical modifications of the original irradiance caching scheme [WRC88]. Křivánek et al. [KGPB05] generalized irradiance caching to radiance caching on moderately glossy surfaces. Essentially, the major difference to irradiance caching is that the full incident radiance field is captured and interpolated in the spherical or hemispherical harmonics basis [GKPB04] rather than just the view-independent irradiance. Since radiance caching is more sensitive to interpolation errors, the authors also propose an adaptive caching scheme [KBPv06] where in multiple iterations visible errors in the cache interpolation are removed by increasing the cache density where possible discontinuities are detected.

The reconstruction quality of all those caching algorithms strongly depends on the pixel traversal order. In [GKBP05] a hybrid GPU-CPU method was presented for the widely used (ir)radiance caching algorithm that overcomes this limitation. The approach reverses the caching procedure by splatting cache samples to the image plane. First, each cache sample is computed on the GPU by rasterizing directly illuminated scene geometry (similar to [LC04]). Then the cache sample is splatted to the image by rasterizing a bounding quad in the image plane that encloses all pixels in the footprint of the projected maximum sphere of influence of the cache sample. And the process is repeated until all pixels are computed. However, the method relies solely on screen space projection of cache samples and cannot naturally support splatting of indirect cache samples due to secondary rays resulting from specular light transport towards the camera.

All discussed (ir)radiance caching techniques use Monte Carlo integration and in particular the photon mapping algorithm [Jen01] to compute the incident radiance field or irradiance at cache samples in object space. Alternative caching and interpolation in image space is more sensitive to discontinuities (e.g., object silhouettes or high-frequencies in the BRDF and textures) since it captures only outgoing pixel-radiance modulated by the local surface BRDF. Therefore, image-space radiance-caching techniques need to sample more densely to avoid visible artifacts. Some prominent examples of image-space caching techniques are: *Render cache* [WDP99], *Frameless rendering* [DWWL05, BFMZ94], which also cache radiance in time domain, or *Reconstruction cuts* [WFA*05], an adaptive pixel-radiance caching scheme based on the popular *Lightcuts* algorithm [WFA*05, WABG06].

Another class of light caching techniques stores the lighting in a volume rather than on surfaces. The *irradiance volume* [GSHG98] caches the irradiance distribution in a regular three-dimensional grid, where in-between values are interpolated from the nearest grid cells. Later on, this technique was adapted to real-time rendering [Oat05] on the GPU using cube map rasterization and efficient low-frequency basis function (e.g., spherical harmonics [Mac48]) to capture the radiance field dynamically in a sparse grid.

Another caching paradigm for exploiting spatial coherence in the lighting is the concept of *interleaved sampling* [SIMP06, WKB*02]. Because interleaved sampling is mainly designed for parallel computation on graphics hardware, it uses a regular sampling pattern that decorrelates the spatial lighting or shading information. In contrast to the radiance caching techniques described above, for interleaved sampling the shading function is evaluated at every pixel, however very coarsely.More specifically, all pixels with indices belonging to the same residue class (i.e., having the same offset) use the same subset of random samples to evaluate the lighting integral. This results in structured noise, which needs to be low-pass filtered. Hence, the assumption behind interleaved sampling is that the signal is of low-frequency.

2.2 Real-time Upsampling on the GPU

There are several ways to reduce the pixel workload of the shader pipeline on the GPU. It is possible to reduce the amount of shaded pixels, using techniques such as early-z/deferred shading, visibility tests [KCCo00], or shader culling units [HAM07]. In the following we will focus on the coherence between pixel values over space and time. In some sense, irradiance caching, discussed previously, is a specialized upsampling technique, however an adaptive one operating in 3D world space. The discussed techniques below rely on mainly regular, sparse sampling of pixels in a 2D image.

2.2.1 Temporal Caching

Temporal methods accelerate rendering by updating only a certain amount of pixels per frame [BFMZ94], but are susceptible to artifacts arising from changes in view or geometry. An improvement can be achieved by adaptively sampling and reprojecting the frame content [DWWL05], but this is most efficient for raytracing. In such a context, some solutions suggested 3D warping to accelerate display and global illumination computations [AH95, MMB97, WDP99] (we refer to [SaLY*08a] for a more complete list of references). These sample-based reprojections can lead to significant fluctuations of the sample density in the derived frames. Better quality is obtained by applying per object 4D radiance interpolants with guaranteed error bounds [BDT99]. The reduced sampling coherence, or the involvement of ray tracing (although steps in this direction exist) make such solutions less GPU-adapted.

Reprojection caches [NSL*07, SJW07, SaLY*08a] are more GPU-friendly. Here, supplementary buffers encode fragment movements. In contrast to image analysis, for geometry, it is relatively cheap to obtain displacement information by evaluation in the vertex shader and storage in the framebuffer. Given a current fragment, one can easily verify its presence in the previous frame and, if possible, reuse its value. This recovered information is not necessarily a final color, but can be an intermediate shader result, the so-called *payload*. The underlying assumption is that, for a fixed surface point, such value should remain constant over time. Such an assumption gave rise to the idea of integrating samples over time for static scene antialiasing [NSL*07], where static also excludes illumination variations and changes introduce significant artifacts.

The work of Yang et al. [YNS*09] extends this idea and proposes an adaptive method for antialiasing with reprojection caching with a profound error analysis. They adapt the exponential smoothing factor that determines the decay of previously computed samples over time to temporal changes in shading and to blur due to the bilinearly filtered reprojection cache. Further, they temporally interleave frames to virtually increase the spatial resolution. They target antialiasing and, thus, use only 2×2 sub-pixel buffers which is sufficient for their purposes. Since their method is mainly designed for antialiasing of procedural shaders, it cannot deal with dynamic object silhouettes and also needs an additional hole-filling per-pixel rendering step.

To detect almost constant shader components, one can use a learning stage with particular objects [SaLY*08b]. This requires a long preprocessing time and such a setup cannot exploit coherence that might arise from the application itself, e.g., if the shader is light-position dependent, but the light is stationary during the execution.

2.2.2 Spatial Upsampling

Upsampling is the process of increasing the sampling rate of a signal. Hence, upsampling raster images means increasing the resolution of the image.

The underlying assumption is that many signals in synthetic images are spatially slowly varying and can be reconstructed by sparse sampling followed by interpolation. This is true for many low-frequency shaders, e.g., lighting computation, which has een exploited by Yang et al. [YSL08], which deal with dynamic changes and reduce the shading workload by producing low resolution frames that are upsampled to produce a complete frame. The authors apply a joint-bilateral filter [ED04, PSA*04] to perform image upsampling [KCLU07] where the filter weights are steered by geometric similarity encoded in a high-resolution *geometry buffer*. This means that samples which are close in world space and have similar surface orientation are better interpolation candidates.

The downside of such a solution is that high-frequency detail might not be captured in the low resolution frame, and hence it is not always possible to deduce the information needed for the current frame. Super-Resolution techniques, e.g., [CM98], deal with this problem and attempt to add new frequencies recovered through de-interlacing, image content analysis, or edge preserving interpolation.

2.2.3 Spatio-Temporal Processing

Spatio-temporal filtering exploits temporal coherence by involving samples, possibly motion-compensated, from previous frames. Such filtering is commonly used in video restoration [Tek95, BM05], and has been successful in suppressing aliasing artifacts in ray tracing [Shi93].

Spatio-temporal filtering is a powerful tool for removing noise in Monte Carlo image synthesis, in particular when using an edge-preserving bilateral filter [KCLU07, ED04, PSA*04]. Such filter has been successfully applied to photon density estimation [WMM*04a] where the spatio-temporal filter support is adapted to space-time changes. For example in a static scenario the spatial support is reduced while the temporal support is extended favoring convergence.

Temporal coherence has been used to greatest advantage in a number of global illumination solutions discussed in the survey paper by Damez et al. [DDM03]. Many of the presented techniques are off-line or even require knowledge of subsequent keyframes which is unacceptable for interactive rendering purposes. Other approaches exploit temporal coherence at a very low level, e.g., single photon paths. Low-level coherence usually gives more flexibility and enables the exchange of information between many frames at once. However, it is difficult to be efficiently exploited on current GPUs. Therefore, more GPU-compatible strategies relate to interleaved sampling [WKB*02, SIMP06] which has roughly similar goals in its CPU and CPU/GPU incarnation. To our knowledge, spatio-temporal upsampling has not been addressed in real-time rendering on the GPU.

3 Perceptual Rendering

Adapting the rendering quality to the human visual perception, is a different track for saving costly shading computations. Its goal is to speed up the process by taking into account the limits of the human visual system. It is based on the fact that certain numerical errors in an image, are less perceivable by our visual system than others (see Chapter 1 Section 4). Visual models have been developed in order to compute the perceived error, which depends on many factors such as spatial frequency, orientation, background intensity, color, time. A common approach to determine whether an approximation is indistinguishable from a reference image is to use a visual differences predictor (VDP) [Dal93].

Similar perceptually-based error metrics have been utilized to steer the costly global illumination computation [BM98, RPG99]. In all these techniques the goal was to continue the computation until rendering inaccuracies do not affect anymore the visual quality of images. Advanced visual models have also been used to efficiently compute animation sequences [MTAS01, YPG01].

To predict the visual error thresholds on textured surfaces for rendering, a visual model based on the discrete cosine transform (DCT) has been proposed by Walter et al. [WPG02]. They successfully used it for off-line estimation of visual masking by textures. In their approach, both the scene lighting and geometry are not considered and thus do not contribute to the masking prediction, which may result in too conservative (precise) rendering.

Bolin and Meyer [BM95] developed a ray tracer, which projects pixel samples directly onto the DCT basis for an 8×8 pixel block. The technique was developed for static images. As a result of such rendering a JPEG-like image representation is directly obtained. Since samples are generated sparsely for each block, a costly least squares procedure is required to approximate all DCT coefficients that best interpolate the sampled data. When samples are progressively added additional frequency terms in the DCT block representation are also added.

4 Successive and Active Future Work

Our methods described in this thesis has been adopted in different fields of rendering in particular in the area of real-time and interactive global illumination. In this section we will list a few known subsequent publications that are closely related to our work.

McGuire and Luebke [ML09] have developed an image-space algorithm of our photon ray splatting described in Chapter 4 called *Image Space Photon Mapping* (ISPM) that rasterizes a light-space bounce map of emitted photons on the GPU. Their algorithm runs in real-time for about 20000 photons and HD resolution (upsampled) on current high-performance GPUs.

In [BGB08] a method was proposed that combines irradiance caching with photon mapping for interactive walk-throughs in a similar way we did for our photon ray splatting except that they refine the cache in a third “final-gather” pass.

Fabianowski and Dingliana [FD09] combined our photon bandwidth-selection proposed in Section 5 with the photon’s path differentials [SW01]. Besides, they also used a splatting approach for the density estimation to achieve interactive frame rates.

Our ray density estimation metric has also been applied later for simplifying and boosting the computation of volumetric photon mapping [JZJ08], which just seems to be destined for this method. The main difference to our approach is that Jarosz et al. reversely apply this metric for gathering photons in a “viewing-ray-frustum” traversing the participating media whereas we gather only the surface hit-points of the viewing-rays in a conical frustum associated with the photon-ray (we assume vacuum without participating media).

Photon Ray Splatting

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 [Jen01], 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 [Sil85] 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* [Sil85, Sch03] (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* [Sch03, HBHS05] (i.e., wrong estimation of the surface area on complex surfaces). See the examples in Fig. 4.1.

Recently, Lastra et al. [LURM02] and Havran et al. [HBHS05] 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. 4.1c). 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. 4.1c) 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 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

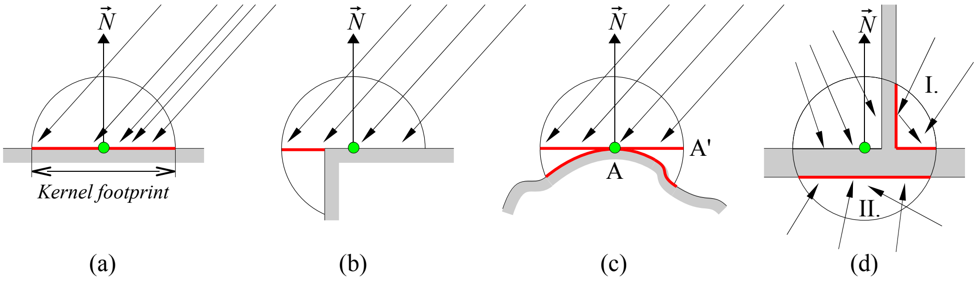


Fig. 4.1 – Bias sources in photon density estimation

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.

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. 4.1d).

A direct visualization of the photon map using either density estimation for rays in the tangent plane [HBHS05, LURM02] or hit points on the surfaces [Jen01] leads to insufficient quality. Therefore, for complex indirect lighting expensive final gathering must still be performed to achieve high-quality indirect illumination.

A remedy is to use better density estimation techniques with adaptive bandwidth selection developed in statistics [Sil85]. So far, only very few approaches have taken advantage of more sophisticated bandwidth selection in photon density estimation [Sch03, WHSG97, Mys97]. However, simply applying statistical bandwidth selection is not only inefficient but also ignores all gathered information during the photon generation phase. Most algorithms rely on the simple and robust k-nearest-neighbor (K-NN) density estimation [Sil85, Jen01], which only attempts to suppress the noise in the irradiance estimate to a uniform level without caring about introduced bias. Besides, exact K-NN searches are inefficient in particular for millions of search queries. Alternatively, we can reverse the process of the density estimation and "splat" photon energy to the nearest-neighbor pixel samples in a precomputed bandwidth per photon [HHK*07a, HHS05, LP03]. The disadvantage is that we loose the view dependence.

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 [HBHS05] nor relies on the spatial coherence of the density estimation queries. In contrast to [HBHS05] and [LURM02] 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, which is capable of handling illumination on complex geometry (e.g., wrinkled and bump-mapped surfaces). We replace the line density estimation in the tangent plane by energy splatting along photon rays to the eye samples.

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.

2 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. 4.2). 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 [K0̌5b].

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* [McC99, WKB*02] 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 11,12). Next, a kd-tree is constructed over the eye samples storing several eye samples per leaf (see Section 6).

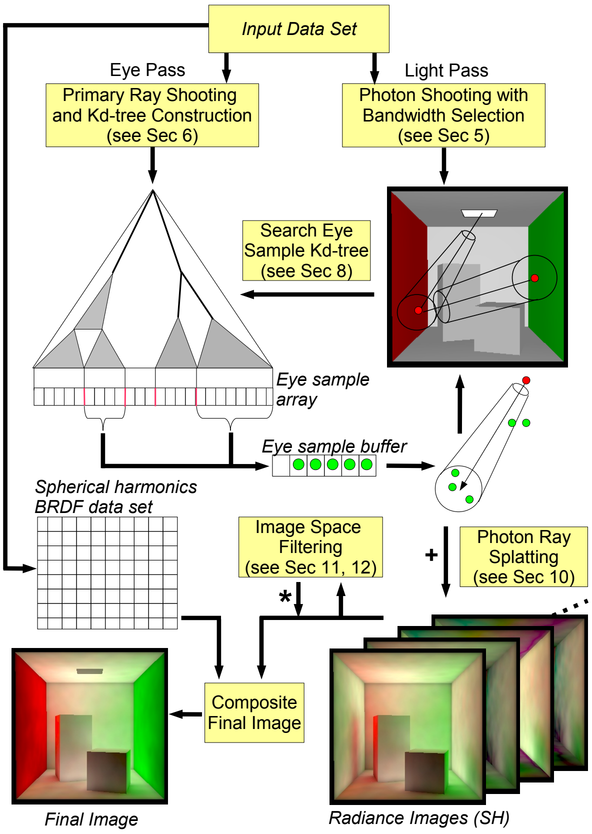

Input Data Set
Eye Pass
Light Pass
Primary Ray Shooting and Kd-tree Construction (see Sec 6)
Photon Shooting with Bandwidth Selection (see Sec 5)
Search Eye Sample Kd-tree (see Sec 8)
Eye sample array
Eye sample buffer
Spherical harmonics BRDF data set
Image Space Filtering (see Sec 11, 12)
+
Photon Ray Splatting (see Sec 10)
*
Composite Final Image
Final Image
Radiance Images (SH)

Fig. 4.2 – Processing flow in the photon-ray-splatting architecture

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

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 5).

Photon splatting: In the next phase all photons are splatted sequentially to neighboring eye samples in the vicinity of the photon rays (Section 3) using a density estimation kernel as described in Section 5. 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 8). 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 11) as well as the radiance cache spacing (Section 12).

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. 4.2.

3 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 [Sil85, Jen01]. 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 $\mathbf{y}$, 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 $\mathbf{x}_i$. Such an approach has been proposed by Bekaert et al. [BPC*03]. 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 (pdf) p_r for sampling the next photon hit point $\mathbf{x}_{i+1}$ from a particular photon location $\mathbf{x}_i$. This pdf is proportional to

$$p_r(\mathbf{x}_i \rightarrow \mathbf{x}_{i+1}) \propto V(\mathbf{x}_i, \mathbf{x}_{i+1}) p_s^{\perp}(\mathbf{x}_{i-1}, \mathbf{x}_i, \mathbf{x}_{i+1}) \frac{\cos\theta_{\mathbf{x}_{i+1}}}{\|\mathbf{x}_i - \mathbf{x}_{i+1}\|^2}, \tag{4.1}$$

where $p_s^{\perp}(\mathbf{x}_{i-1}, \mathbf{x}_i, \mathbf{x}_{i+1}) = p_s(\mathbf{x}_{i-1}, \mathbf{x}_i, \mathbf{x}_{i+1}) \cdot \cos\theta'_{x_i}$ is the pdf for sampling the cosine weighted BRDF at point $\mathbf{x}_i$. Refer to first Chapter Section 4 for explanation of the symbols used in the following sections.

Considering Eq. 4.1 we can draw some conclusions about the photon sampling density (i.e., flux density or irradiance) changes at hit point $\mathbf{x}_{i+1}$. If we assume the density estimation footprint at hit point $\mathbf{x}_{i+1}$ to be relatively small with respect to the squared distance $\|\mathbf{x}_i - \mathbf{x}_{i+1}\|^2$ (such that visibility $V(\mathbf{x}_i, \mathbf{x}_{i+1}) = 1$ is likely within the footprint) and the BRDF sampling density p_s at $\mathbf{x}_i$ to be of low-frequency, then the change in photon sampling density in the neighborhood of $\mathbf{x}_{i+1}$ mainly depends on the change in surface orientation $(\cos\theta_{i+1})$ in the vicinity of point $\mathbf{x}_{i+1}$. In normal photon mapping this factor is simply ignored, assuming the density estimation domain is planar and continuous near $\mathbf{x}_{i+1}$, which results in visible bias in corners and on curved surfaces (see Fig. 4.1). 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 [Jen01] and ray maps [HBHS05], 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 $dA_{\theta}^{\perp}$ and differential solid angle $d\omega$

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

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, the photon density is projected in the direction ω. This gives us the number of photons passing through a differential area $dA_{\theta}^{\perp}$ perpendicular to ω (see Fig. 4.3).

Equivalently, if we know the radiance $L(\mathbf{y}, \omega)$ for the differential area $d\omega$, we can compute its contribution to the irradiance of an arbitrarily oriented surface with area dA by multiplying $L(\mathbf{y}, \omega)$ with the cosine of the angle θ between surface normal and ω (see Fig. 4.3).

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

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

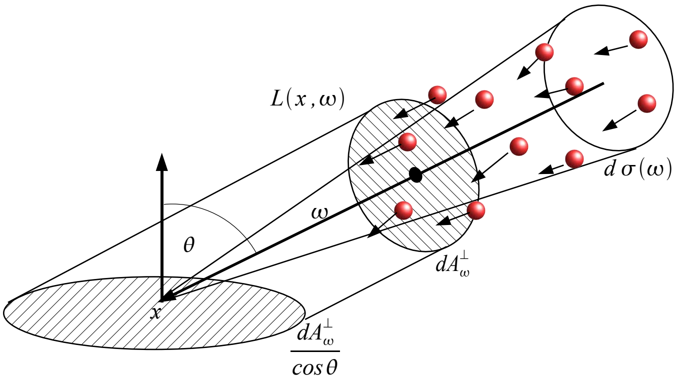


Fig. 4.3 – From photon-flux to radiance, a didactic view

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

Intuitively, this can be understood as accumulating the flux of all photons arriving in a small surface area around a point $\mathbf{y}$, 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 $\tilde{E}_1(\mathbf{y})$ to the real irradiance via density estimation

$$E(\mathbf{y}) \approx \tilde{E}_1(\mathbf{y}) = \sum_i^K \mathcal{K}_h(\mathbf{y},\mathbf{x}_i)\frac{\Delta\Phi_i(\mathbf{x}_i,\omega_i)}{\Delta A(\mathbf{y})}, \tag{4.4}$$

where $\mathcal{K}_h(\mathbf{x},\mathbf{y})$ is the density estimation kernel that satisfies $\int_S \mathcal{K}_h(\mathbf{x},\mathbf{y})dy = 1, \forall\mathbf{x}$. 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 $\mathbf{y}$ 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

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

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

$$L(\mathbf{y},\omega_o) \approx \sum_i^K f_s(\mathbf{y},\omega_i,\omega_o)\mathcal{K}_h(\mathbf{y},\mathbf{x}_i,\omega_i)\frac{\Delta\Phi_i(\mathbf{x}_i,\omega_i)\cos\theta_i}{\Delta A^{\perp}_{\theta_i}(\mathbf{y})},$$

where $f_s(\mathbf{y},\omega_i,\omega_o)$ is the BRDF at surface point $\mathbf{y}$ for incoming radiance direction ω_i and outgoing radiance direction ω_o. $\mathcal{K}_h(\mathbf{y},\mathbf{x}_i,\omega_i)$ 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 $\mathbf{y}$ to the ray $(\mathbf{x}_i,\omega_i)$ rather than the distance to the ray-intersection $\mathbf{x}_i$ in the local tangent plane (see Fig. 4.4)

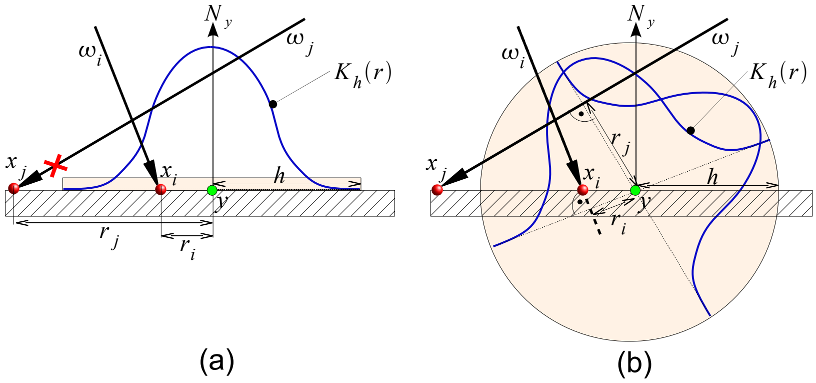


Fig. 4.4 – Density estimation metrics: tangent plane distance versus distance to photon ray

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 $\boldsymbol{y}$ contribute to the irradiance at $\boldsymbol{y}$.*

4 Photon Splatting Instead of Gathering

Instead of computing the photon density at an eye sample point $\mathbf{y}$ 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 [Sil85].

Variable KDE with adaptive kernel width is preferable over k-nearest neighbors (K-NN) density estimation [Sil85]. 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. 4.4b) is equivalent to searching for each photon all eye sample points in the cylinder centered along the photon's ray $(\mathbf{x}_i, \omega_i)$. 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 5. Each photon splats its energy weighted by a 2D kernel, which is aligned with its ray direction, to the found eye samples (see Fig. 4.5).

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. 4.5). 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. 4.5.

5 Choice of Splat Kernel and Bandwidth Selection

According to statistical studies [Sil85], 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* [Sch03, Sil85]:

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

As an alternative we have also tested the *biweight* or *quartic kernel* [Sil85], 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*

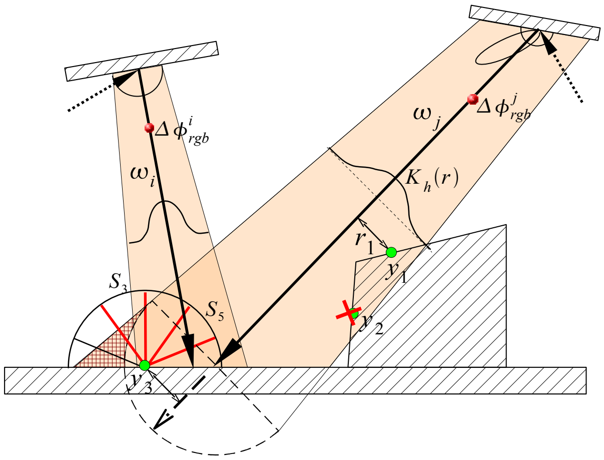


Fig. 4.5 – Concept of photon ray splatting instead ray gathering

Density estimation via photon ray splatting to stored eye samples ($\boldsymbol{y}_1$, $\boldsymbol{y}_2$, $\boldsymbol{y}_3$) in 1D. Photon j splats its energy within a certain kernel support (shaded area) of a function $\mathcal{K}(r)$ to $\boldsymbol{y}_1$ and $\boldsymbol{y}_3$, but not to back-facing $\boldsymbol{y}_2$. 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 10.1) 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.

or *bandwidth* (see [Sil85]). 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 heavily "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

$$p(X) \propto p_e(\mathbf{x}_0, \mathbf{x}_1) g(\mathbf{x}_0, \mathbf{x}_1) \prod_{i=1}^{n-1} p_s^{\perp}(\mathbf{x}_{i-1}, \mathbf{x}_i, \mathbf{x}_{i+1}) g(\mathbf{x}_i, \mathbf{x}_{i+1}), \tag{4.7}$$

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

Based on Eq. 4.7, we relate the bandwidth $h(\mathbf{x}_i)$ 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. [SW00] use a similar metric 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 $\mathbf{x}_{i+1}$ and the cosine term $\cos\theta_{\mathbf{x}_{i+1}}$ cancels out. Moreover, the path density $p(\mathbf{x}_i|\mathbf{x}_{i-1})$ can be arbitrarily small and arbitrarily large due to the distance term $\|\mathbf{x}_i - \mathbf{x}_{i+1}\|^2$ and we need to clamp it before being used in the bandwidth selection:

$$\tilde{p}(\mathbf{x}_{i+1}|\mathbf{x}_i) = \begin{cases} \frac{p_e(\mathbf{x}_0,\mathbf{x}_1)}{D(\mathbf{x}_0,\mathbf{x}_1)} & i = 0 \\ \tilde{p}(\mathbf{x}_i|\mathbf{x}_{i-1}) \cdot \frac{p_s^{\perp}(\mathbf{x}_{i-1},\mathbf{x}_i,\mathbf{x}_{i+1})}{D(\mathbf{x}_i,\mathbf{x}_{i+1})} & i > 0, \end{cases} \tag{4.8}$$

where $D(\mathbf{x}, \mathbf{y}) = \max(\tilde{D}^2, \|\mathbf{x} - \mathbf{y}\|^2)$ 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 [Kel97].

Using the bounded path probability-density defined in Eq. 4.8, we compute the bandwidth $h(\mathbf{x}_i)$ per photon ray by the following heuristic

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

where $h(\mathbf{x}_0) = 0$, C is the user defined "smoothness" parameter, and $S \in]0..1]$ is the user defined bandwidth sensitivity controlling the variance of $h(\mathbf{x}_i)$ (if S is set to 0, the bandwidth $h(\mathbf{x}_i)$ is constant for all rays).

According to the optimal bandwidth for minimizing the mean integrated square error, $h(\mathbf{x}_i)$ should be inversely proportional to the sixth root of the total number of samples M [Sil85]. 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 $r = 1/\sqrt{\tilde{p}(\mathbf{x}_i|\mathbf{x}_{i-1})^S}$. Coefficient w is then computed as

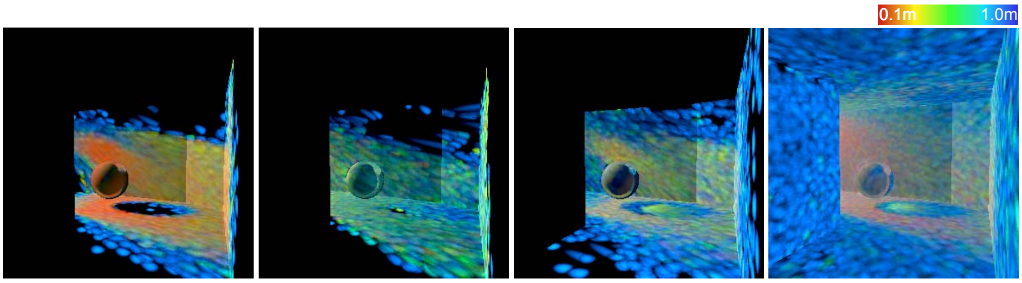
$$w = m_0 \frac{\mu_C}{\bar{r}}, \tag{4.10}$$

where the scene size dependent parameter

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

is computed from the average path segment lengths $\bar{D}(\mathbf{x},\mathbf{y})$ 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 $h(\mathbf{x}_i)$ such that the mean bandwidth $\bar{h} = C \cdot \mu_C$ for $M = 10^5$. For the sake of robustness, the resulting bandwidth $h(\mathbf{x}_i)$ 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 $\frac{C}{\sqrt[6]{M}} \cdot \mu_C \cdot R \leq h(\mathbf{x}_i) \leq \frac{C}{\sqrt[6]{M}} \cdot \mu_C / R$ holds.

Each photon ray $(\mathbf{x}_{i-1} \rightarrow \mathbf{x}_i)$ stores the initial bandwidth $h_0 = \min\{h(\mathbf{x}_i), h(\mathbf{x}_{i-1})\}$ and the differential bandwidth per ray length $dh = \frac{\max\{0, h(\mathbf{x}_i) - h(\mathbf{x}_{i-1})\}}{\|\mathbf{x}_i - \mathbf{x}_{i-1}\|}$, 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. 4.6, 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.



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

Fig. 4.6 – Visualization of photon-ray bandwidth selection

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 sphere.

6 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 [Jen01], 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) [LURM02, HBHS05]. 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 5.

7 The Splat KD-Tree Layout

We have chosen a standard axis-aligned kd-tree [WGS04, HHS05] 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 [HBHS05] 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 [HBHS05] is that we store not only the reference normal in the node but also the maximum angular deviation from the reference normal (see Fig. 4.7). This yields higher efficiency of successfully culling rays in particular on planar surfaces where the angular deviation of the normals is zero.

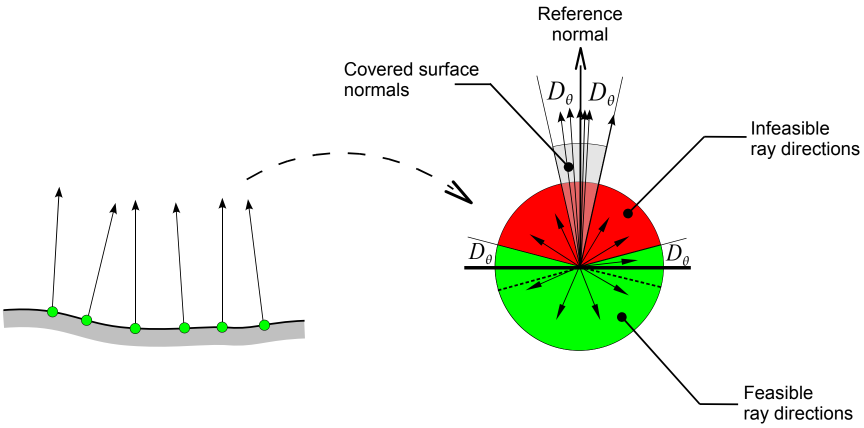


Fig. 4.7 – Ray-culling with directional kd-tree nodes

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 takes considerable time, 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. 4.7). If D_θ is smaller than a constant threshold ($15\,\mathrm{deg}$), 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.

8 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 [Dah04]. 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 [Dah04]. 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. 4.8. Therein, the ray needs to visit both front and back child-nodes. Fig. 4.9 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 10.

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 ($ray.dir \bullet node.normal > node.C_{max}$) 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
    $R_1 := ray.h_0 + ray.dh \cdot t_1$ /* compute splat radius at $t_1$ */
    /* compute ray length $\Delta t$ between virtual plane and splitting plane */
    $\Delta t := R_1\ /\ ray.dir[a]$
    $t := (node.plane1D - ray.org[a])\ /\ ray.dir[a]$
    /* compute ray lengths $t_\alpha$ and $t_\beta$ to virtual planes $\alpha$ and $\beta$ */
    $t_\alpha := t - \Delta t$
    $t_\beta := t + \Delta t$
    if $t_0 < t_\beta$ then
      /* traverse front */ TraverseTree($ray,\ i_n + 1,\ t_0,\ \min(t_\beta, t_1)$)
    end if
    if $t_1 > t_\alpha$ then
      /* traverse back */ TraverseTree($ray,\ node.offset,\ \max(t_\alpha, t0),\ t_1$)
    end if
  end if
```

9 Algorithmic Extensions

Like photon mapping [Jen01], 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.

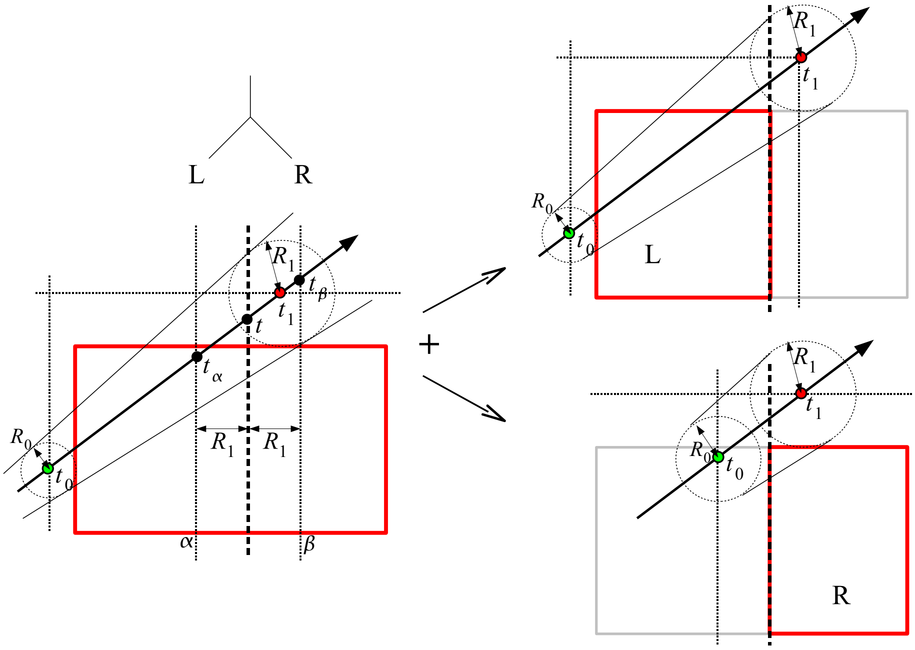


Fig. 4.8 – Conical frustum traversal of a spatial kd-tree node

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.

10 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 [Jen97] 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. [Stü98] 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 [LP03, HHS05]. 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

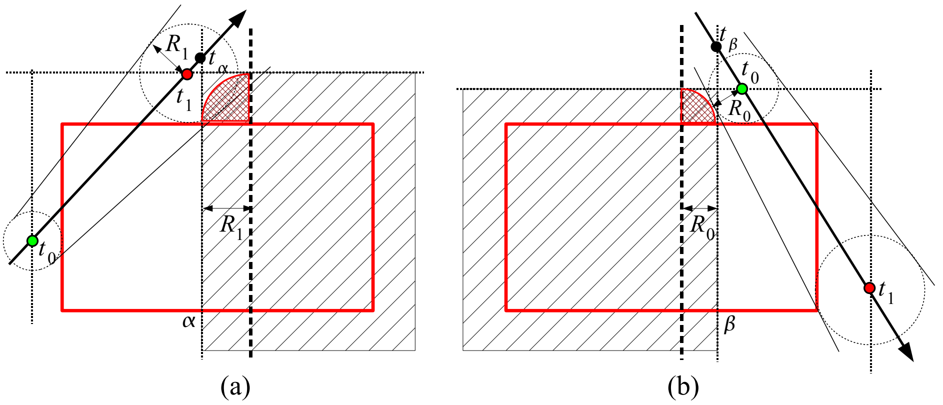


Fig. 4.9 – Conical frustum traversal of a spatial kd-tree node with conservative culling of sub-tree nodes

(a) If $t_1 \leq t_\alpha$ only the front child node needs to be traversed. (b) If $t_0 \geq t_\beta$ only the back child node needs to be traversed. 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 "rounded box", indicated by the 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.

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. 4.5).

The second approach uses a different basis for representing incoming radiance in frequency space. We have chosen spherical harmonics (SH) [Mac48, RH04], 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.

10.1 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 [RH04]. 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. 4.5). 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 [Stü98] 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.

10.2 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.

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 [Kŏ5b]. The BRDF SH coefficients $f_l^m(\theta'_o, \phi'_o)$ for every outgoing direction are precomputed by evaluating the integral

$$f_l^m(\theta'_o, \phi'_o) = \int\limits_{\Omega^+} f_s(\theta'_o, \phi'_o, \omega) \cdot Y_l^m(\omega)\, d\omega. \tag{4.12}$$

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

Algorithm 2 Photon ray splatting – ray density estimation metric

```
$SplatPhotonRay(ray, candidates)$
  $i_d := MapDirectionToStratum(ray.dir)$
  $W := coeffTab[i_d]$
  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 5) */
    $r_{es}^2 := \|D_{es} \times ray.dir\|^2$ /* 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
```

radiance SH coefficients λ_l^m for a certain pixel are computed as

$$\begin{aligned}\lambda_l^m &= \int\limits_{\Omega^+} L(\omega) \cdot \cos\theta \cdot Y_l^m(\omega)\, d\omega \\ &= \int\limits_{\Omega^+} \frac{d\Phi}{dA_\theta^\perp\, d\omega} \cdot \cos\theta \cdot Y_l^m(\omega)\, d\omega \\ &\approx \sum_i^K \frac{\Delta\Phi_i}{\Delta A_{\theta_i}^\perp} \cdot \cos\theta_i \cdot Y_l^m(\omega_i), \end{aligned} \tag{4.13}$$

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 [RH04] with small error. For our strategy, only the DC coefficient 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 6. 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

$$\lambda_l^m := \lambda_l^m + \mathcal{K}_{h_i}(\mathbf{y}, \mathbf{x}_i, \omega_i) \frac{\Delta\Phi_i(\mathbf{x}_i, \theta_i, \phi_i) \cos\theta_i}{\Delta A_{\theta_i}^{\perp}} \cdot Y_l^m[\theta_i, \phi_i]. \tag{4.14}$$

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.

$$\begin{aligned} L(\theta_o', \phi_o') &= \int_{\Omega^+} f_s(\theta_o', \phi_o', \omega) \cdot \cos\theta \cdot L(\omega) d\omega \\ &\approx \sum_{l=0}^{n} \sum_{m=-l}^{m=l} f_l^m(\theta_o', \phi_o') \cdot \lambda_l^m, \end{aligned} \tag{4.15}$$

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

11 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 need to 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 [WKB*02, McC99]. A naive algorithm is computationally expensive since it performs a convolution for a large number of pixels for all radiance images. In [HHK*07b] we presented a way to filter the radiance images in constant time per pixel making efficient use of discontinuity-preserving summed-area-tables. However, lookups in summed area tables correspond only to a rectangular filter, whose results are of inferior quality compared with a rotational invariant filter. Therefore, in Section 12 we will exploit the radiance cache splatting to filter noisy photon splats more effectively.

12 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 [WRC88, KŎ5b, GKBP05].

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 [WRC88, KŎ5b]. The cache sampling density adapts to a relative error estimate. The cache density in the original irradiance cache algorithm [WRC88] 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 [KŎ5b], we also perform the caching in the spherical harmonics basis. However, the difference lies in the illumination computation. While Křivánek [KŎ5b] 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 7). 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 [GKBP05]. However, the disc-shaped splat footprint of each cache record is computed in world space and projected to image space as visualized in Fig. 4.14b.

12.1 Approximative Harmonic Mean Distance to Visible Surfaces

The initial world-space radius r of each cache splat is computed from the harmonic mean distance $R(\mathbf{y}_c)$ [WRC88], which we measure only from incoming photon rays $(\mathbf{x}_i, \omega_i)$ during ray splatting.

$$\begin{aligned} r(\mathbf{y}_c) &= \varepsilon \cdot R(\mathbf{y}_c), & (4.16) \\ R(\mathbf{y}_c) &= \max\left\{ R_-(\mathbf{y}_c), \min\left\{ R_+(\mathbf{y}_c), \frac{1}{\sum_{i=1}^{K} \frac{1}{z_i}} \right\} \right\}, & (4.17) \end{aligned}$$

where ε is the user-defined cache error [WRC88], 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 $\mathbf{y}_c$ [KŎ5b, KGPB05, TL04]. 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 $\mathbf{y}_c$ to the ray origin $\mathbf{x}_i$. Because this is computationally intensive in particular for larger ray splat footprints, we approximate this distance by the projected distance $z_i = (\mathbf{y}_c - \mathbf{x}_i) \bullet \omega_i$ along the ray direction ω_i (see Fig. 4.10). This approximation is sufficient for our purposes and is computed as a by-product in the density estimation.

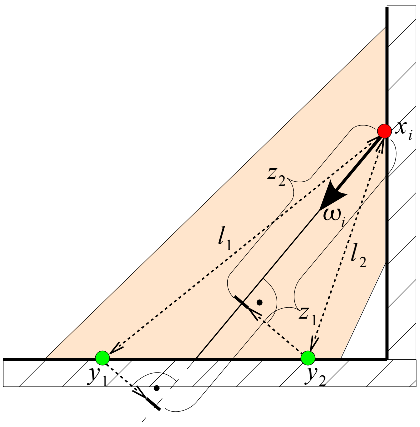


Fig. 4.10 – Approximating the harmonic mean distance to all surfaces visible to a cache record

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. 4.10), 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 [KBPv06] 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.

12.2 Radiance Cache Weighting

During cache splatting in image space a weight w_c is computed for each eye sample $\mathbf{y}$ 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 [WRC88] since it is unbounded and has a singularity at the cache location (see Fig. 4.12), which creates spiky artifacts as shown on the top left in Fig. 4.11. Instead we have chosen a smooth filter function, which goes to zero at the maximum cache distance $\varepsilon \cdot R_c$:

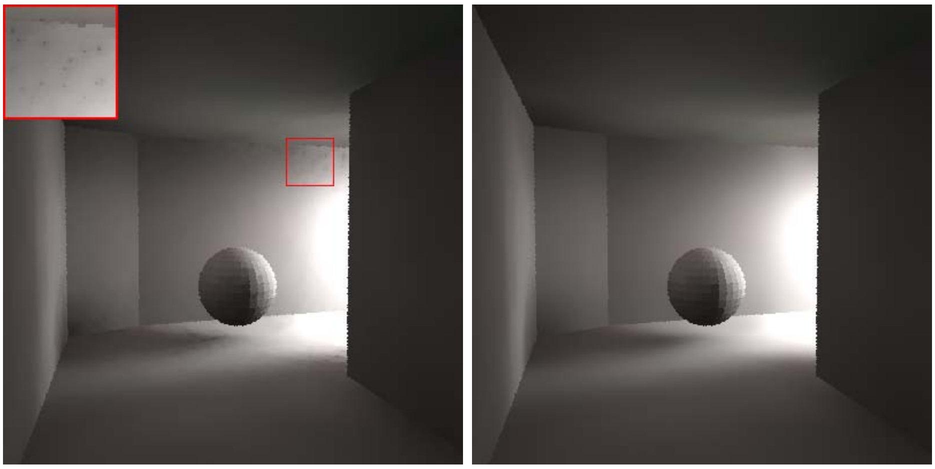


Fig. 4.11 – Radiance caching on top of photon ray splatting with a weighting function able to filter cache errors

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.

$$w_c(\mathbf{y}) = \left(\max \left\{ 1 - \frac{||\mathbf{y}_c - \mathbf{y}||^2}{\varepsilon \cdot R(\mathbf{y}_c)} \cdot \frac{1}{(\vec{n} \bullet \vec{n}_c)^4}, 0 \right\} \right)^2. \tag{4.18}$$

The term $\frac{1}{(\vec{n} \bullet \vec{n}_c)^4}$ penalizes cache records with deviating surface normal $\vec{n}_c$, where the exponent 4 was chosen for efficiency reasons. A comparison of Ward's weighting function with our weighting function is shown in Fig. 4.12 in 1D. If the weight

$w_c(\mathbf{y})$ 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.

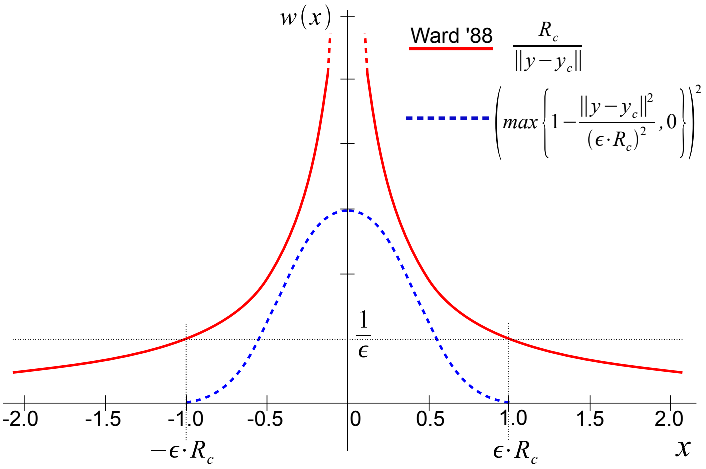


Fig. 4.12 – 1D plot of different cache weighting function

Comparing Ward's traditional cache-weighting function [WRC88] (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.

After each cache splatting pass the *cache splat image* [GKBP05], 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. 4.13.

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 $\mathbf{y}$ are divided by the accumulated weight per pixel, which has been stored in the cache

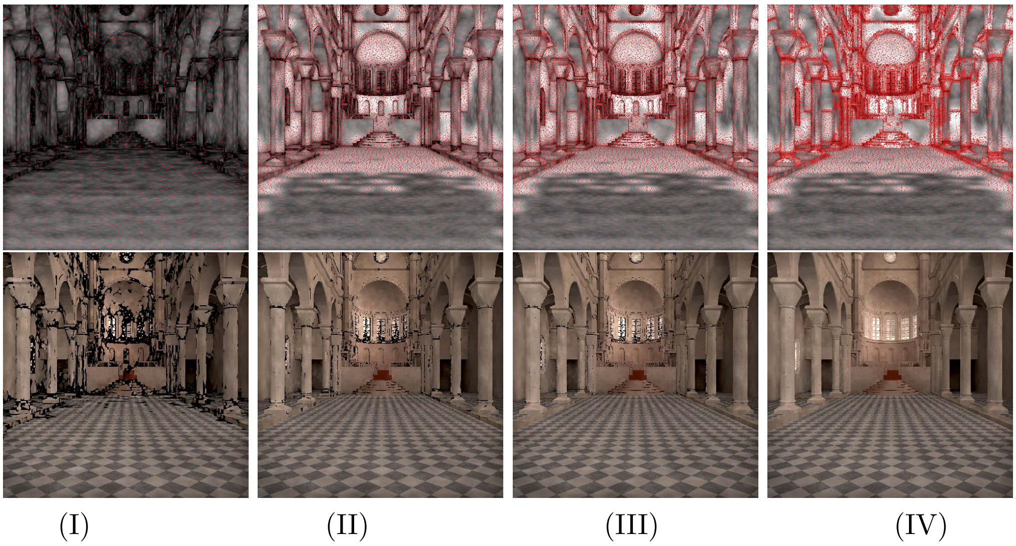


Fig. 4.13 – Visualization of iterative cache refinement with photon ray splatting

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 [WRC88]. 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).*

splat image:

$$\lambda_l^m(\mathbf{y}) = \frac{\sum_{c\in\mathcal{C}_y} \lambda_l^m(\mathbf{y}_c)\cdot w_c(\mathbf{y})}{\sum_{c\in\mathcal{C}_y} w_c(\mathbf{y})}, \tag{4.19}$$

where the $\mathcal{C}(\mathbf{y}) = \{c | w_c(\mathbf{y}) > 0\}$ is the set of contributing cache records to pixel sample $\mathbf{y}$. Note that we do not apply a rotation of the cached spherical harmonics (SH) coefficients [K̆05b] to align them with the local coordinate frame at $\mathbf{y}$ since the computational overhead of the SH rotation is too intensive compared with the computation of a new cache record in our framework. Therefore, our weighting function in Eq. 4.18 enforces a higher cache density on curved surfaces.

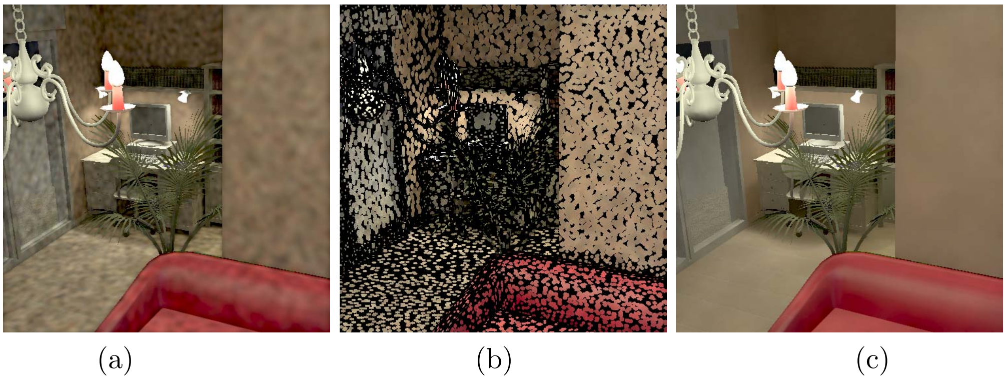


Fig. 4.14 – Visualization of initial cache-splat footprints and interpolated result for radiance caching with photon ray splatting

Results of photon ray splatting (direct and indirect light) with radiance caching in spherical harmonics basis in the Apartment *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 cache extrapolation in 3 iterations. The time for computing photon ray splatting and cache splatting in 3 passes took 15 seconds for* 500,000 *photons and* 500×500 *pixels.*

12.3 Illumination Gradient Estimation

When using (ir)radiance caching the cache-record density adapts to the geometric gradient magnitude estimated from the split-sphere model [WRC88], 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. [WH92] 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 [K0̌5b]. 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 [WH92] 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

[1] Increasing the cache density according to the gradient magnitude and at the same time

final gathering, we need a new approach for the gradient computation. Estimating the gradient $\nabla E(\mathbf{y})$ 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:

$$\begin{aligned}\frac{\partial}{\partial u}E(\mathbf{y}) &\approx \sum_{i=1}^{K}\frac{\Delta\Phi_i(\mathbf{x}_i,\omega_i)}{\Delta A^{\perp}_{\theta_i}}\cdot\left[\frac{\partial}{\partial u}\{\mathcal{K}_h(\mathbf{y},\mathbf{x}_i,\omega_i)\}\cdot\cos\theta_i+\mathcal{K}_h(\mathbf{y},\mathbf{x}_i,\omega_i)\cdot\frac{\partial}{\partial u}\{\cos\theta_i\}\right]\\ &= \sum_{i=1}^{K}\frac{\Delta\Phi_i(\mathbf{x}_i,\omega_i)}{\Delta A^{\perp}_{\theta_i}}\cdot\frac{\partial}{\partial u}\{\mathcal{K}_h(\mathbf{y},\mathbf{x}_i,\omega_i)\}\cdot\cos\theta_i, \qquad (4.20)\end{aligned}$$

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 $||\vec{\mathbf{r}}||$:

$$\begin{aligned}\frac{\partial}{\partial u}\mathcal{K}_h(\mathbf{y},\mathbf{x}_i,\omega_i) &= \frac{\partial}{\partial u}\left\{2\cdot\left(1-\frac{||\vec{\mathbf{r}}(\mathbf{y},\mathbf{x}_i,\omega_i)||^2}{h_i^2}\right)\right\}\\ &= \begin{cases}-\frac{2}{h_i^2}(2r_u) & ||\vec{\mathbf{r}}||^2<h_i^2\\ 0 & otherwise,\end{cases} \qquad (4.21)\end{aligned}$$

where $\vec{\mathbf{r}}(\mathbf{y},\mathbf{x}_i,\omega_i)=\{\omega_i\times(\mathbf{y}-\mathbf{x}_i)\}\times\omega_i$ is the distance vector to the photon ray and $r_u=\vec{\mathbf{r}}\bullet\vec{\mathbf{u}}$ is the projection into the local coordinate-frame axis $\vec{\mathbf{u}}$ at $\mathbf{y}$. ω_i is the ray direction in Euclidean coordinates. Note that the term $\frac{\Delta\Phi_i(\mathbf{x}_i,\omega_i)}{\Delta A^{\perp}_{\theta_i}}$ as well as the bandwidth h_i are assumed to be independent of the density estimation point $\mathbf{y}$ and therefore do not change when displacing $\mathbf{y}$ along $\vec{\mathbf{u}}$[2]. For a better understanding see the schematic draft in Fig. 4.15. And analogously for the $\vec{\mathbf{v}}$ direction:

$$\frac{\partial}{\partial v}E(\mathbf{y})\approx\sum_{i=1}^{K}\frac{\Delta\Phi_i(\mathbf{x}_i,\omega_i)}{\Delta A^{\perp}_{\theta_i}}\cdot[\frac{\partial}{\partial v}\{\mathcal{K}_h(\mathbf{y},\mathbf{x}_i,\omega_i)\}\cdot\cos\theta_i], \qquad (4.22)$$

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

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

$$R(\mathbf{y}):=\frac{E(\mathbf{y})}{||\nabla E(\mathbf{y})||}, \qquad (4.23)$$

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 $\mathbf{y}$ if the bandwidth is constant along the ray, i.e., cylindrical splatting footprint.

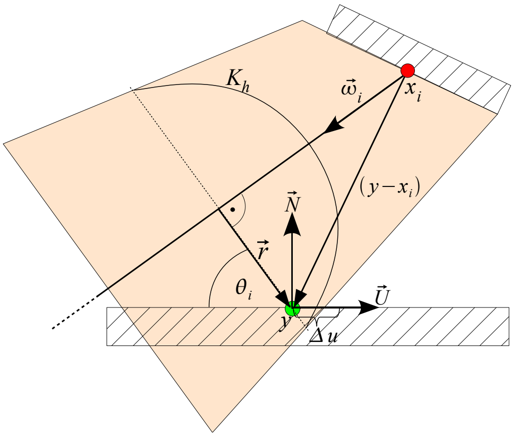


Fig. 4.15 – Quantities for computing a translational gradient in photon ray splatting

Quantities for computing a translational gradient in photon ray splatting (here in 1D along $\vec{\mathbf{u}}$).

which consequently results in an increased cache density around **y** (see Fig. 4.16). The images in Fig. 4.16 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. 4.20 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 [RS91] to filter the noisy gradients without cancellation effects [Wei98].

13 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*) [Jen01]. 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 by 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. 4.17(a). In such

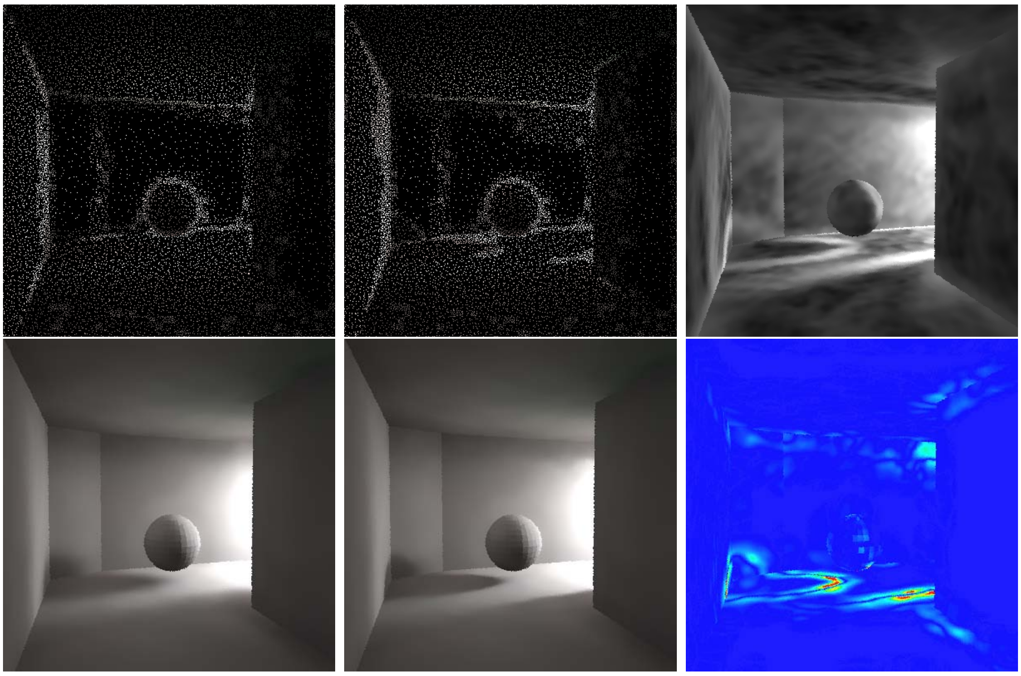


Fig. 4.16 – Visual results of photon ray splatting with irradiance caching with and without translational gradients

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 [WRC88], 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. 4.23. 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.*

cases secondary final gathering is initiated, drastically increasing the computation cost of the corresponding pixel, which is especially problematic for (ir)radiance caching [WRC88] 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 2. 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 [HHS05]. Further, we do not use the radiance map (Section 10), 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.

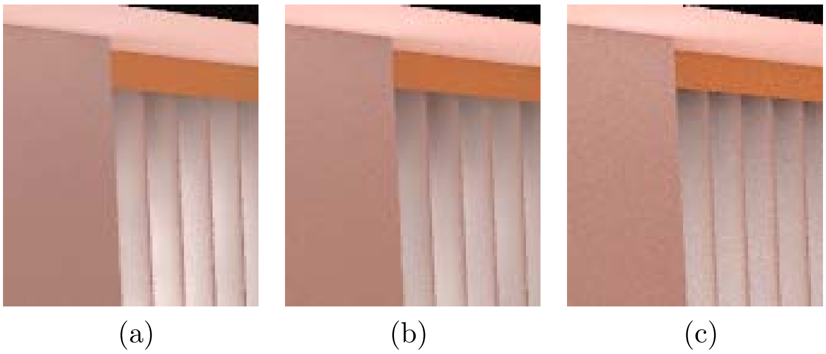


Fig. 4.17 – Visual results of photon ray splatting with final gathering and photon mapping compared to unbiased reference

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. 4.21(d).

14 Progressive Photon Ray Splatting

Our photon ray splatting naturally allows for progressive refinement in the spirit of *progressive photon mapping* (PPM) [HOJ08]. The main difference between PPM and our approach is the computation of the kernel bandwidth. Whereas PPM is based on the k-nearest neighbors scheme, we compute the bandwidth h for each photon individually as explained in Section 5. Similar to the PPM formulation proposed in [HJJ10], in each iteration i we allow the variance of the radiance estimation error ε_i to increases by a factor

$$\frac{Var\left[\varepsilon_{i+1}\right]}{Var\left[\varepsilon_i\right]} = \frac{i+1}{i+\alpha}, \tag{4.24}$$

for some constant $\alpha \in]0\ldots1[$. Further, in [HJJ10] it has been shown that the variance decreases with the square of the kernel radius h, which can be computed from the sequence of variance estimates using the following relationship

$$\frac{h_{i+1}^2}{h_i^2} = \frac{Var[\varepsilon_i]}{Var[\varepsilon_{i+1}]}. \tag{4.25}$$

Then, given an initial bandwidth h_1 and combining the two equations above, we can reformulate an explicit equation for computing the kernel size in the i-th iteration leading to an asymptotic estimate

$$h_i^2(\mathbf{x}) = h_1^2(\mathbf{x}) \cdot \left(\prod_{k=1}^{i-1} \frac{k+\alpha}{k} \right) \frac{1}{i}, \tag{4.26}$$

where h_1 is computed for photon at $\mathbf{x}$, and α determines the convergence rate of variance versus bias [HJJ10]. Note that our computation is memoryless and can be parallelized since we do not store h_1, but recompute it for each photon ray in each iteration during the photon tracing phase. An example of our progressive ray splatting compared to PPM is shown in Fig. 4.18. The variance/bias convergence-rate factor was set to $\alpha = 0.5$ and $500{,}000$ photons where used in each iteration. Our initial bandwidth parameters ($C = 0.8, S = 0.4$) were set to yield a mean bandwidth ($\bar{h}_1$) similar to the mean radius comprising 500 nearest neighbors. While our method produces already acceptable results initially, PPM converges slowly near boundaries and on glossy surfaces[3].

15 Results

We have evaluated our method for the scenes shown in Fig. 4.20 and Fig. 4.21. The scene Apartment 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.

We compared our method with standard K-NN density estimation using a direct visualization of the photon map [Jen01]. The rendering times are given in Table 4.1. 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

[3] *Stochastic progressive photon mapping* [HJ09] has been proposed to improve convergence on glossy surfaces and to simulate various other view-dependent effects (e.g., depth of field). However, it requires re-shooting (final gathering) of eye paths for each iterations.

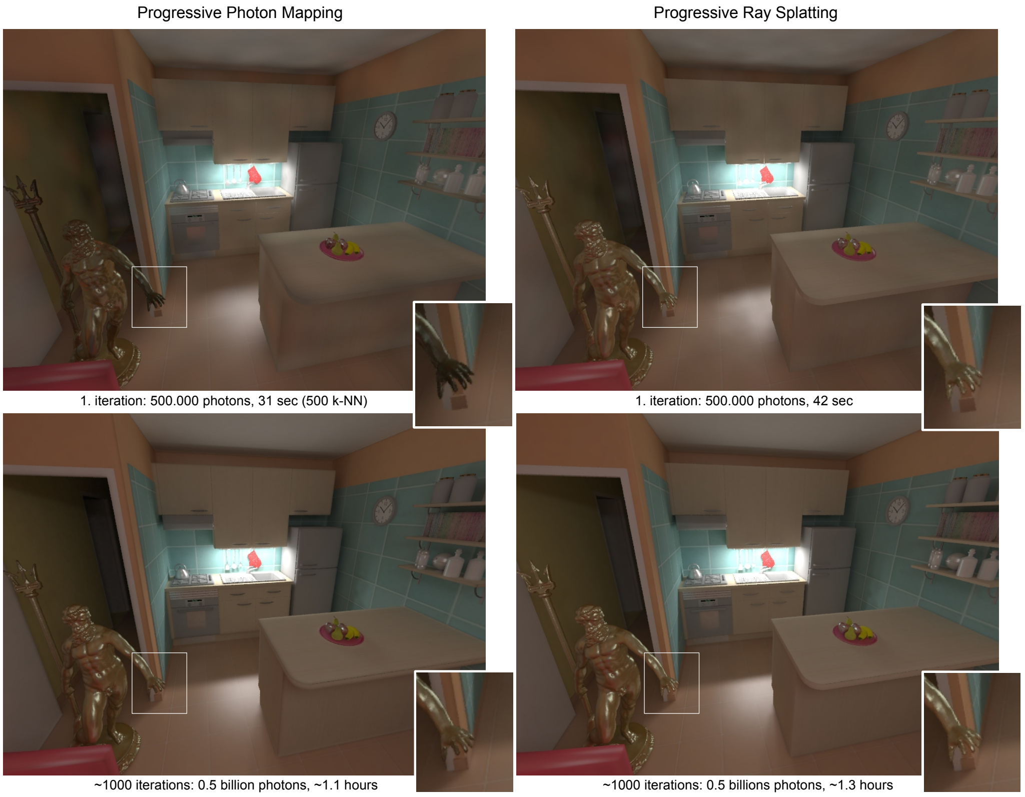

Progressive Photon Mapping
Progressive Ray Splatting
1. iteration: 500.000 photons, 31 sec (500 k-NN)
1. iteration: 500.000 photons, 42 sec
~1000 iterations: 0.5 billion photons, ~1.1 hours
~1000 iterations: 0.5 billions photons, ~1.3 hours

Fig. 4.18 – Progressive ray splatting versus progressive photon mapping

Progressively splatting photon rays to eye samples leads to high-quality and consistent results (right), which converge faster than for traditional progressive photon mapping [HJJ10] (left). Note that even after 500,000,000 *traced photons progressive photon mapping still suffers from boundary bias.*

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 [Jen97], 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).

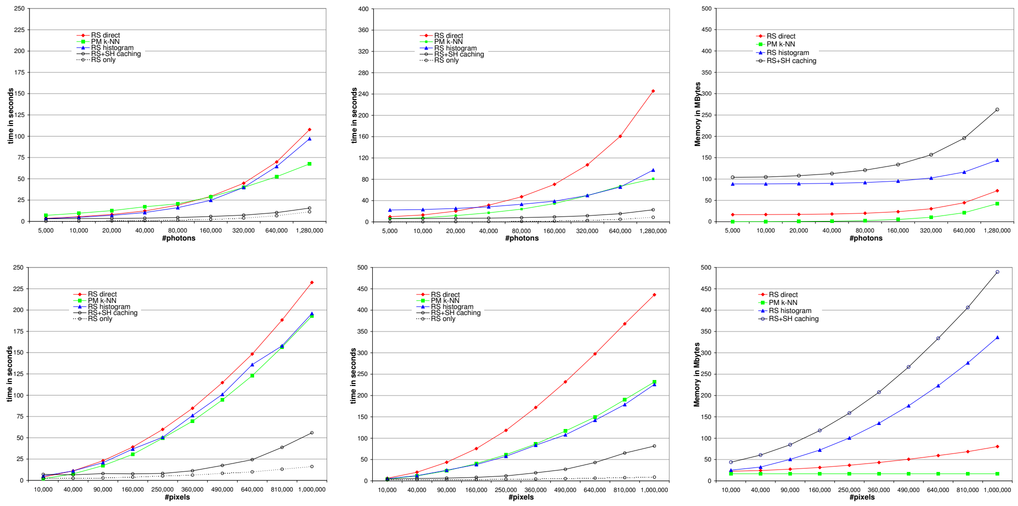
The ray splatting approach also scales well with image resolution and with number of photons. The time and memory dependencies on image resolution and photon number are shown in Fig. 4.19. The graphs show the measured rendering times (columns a and b) and memory usage (column c) of four different methods for the CORNELL BOX (column a) and the APARTMENT 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 APARTMENT 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 4.1). 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 [HHS05] 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.



(a) (b) (c)

Fig. 4.19 – Rendering times and memory consumption with respect to different parameters for photon ray splatting compared to traditional photon density estimation

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* Apartment *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* Apartment *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.*

The ray density estimation and search in a conical frustum (Section 6) 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.

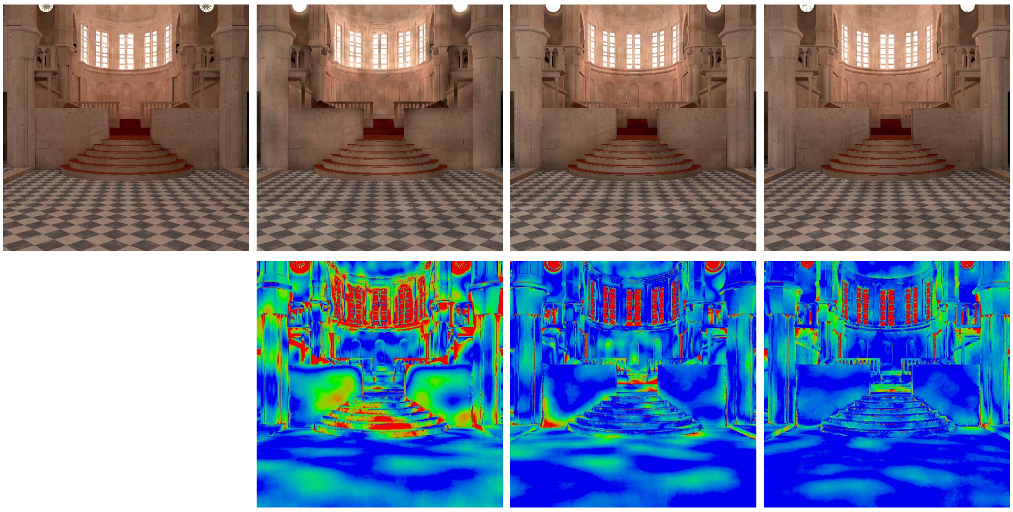
The bandwidth smoothing parameter C determines the noise level in the density estimation, while the sensitivity S controls the variance ($S = 0$ results in a constant bandwidth). Since we normalize the bandwidth term using an initial pilot estimate (Section 5), C is relatively independent of scene size 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 13), 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 5) and cache error ε (Section 12). 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. 4.20d). 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 4.

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

In Fig. 4.20 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 $\boldsymbol{C}$ (see Section 5) 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. 4.14). 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) 3950 s (b) 29 s (c) 35 s (d) 20 s

Fig. 4.20 – Error visualization of photon ray splatting and photon density estimation

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.*

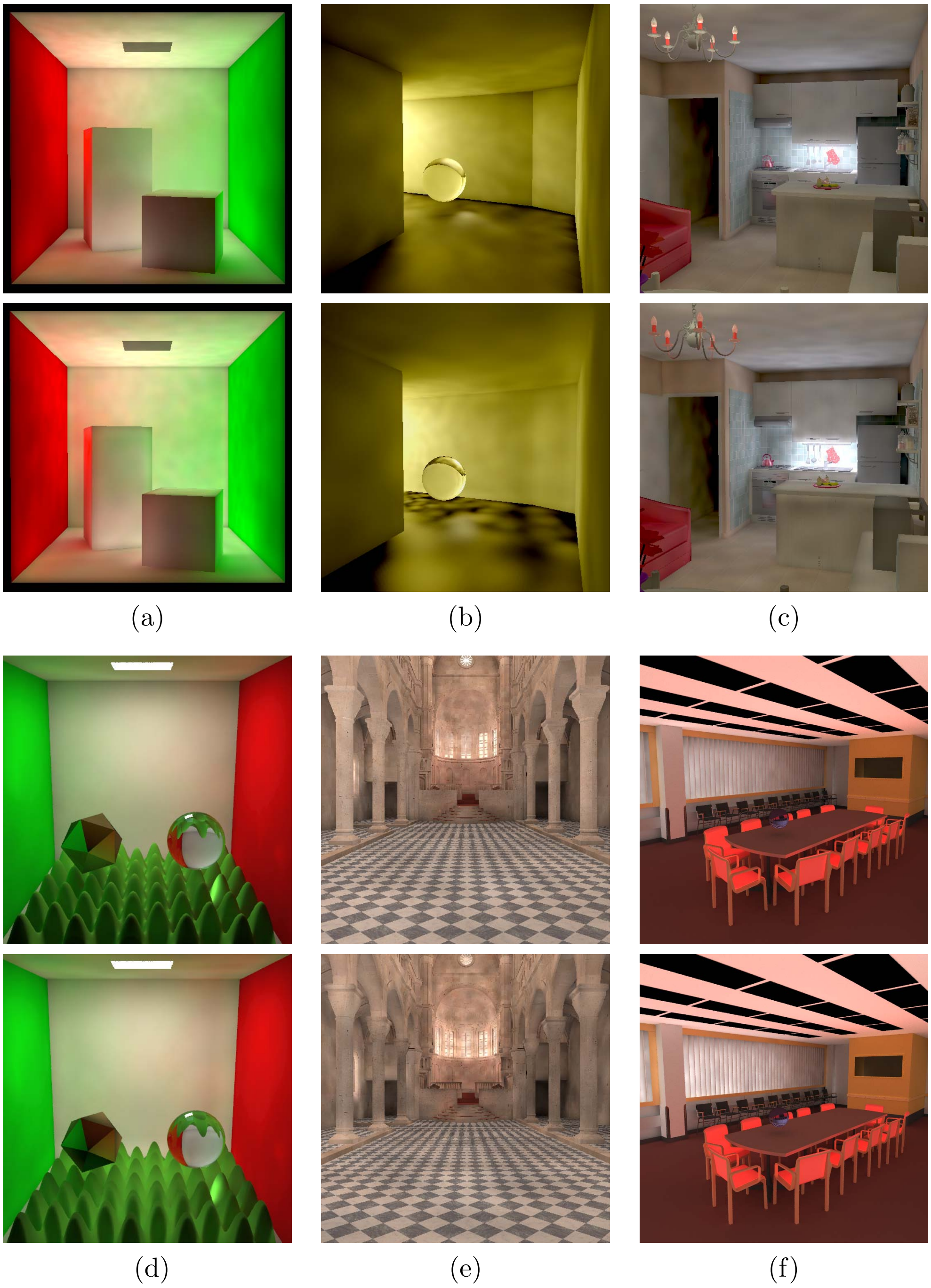


Fig. 4.21 – Visual results for photon ray splatting versus photon mapping

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 4.1). 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* Apartment *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.*

16 Discussion and Future Work

Despite the numerous extension we proposed in this chapter, our general photon ray splatting algorithm leaves space for further optimization and extensions of various kind. It has already been extended and improved in different follow-up publications references in Chapter 3. Next, we will propose a few ideas that our method could be extended and used for. 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.

16.1 Correcting Visibility in the Splatting Footprint

The major drawback of our algorithm as for 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. In the following chapter, we will propose simple yet efficient ways to approximate the visibility during the kd-tree traversal.

16.2 Temporally Coherent Photon Ray Splatting

Second, we would like to extend our method into the temporal domain by reusing radiance information from previous frame(s). Inspired by [SKDM05] 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 [WMM*04b], we could also reduce the low-frequency noise from photon sampling and increase the visual quality of the images.

16.3 High-quality Irradiance Precomputation

Our method can also be used in a preprocessing step for computing a good 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 [Chr99].

16.4 Volumetric Photon Ray Splatting

Another interesting possible extensions of photon ray splatting is the computation of volumetric lighting effects in participating media. Instead of storing photon-volume-interactions as point samples in the media, we could keep the whole ray segments of the photon paths and instead store eye sample points and their associated weights corresponding to the line integration steps in the volume along the viewing rays. Of course, as for the proposed combination with final gathering described in Section 13 such approach would quickly eat up the physical memory resources when storing all generated eye samples for all pixels at once. Therefore, depending on the volume step size this would need multiple ray-splatting passes for rendering the entire image. It would be interesting to see whether such combination could cover all volumetric light paths with sufficient quality without the need for a separate integration pass for the single scattering in participating media [Jen01].

Another possible application for photon ray splatting is to use it for precomputing or updating the cached radiance in an irradiance volume [Oat05, GSHG98]. It is straightforward to extend our radiance caching proposed in Section 12 to irradiance volumes.

16.5 Scalable Photon Ray Splatting in a Distributed Setup

There are different possibilities for mapping the algorithm to a distributed setup with multiple processors. Considering hundreds of thousands to millions of photons, the photon splatting is still the most time consuming phase of our algorithm. Therefore, the perhaps simplest approach to split the work-load to multiple processors is to instantiate the entire kd-tree and all eye samples on several clients whereas the photons are distributed among them (for example by using different quasi-random seeds for the photon sampling on each client). Because light is additive the resulting radiance images (radiance SH coefficients) can be accumulated in a final pass on the server.

However, the generation of the eye samples followed by the construction of the splat kd-tree and the final summation of many radiance images may take a considerable time. Thus, a more scalable approach would split the entire computation, i.e., the image sampling and composition and the photon splatting, into multiple tiles computed on different clients. In order to avoid discontinuities along the tile boundaries, one can apply *interleaved sampling* [WKB*02] for the photons as well as the eye samples. In order to do so we can interleave the image sampling to generate several smaller resolution images on each client where each such image is computed with a different set of photons similar to [WKB*02]. This way we split the number of eye samples as well as the number of photons among the clients. The price we pay is structured noise from the interleaved sampling pattern, which needs to be filtered in a final pass using a discontinuity buffer [WKB*02] as the one proposed in Section 11.

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 [Chr99].

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	$500 \times 500 \times 1$; $M = 500{,}000$; $(0.0, 1.0, 0.0)$; $R = 0.2$								
Apartment	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	2.49	0.16	4114	–	–	5108
	RS-FG (600 FGRs, 25 tiles)	0.83	997	2.58	111	2714	–	–	3825
Scene settings	$700 \times 700 \times 4$; $M = 160{,}000$; $(0.3, 0.6, 0.1)$; $R = 0.4$								

Table 4.1 – Statistics of ray splatting compared with photon mapping

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. 4.21. 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 × 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.

Lighting Details Preserving Photon Density Estimation

This chapter extends the photon ray splatting introduced in previous chapter by approximative visibility-test in the density estimation.

The chapter is organized as follows. In Section 2 we generalize our ray density estimation framework, Section 3 briefly describes the algorithm steps. Section 4 summarizes the scene voxelization algorithm on the GPU. It also describes how the resulting voxel grid can be used for our density estimation technique. Section 5 introduces the new hierarchical data structure for efficient photon splatting. Section 6 reveals the limitations of the method and provides some remedies. Section 7 presents results of our method compared with alternative techniques. Finally we conclude with a discussion and present some ideas for future work in Section 8.

1 Visibility and Density Estimation

The proposed photon ray splatting in Chapter 4 solves two major drawbacks of traditional photon density estimation, the boundary bias and the topological bias problem [HBHS05, Sch03]. However, as described so far the photon ray splatting cannot improve the proximity bias because it does not account for visibility changes within the splatting footprint, which results in light leaking and washed out shadows. Visibility changes yield occlusions (shadows) which impose the highest frequencies in the illumination especially for direct light. Without explicit visibility testing, photon ray splatting and all other photon density estimation approaches only work satisfactory for smooth indirect illumination. One way to solve this problem is to preserve the visibility in the density estimation footprint. 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. A conservative approach like for example beam tracing or shadow maps would be too expensive, in particular that stochastic indirect light paths can be highly incoherent, and ruins the efficiency of photon density estimation, which computes the convolved visibility implicitly via density estimation. The brute-force solution would test each eye sample in the splatting footprint explicitly for occlusion with the photon ray's origin via ray-tracing. Such approach has been proposed in [BPC*03], where they stochastically decide whether to shoot a shadow ray or not.

On the other hand, visibility of indirect illumination is less critical than for direct illumination [RGK*08, GKD07] as the visibility term under the integral of the rendering equation is averaged over the entire hemisphere of directions of mostly low-frequency illumination further convolved with a low-frequency BRDF. Therefore approximative visibility computations are often sufficient for indirect illumination [RGK*08, GKD07].

1.1 Approximating Visibility

Most global illumination algorithms work on a boundary representation such as polygonal meshes, which only represent the surface of objects. This exact computation is often too conservative for indirect illumination in particular when convolving the visibility via density estimation. Therefore, in the following sections we will focus on methods for efficiently approximating visibility suitable for our photon ray splatting framework.

1.1.1 Screen-Space Visibility Approximation

A simple visibility approximation exploits the additional spatial information of the eye samples gathered during the ray traversal of the kd-tree, which can be considered as a point sample representation of the scene's surfaces. 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. 5.1 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 [ZPvBG01]. 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.

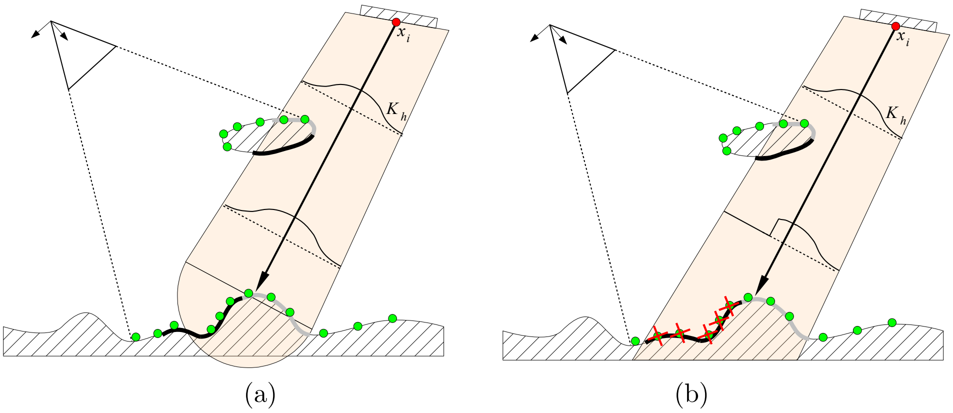
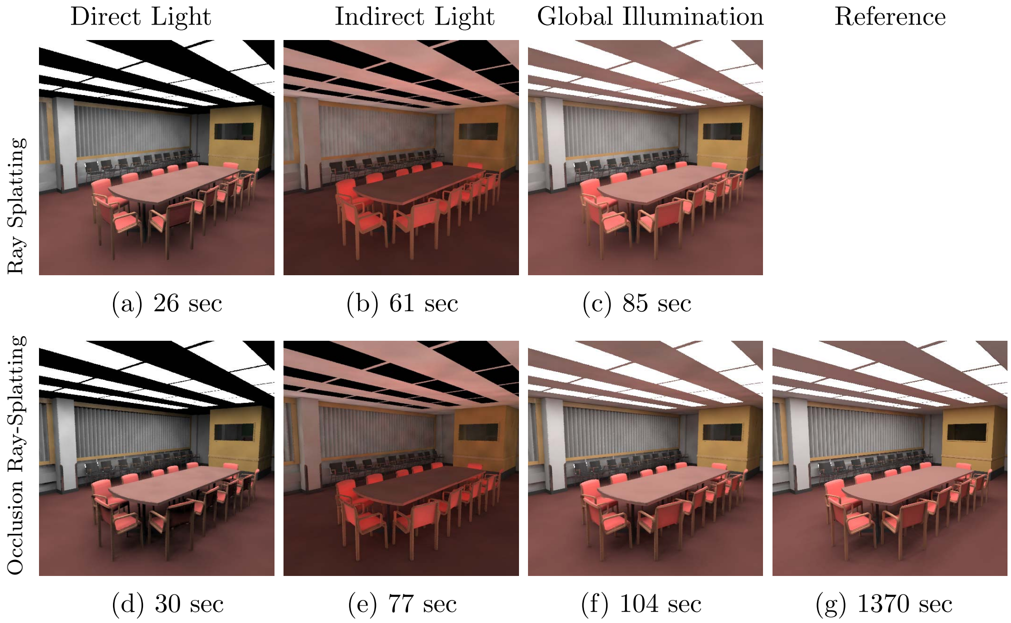


Fig. 5.1 – The concept of occlusion masking for photon ray splatting

(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. 5.2 second row).

1.1.2 Approximating Visibility by a 3D Scene Voxelization

Screen space visibility is only correct if occluders are visible and may miss important shadows cast by hidden occluders. In the following we look at a more conservative alternative to approximate the visibility in synthetic 3D scenes. We create a voxel representation of a polygonal scene, for which we make efficient use of modern rasterization hardware (GPU). We still need the surface representation for computing exact ray intersections and for the scattering of photons. However, the voxelized scene representation is well-suited for occlusion and visibility testing for volumetric ray traversal algorithms as our global illumination framework. It avoids many expensive ray intersection tests inside a ray volume. The occlusion detection has similarities with shadow mapping, which was designed for modern rasterization hardware. However shadow mapping is a global technique considering the whole scene, which is not efficient for indirect lighting with hundreds of thousands of virtual light sources. As a side-effect the voxel representation enables also antialiasing of shadow boundaries since sharp surface boundaries are pre-filtered per voxel during the scene voxelization.



(a) 26 sec (b) 61 sec (c) 85 sec

(d) 30 sec (e) 77 sec (f) 104 sec (g) 1370 sec

Fig. 5.2 – Visibility-preserving photon ray splatting

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 [WFA*05] using an average of* 339 *shadow rays per pixel* (250,000 *virtual point light sources).*

2 Visibility-Preserving Photon Density Estimation in Ray-space

As we have stressed before, the high frequencies in the illumination are the main source of bias in photon density estimation, which mostly result from surface normal variation and wrong visibility assumptions in the neighborhood. The former is solved by operating in ray space (see Section 3) and decoupling the photon density estimation from the surfaces. The latter is more complex because it requires to evaluate the visibility function inside the density estimation footprint between the origin of the photon (x_{i-1}) and all eye samples in the density estimation footprint, which ruins the efficiency of photon density estimation methods. Nevertheless, we will provide an efficient approximation for the visibility estimation. Let us rearrange Eq. 4.4 to exclude the visibility function $V(\mathbf{x},\mathbf{y})$ and the surface orientation $\cos\theta$ from the density estimation to be computed explicitly:

$$E(\mathbf{y}) \approx \sum_{i}^{K'} \mathcal{K}_h(\mathbf{y},\mathbf{x}_i,\omega_i) V(\mathbf{y},\mathbf{x}_{i-1}) \cos\theta_i \frac{\Delta\Phi_i(\mathbf{x}_i,\omega_i)}{\Delta A^{\perp}_{\theta_i}(\mathbf{y})}, \tag{5.1}$$

where $\mathcal{K}_h(\mathbf{y}, \mathbf{x}_i, \boldsymbol{\omega}_i)$ is the density estimation kernel whose domain is oriented perpendicular to the direction $\boldsymbol{\omega}_i$ as illustrated in Fig. 4.4. The point $\mathbf{x}_{i-1}$ is the origin of the photon ray and θ_i is the angle between the photon ray and the surface normal at point $\mathbf{y}$. $\Delta A^{\perp}_{\theta_i}(\mathbf{y}) = \Delta A(\mathbf{y}) / \cos\theta_i$ is the area of the unprojected density estimation footprint. See Fig. 5.3 for a geometric interpretation in 2D. Note that we partially compute the geometric term[1]

$$G(\mathbf{x}, \mathbf{y}) = V(\mathbf{x}, \mathbf{y}) \frac{\cos\theta_x \cos\theta_y}{||\mathbf{x} - \mathbf{y}||^2}. \tag{5.2}$$

Since we use a splatting approach, one photon ray splats its energy to all visible eye sample points $\mathbf{y}$ inside its kernel footprint as shown in Fig. 5.3.

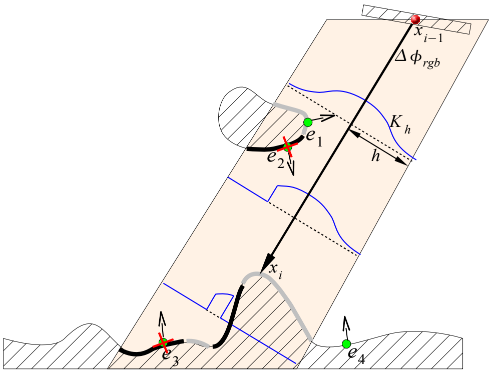


Fig. 5.3 – Visibility Splatting

Photon splatting in ray space for one photon ray with care of occlusions and surface orientation (shown in 2D). To preserve the photon energy $\Delta\Phi_{rgb}$, the axis-aligned kernel footprint (shaded parallelogram) is tested for occlusion along its traversal. The figure shows the masked kernel K_h (blue) and the occluded areas (thick black curves) and visible areas (thick grey curves). The photon flux is only splatted to eye samples e_1 because e_2 is back-facing, e_3 is occluded, and e_4 is outside the kernel footprint.

[1] Our density estimation metric has a similarity with *instant radiosity* [Kel97] in the sense that a photon's energy contribution to an eye sample is explicitly weighted by the cosine and visibility term for the incoming photon direction. However, in contrast to instant radiosity, our method does not suffer from singularities near corners since the squared distance term in the denominator of the geometric term is implicitly handled by the density estimation.

3 Algorithm Outline

Our density estimation algorithm uses the splatting technique described previously with the difference that we additionally account for occlusion of individual eye samples in the density estimation. First, all primary rays are shot from the eye. Then hit-point records with the scene surfaces referred to as *eye samples* are stored. Second, the scene is voxelized into a grid using the GPU and a voxel hierarchy, basically an axis-aligned kd-tree, is constructed over the generated grid. During the tree construction the eye samples are inserted into the hierarchy at the appropriate level. Next, the photons are traced through the scene using Quasi-Monte Carlo sampling until a desired number of direct, caustics, and global indirect photons has been stored. During the photon sampling phase a splat radius is computed from the entire photon path for each individual photon and for each type of light transport as described in Chapter 4. The splat radius and the photon flux is then stored together with the ray parameters. In the following phase all photon rays are sequentially splatted to the image plane using the novel density estimation technique that preserves the orientation and visibility of each eye sample. For each photon ray a search is initiated for the nearest-neighbor eye samples in a cylindrical volume associated with the ray, which we call the splatting footprint. For accelerating the nearest neighbor search and at the same time testing for occlusion inside the splatting footprint, the generated voxel hierarchy is traversed in front to back order in the spirit of raytracing with kd-trees. However, raytracing algorithms work in 1D ray space, while our algorithm is of volumetric nature.

4 Synthetic Scene Voxelization

The basis of our global illumination algorithm is the discretization of the scene model. Much research has been devoted to the problem of generating surfaces from measured volumetric data that was initially sampled in 3D. This was necessary because of memory limits and slow rendering hardware. However, for a few applications it is beneficial to keep the sampled data rather than generating continuous surfaces from it. In [DCB*04, ED06] it was pointed out that a discretization of a continuous scene model in form of a 3D grid with voxels as primitives has several advantages. First, one can exploit the volumetric pre-filtering per voxel for anti-aliasing purposes. Second, the voxelization is independent of the scene complexity. However, aliasing may occur if the scene is sampled too coarsely without pre-filtering. Third, it is more general and any kind of primitives, e.g. implicit surfaces, polygons, participating media and naturally measured volume data (e.g. CT scans) can be sampled and placed in a volumetric grid. Thanks to modern hardware, storing a high-resolution grid does not impose serious problems and the precomputation of the grid is nowadays possible in real-time [DCB*04,ED06] due to mainstream high-performance graphic cards.

Summarizing, the *voxelization* process runs as follows: First the scene model is, if not yet present, tessellated to triangles[2]. Next, all triangle vertices are passed to the GPU memory as vertex buffer objects. OpenGL states are set, in particular the logic operation must be set to 'OR' mode. For all three dimensions, the axis-aligned bounding box of the scene is divided into intervals composed of multiple grid slices. The number of grid slices depends on the hardware capabilities, which currently supports 128 bits per pixel using 4 render targets with 4 channels à 8 bits per pixel (for logical operations). Since we rasterize front and back-facing triangles separately, we can handle 64 slices at once. For each interval all triangles are rasterized by an orthographic camera with its frustum defined by the current grid interval boundaries. All triangles falling into the currently processed interval are voxelized in the fragment shader according to their interpolated z-distance such that each bit in a color channel determines whether a voxel of a slice is empty or not (see [ED06] for details). In addition, we apply 4×4 super-sampling per voxel for anti-aliasing purposes and also for the reconstruction of a discretized average normal per voxel. This assumes the surface crossing the voxel is piecewise linear inside the voxel. The composition of the 3D occlusion grid and the following hierarchy construction are computed on the CPU.

4.1 Photon Ray Splatting in the Voxelized Scene Grid

Before explaining the traversal of the voxel hierarchy, we describe our initial algorithm for traversing the uniform grid that is directly taken from the output of the voxelization process. For the ray traversal we used a 3D DDA algorithm as often used in standard raytracing algorithms, however for "thick" cylindrical rays. Therefore, not only the currently traversed voxel has to be tested for occlusion but also the neighboring voxels within the cylindrical slice (see Fig. 5.5 for an example in 2D), which follow the same 3D DDA traversal sequence. This corresponds to a parallel projection. Those traversed slices are always aligned with the grid axes and are defined by an axis-aligned cut through the cylinder which is perpendicular to the largest dimension (N) of the ray direction. The width and the height (axes U and V) of the elliptically shaped slice depend on the photon's splat radius associated with the ray, which is precomputed before each ray traversal. The traversal history of the axis-aligned slice is stored in a depth buffer, which we call the *occlusion mask*. In each 3D DDA traversal step the new center voxel of the mask is computed (this is the voxel the ray traverses) and all neighboring voxels overlapping with the mask are tested for occlusion. All non-empty voxels update the occlusion in the corresponding cell of the occlusion mask until a cell is fully

[2] Optionally, we test the polygonal scene for wrongly oriented triangles by performing a random walk by means of raytracing through the whole scene. Triangles that are detected as back-facing are flipped and triangles, which were flipped more than once are marked as two-sided.

occluded. To avoid self-shadowing, only voxels in front of the tangent plane at the ray's origin update the occlusion in the mask. For all visible voxels, which store eye samples, the photon energy is splatted to individual pixels associated with their eye samples. The photon energy contribution to an eye sample is weighted by the following factors (see Section 2):

- the density-estimation kernel-weight, which depends on the eye sample's distance to the ray,
- the cosine of the angle between photon ray and eye sample normal,
- the occlusion weight of the corresponding cell in the occlusion mask, which is linearly interpolated between the four nearest neighbor cells according to the eye sample's position in the voxel.

Note that all positions and distance values are represented in grid coordinates for ease of computation.

Finally, the radiance contribution of the photon ray to the pixels in the image is computed as in photon density estimation [Jen01] with the difference that the contribution of *one* photon is computed for *many* pixels at once (*splatting*) instead of computing the contribution of *many* photons to *one* pixel, which is refered to as *gathering*. The local surface BRDF associated with the eye sample is evaluated and the outcome is multiplied with the previously computed weights per eye sample and added to the corresponding pixel.

5 Constructing the Voxel Hierarchy

So far we have generated a regular grid for storing the approximate occlusion for the six axis-aligned directions in each voxel. As known from raytracing, a grid has the disadvantage that it cannot adapt to local scene complexity and does not scale well with its resolution. In our case it is even worse since we need a high-resolution grid to obtain images of good quality, which results in mostly empty voxels. Therefore, we build a hierarchy, more precisely a kd-tree, on top of the precomputed grid that merges all empty voxels during its recursive construction.

The kd-tree is build in a recursive top-down fashion and the axis-aligned splitting plane of a node is always aligned with the boundary of a voxel. We use a simple splitting heuristic to determine the discrete position of the splitting plane. The plane is either positioned at the spatial-median voxel for the largest dimension of the node's bounding box or at the closest non-empty voxel if either half-space is empty. Searching for empty space is expensive if we naively test each voxel in the current bounding box associated with a node. We therefore build a 3D summed area table (SAT) of the occlusion grid. The SAT is highly memory consuming because we need 32 bit precision instead of 8 bits per voxel due to large potential sums. However, it is only kept temporarily during the tree construction. The construction of the 3D SAT is done in linear time with respect to the grid size and is negligible compared to the rest of the computation. The computation for the best splitting plane according to our heuristic can then be computed in constant time independent of the grid resolution. The discretization of the tree construction increases not only the performance but also allows for higher compression of the size of the kd-tree nodes. Each node consists of only 8 bytes and there are four basic node types:

- Empty Nodes
- Splitting Nodes
- Occlusion Nodes
- Visible Nodes

Empty nodes represent empty space that can be skipped. Splitting nodes subdivide their associated bounding box into half-spaces. An occlusion node is created when either a voxel is reached (the smallest entity) or the occlusion in the sub-tree is uniform and further subdivision cannot yield new information. Occlusion nodes contain information about local occlusion (1 byte) and the average quantized normal compressed to 3 bytes, which is used to determine if occlusion is feasible. A visible node stores the index and the number of eye samples associated with the node and has always one child node. The outcome of the tree construction is an adaptive voxelization of the scene, which adapts to the scene complexity as opposed to the grid structure and has a small memory footprint even for high resolutions. Highest compression can be achieved for planar axis-aligned surfaces.

5.1 Efficient Traversal of the Voxel Hierarchy

Once the hierarchy is constructed and the photons are generated, each photon ray traverses the voxel hierarchy for searching eye samples and splatting energy to all eye samples found in the splatting footprint. Similar to raytracing with kd-trees, we recursively traverse the tree from the root to the leaves in near to far order. This requires maintaining a stack for caching the necessary information to traverse the far nodes after having processed the near nodes. In standard raytracing the recursive tree traversal works in 1D ray space and needs only three parameters: the minimum and maximum distance along the ray corresponding to the entry point and the exit point of the axis-aligned bounding box associated with a kd-tree node and the node's index. Since we deal with a volumetric ray, we need to keep track of the minimum and maximum bounds tightly encompassing the ray volume in three dimensions, see Fig. 5.4. The ray volume is represented by a cylinder, whose radius is defined by the photon's splat radius. At first glance a traversal in three dimensions seems to be too expensive compared with a simple 1D ray traversal. However, we can set two simplifying assumptions: first, the kd-tree and most parameters are discretized to the grid resolution and second, the kd-tree and its traversal are always axis-aligned, which simplifies many computations and allows us to precompute several parameters, which we describe next.

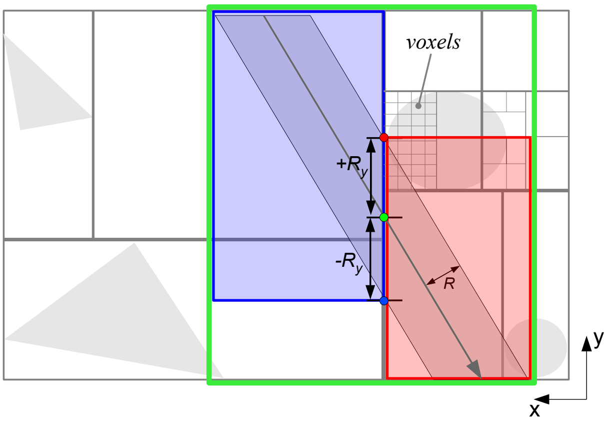


Fig. 5.4 – Voxelization Tree Traversal

One traversal step in 2D for an interior node in the kd-tree. The enlarged green rectangle marks the initial bounding box of the ray cylinder, the blue-shaded and the red-shaded area are the bounding box for the near child and far child, respectively. This recursive traversal stops when we reach an empty node or an occlusion leaf node, i.e., voxel.

Since our tree is axis-aligned, we restrict the recursive sub-division of the ray volume to axis-aligned bounding boxes starting with the whole bounding box of the ray volume. For the sub-divisions of the ray volume, we only need to consider cross-sections with the cylinder and the axis-aligned splitting planes of the kd-tree in order to compute the two new bounding boxes for the front and back side of a splitting node. The height and width of each cross-section in each dimension are precomputed by projecting the cylinder onto the three axis-aligned planes. For example the cross-section parameters in the yz-plane for the y-axis ($cutSize[X][Y]$) and z-axis ($cutSize[X][Z]$) are computed as

$$\begin{aligned} cutSize[X][Y] &= (R/L_{vox}) \cdot \sqrt{1.0-(\vec{\mathbf{D}}[Z])^2} / \left|\vec{\mathbf{D}}[Z]\right| \\ cutSize[X][Z] &= (R/L_{vox}) \cdot \sqrt{1.0-(\vec{\mathbf{D}}[Y])^2} / \left|\vec{\mathbf{D}}[Z]\right|, \end{aligned}$$

where R is the photon's splat radius, L_{vox} is the voxel length, and $\vec{\mathbf{D}}$ is the ray direction. The parameters of the other dimensions are computed analogously. Hence, the occlusion mask has elliptical shape and is axis-aligned with the cross-section that is perpendicular to the major traversal axis N corresponding to the largest component of $\vec{\mathbf{D}}$.

Having precomputed these values, we can quickly compute the new ray bounds for left and right side of a splitting node (see Fig. 5.4) for all three splitting axes.

6 Limitations of Discretized Occlusion

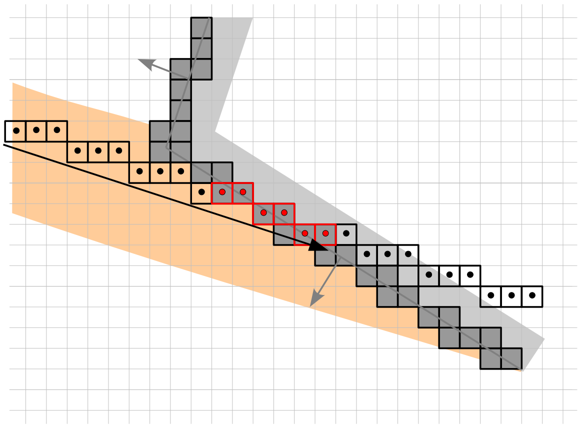


Fig. 5.5 – Self-Occlusion

Updating the occlusion (2D) in the axis-aligned occlusion mask can lead to erroneous self-occlusion on surface voxels (red point-squares) since many voxels, representing the surface, map to the same cell in the occlusion mask.

The voxel hierarchy is build in an axis-aligned manner. Therefore, it is most efficient for scenes which comprise many axis-aligned surfaces, such as walls of a building for example. Since the occlusion is entirely discretized and represented as voxels in the hierarchy, aliasing as it occurs in ray tracing is implicitly filtered. The occlusion per voxel is pre-integrated due to the rasterization preprocessing step. On the other hand, we face a different problem because of discretizing the visibility, which contains the highest frequencies in a visual signal. Restricting to opaque surfaces, occlusion of a ray happens exactly at one point along the ray. Hence the signal in ray space is 1 in front of this point and 0 behind. Since we do not regard infinitesimal thin rays but rather deal with discrete volumetric rays, many discrete occlusions can map to the same cell of the occlusion mask in ray space, which leads to self-occlusion on an actual flat surface. In Fig. 5.5 the high-lighted voxels (point-squares) show the state of one particular cell in the history of the occlusion mask. Depending on the incident angle between ray and voxelized surface (grey squares), many surface voxels map to the same cell in the occlusion mask.

This problem is also evident in all rasterization approaches as in the popular shadow mapping technique. One naïve solution is to constrain a discrete occlusion event to become valid only at a certain distance from the occluder, basically adding a small threshold to its distance when comparing it with the current surface distance. A constant threshold however only works for distant occlusion and may produce visible light leakage in case of near occlusions (see Fig. 5.7). A better solution is to compute the minimum occluder distance adaptively depending on the incident angle of the ray to the voxelized surface (distance from first to last red square in Fig. 5.5). Yet a simpler way is to update the occlusion only for back facing voxels, which fails for voxels containing front and back facing surfaces. In either way we need to know the surface normal in the voxel at the occlusion event.

6.1 Estimating the Surface Normal

Explicitly saving the normal for each rasterized fragment during the occlusion grid computation on the GPU is inefficient since we would loose the advantage of computing many slices of the grid at once. However, the approximate surface normal in a voxel is implicitly computed during the initial 3D rasterization process. Assuming that a voxel does not contain front and back facing surfaces simultaneously, the average quantized normal $\vec{\mathbf{n}}$ can be approximated from the six discrete front and back facing occlusion ratios $\vec{\mathbf{O}}_c$ stored in a voxel

$$\vec{\mathbf{n}}[i] \approx \begin{cases} \vec{\mathbf{O}}_c[i+3], & \vec{\mathbf{O}}_c[i] = 0 \wedge \vec{\mathbf{O}}_c[i+3] >= 0 \\ -\vec{\mathbf{O}}_c[i], & \vec{\mathbf{O}}_c[i] > 0 \wedge \vec{\mathbf{O}}_c[i+3] = 0, \end{cases} \tag{5.3}$$

for all $i \in [0,1,2]$. $\vec{\mathbf{O}}_c[i], \vec{\mathbf{O}}_c[i+3]$ is the ratio of occluded pixels to total number of pixels for the rasterized front-facing surface(s), back-facing surface(s) respectively inside the voxel projected to 2D onto the axis-aligned plane perpendicular to axis i, see Fig. 5.6.

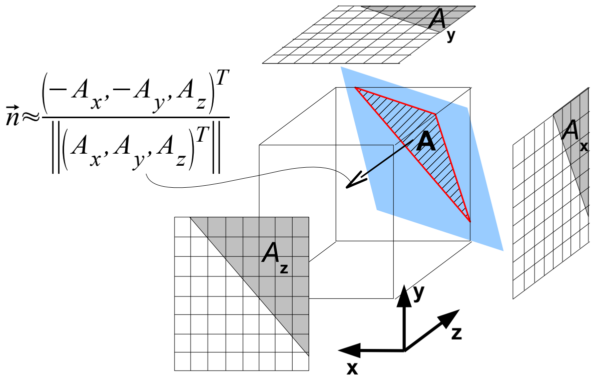

$\vec{n} \approx \frac{(-A_x, -A_y, A_z)^T}{\|(A_x, A_y, A_z)^T\|}$
A_y
A
A_x
A_z
y
z
x

Fig. 5.6 – Approximating the Surface Normal

Estimating the average surface normal in a voxel from the 3 axis-aligned projections A_x,A_y,A_z of a surface (blue) with clipped area A inside the voxel.

For the back facing test the normalization of $\vec{\mathbf{n}}$ can be avoided since we are only interested in the sign of the dot product between ray direction and $\vec{\mathbf{n}}$.

In the case of ambiguous voxels, which contain front and back facing triangles (i.e., $\vec{\mathbf{O}}_c$ is non-zero in at least 4 components), we do not reconstruct the normal. Instead we fall back to the simple solution of adding a small constant offset to the occluder distance before rendering the occlusion. However, the distance for the offset is measured in the dimension of the highest occlusion in the voxel, which corresponds to the largest component of a potential voxel normal. The largest dimension is precomputed during the voxel hierarchy construction and stored in the corresponding node. This heuristic yields slightly better results than adding only a constant offset to the ray distance.

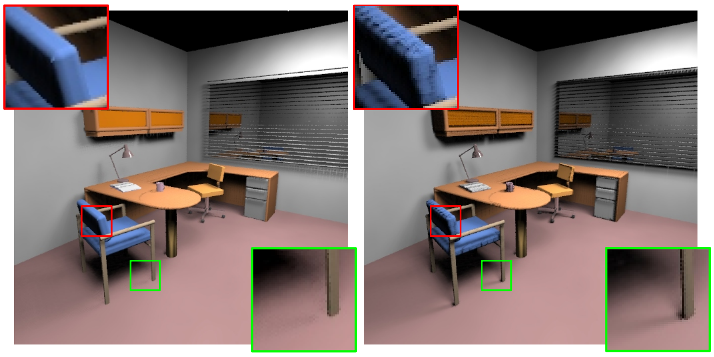


Fig. 5.7 – Occlusion bias

When adding a constant threshold to the occlusion distance, light leakage may occur if the threshold is too large (left). Contrarily, if the threshold is too small, erroneous self-occlusions appear (right).

7 Results

We have evaluated our method in comparison with instant radiosity combined with light cuts [WFA*05] using three scenes of different complexity and different lighting condition. The scene setting and the photon distribution is the same for both methods (except for the caustics). The resulting images are shown in Fig. 5.9. The rendering times and parameters are given in Table 5.1. All results were computed with an Athlon 64 X2 2.2 GHz Processor using only one core. The scene settings are: image resolution times the number of super-samples per pixel (i.e., 1 million primary rays for all images); total number of stored photons (M); and the percentage of stored direct, global indirect, and caustics photons. *Prim* is the number of geometric primitives in the scene. For our ray-splatting method the entries in the table show: (2. column) the grid resolution, (3. column) the memory for the occlusion tree plus the eye samples plus the photon rays, (4. column) the time in seconds to build the raytracing kd-tree and the occlusion hierarchy, (5. column) the time for the eye pass (shooting primary rays and storing eye samples), (6. column) the time for the light pass (photon

Scene	Method	Memory	Build	Eye	Light	Total
Cornell Box (81 % diffuse) Prim = 33	Ray-splatting 192×192×192	(5+26+6)	3.2	1.4	54	59
	Light-cuts $\bar{s}$=431	(14+5)	0.8	626	1.2	628
	Path-tracing (2500)	–	–	2720	–	2720
Scene settings	$500 \times 500 \times 4$; $M = 100,000$; (20%,50%,30%)					
Office (91 % diffuse) Prim = 34000	Ray-splatting 384×240×384	(13+23+3)	5.2	1.5	58	65
	Light-cuts $\bar{s}$=261	(8+3)	0.9	390	0.5	391
Scene settings	$500 \times 500 \times 4$; $M = 55,000$; (40%,60%,0%)					
Conference (86 % diffuse) Prim = 265880	Ray-splatting 560×368×160	(17+23+9)	10.6	3.4	98	112
	Light-cuts $\bar{s}$=321	(21+7)	5.6	680	3.4	689
Scene settings	$500 \times 500 \times 4$; $M = 150,000$; (35%,65%,0%)					

Table 5.1 – Numerical results for occlusion-ray-splatting

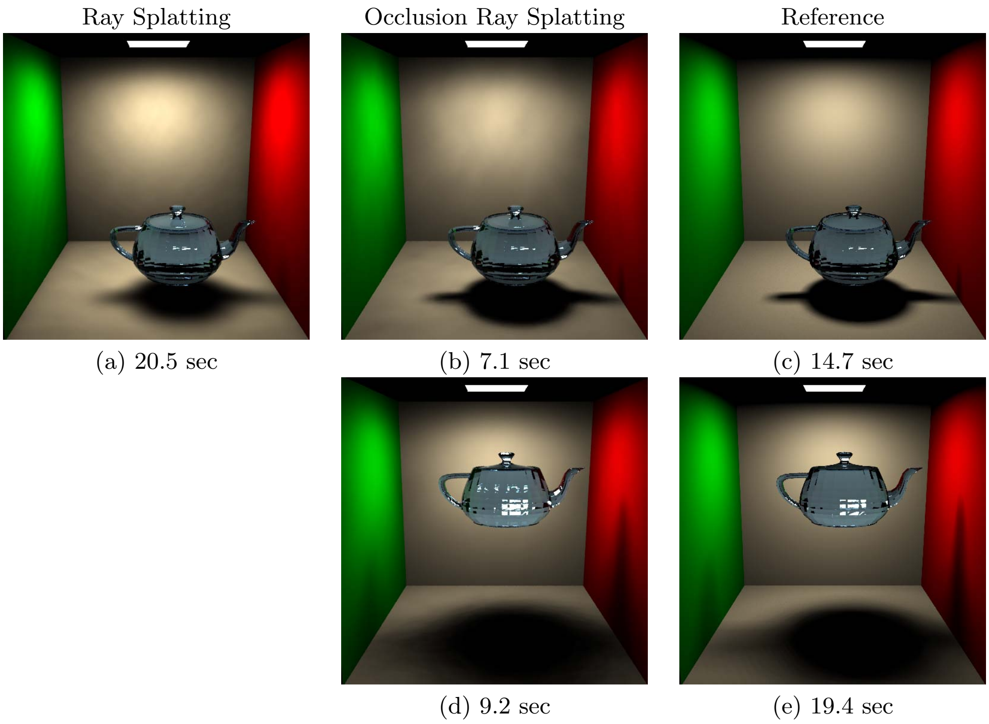
Computation times (in seconds) and memory consumption (in megabytes) for the rendering phases of our ray splatting method, instant radiosity with lightcuts, and path tracing. The computed images are shown in Fig. 5.9.

sampling and ray splatting), and (7. column) the total time needed to compute a single image.

For the light-cuts method the table entries represent: (2. column) the average number $\bar{s}$ of evaluated shadow rays per pixel sample with 2% error metric, (3. column) the memory used for the tree over light clusters [WFA*05] plus the photons (VPLs), (4. column) the time for constructing the raytracing kd-tree and clustering the VPLs, (5. column) the rendering time (computing the light cuts), (6. column) the time for the VPL sampling (which is the same in our method), and (7. column) the total time.

The cornell box scene contains a glass sphere and a glossy icosahedron and is rendered with global illumination. Since light-cuts is not able to generate caustics, we rendered this scene with unbiased *path tracing* [Kaj86] using 2500 samples per pixel. The office scene contains fine detailed geometry and requires a high resolution voxelization in order to capture all subtle shadow effects during ray splatting. In Fig. 5.9a-c the illumination in the office scene is decomposed into the direct, indirect, and global illumination. The reference is shown in Fig. 5.9d and was computed with light cuts using on average 261 shadow rays per pixel sample. The darkening in the corners of the light cut image is due to the automatic clamping of too high contributions of indirect VPLs. The conference scene is difficult to render with our method since it contains many wrongly facing or two-sided polygons as well as thin objects (e.g. the chairs around the table).

In Fig. 5.8 we show that our method is also capable of producing high quality direct lighting with soft shadows, while still using only a small number of photons and a grid resolution of 160^3 voxels.



(a) 20.5 sec (b) 7.1 sec (c) 14.7 sec (d) 9.2 sec (e) 19.4 sec

Fig. 5.8 – Comparison for direct lighting with occlusion-ray-splatting

The teapot scene rendered with direct illumination. Images (a), (b), and (d) were rendered with photon ray splatting with (a) using 80,000 *photons but ignoring visibility and (b), (d) using* 10,000 *photon splats with our voxelized occlusion. The image (c) and (e) are the ground-truth generated by Monte Carlo sampling of the area light with* 100 *and* 200 *shadow rays per pixel sample, respectively.*

8 Discussion and Future Work

Our method can be understood as a trade-off between instant radiosity [Kel97] and a direct visualization of the photon map [Jen01]. High-frequency surface normal variations are preserved through our novel splatting technique in ray space described in the previous Chapter. And high frequency shadows are handled via discrete occlusion testing in the voxelized scene. This way, we can even reproduce direct illumination with density estimation and achieve a fairly good reproduction of global illumination images generated from only a small number of photons (e.g. 50,000). Besides that, the density estimation becomes less sensitive to the bandwidth selection in contrast to traditional photon density estimation.

Nevertheless, our method also bears some limitations. Because the costs for processing and splatting a single photon are relatively high compared to traditional photon mapping [Jen01] and our photon ray splatting described in the previous chapter, the computation of fine illumination details (such as caustics), which re-

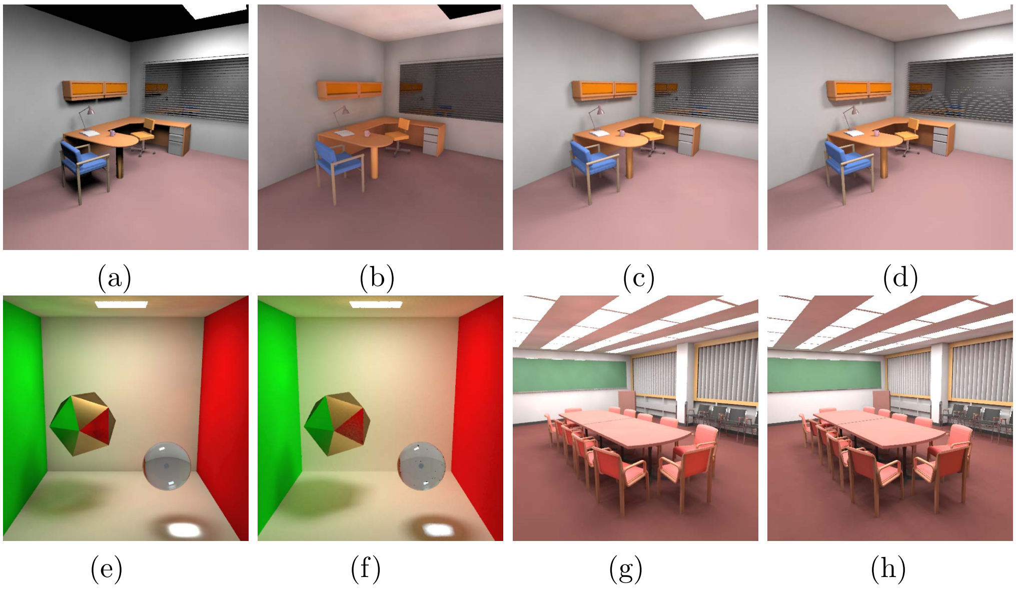


Fig. 5.9 – Visual results for occlusion-ray-splatting

Results for rendering times given in Table 5.1, first row: (a) office scene with direct, (b) indirect (1 f-stop brighter), (c) global illumination rendered with our method, and (d) with light cuts [WFA 05] as reference. Second row: modified cornell box with caustics and glossy illumination rendered with our method (e), and path tracing (f), and the conference scene with global illumination rendered with our method (g), and with light cuts (h).*

quire a high photon density, becomes computationally expensive. Another weak point is the scene voxelization, which is not adaptive to the camera view. This can reveal staircase artifacts for close-up views of shadows similarly as in shadow mapping. Further, artifacts can occur for badly modeled scenes that contain holes, or self-intersecting objects. Furthermore, the occlusion computation is view-independent in the sense that many occlusion computations are wasted in invisible regions of the scene. Using a conservative bidirectional traversal in the occlusion tree, we could prune unnecessary traversal steps at the lower levels of the tree if they are not contributing to the image or if they are fully occluded.

The voxel data-structure naturally fits to a multi-resolution representation (e.g. 3D Haar-Wavelets) of the scene as opposed to a polygonal mesh representation with level of detail. Photon rays with small contribution, for example indirect rays, could have a higher error tolerance when traversing the voxel hierarchy.

And last, we would like to map the entire splatting algorithm in a simplified form including the voxel hierarchy construction to the GPU. It would be particularly interesting to see whether we can get by with our approximative scene representation for the entire global lighting computation including photon path-tracing. Because of the (low-pass) density estimation small errors in surface orientation and location should be more forgivable than for primary eye-raytracing.

Anisotropic Radiance Cache Splatting

1 Introduction and Overview

While in previous chapters we have dealt with approximative algorithm for solving the global illumination, in this chapter we will focus on computing high quality solutions to the global illumination problem in synthetic scenes. Although our proposed method is still based on biased algorithms, as unbiased algorithms converge very slowly, exhibit high-frequency noise and are rarely used in industry, we obtain results which are hardly distinguishable from unbiased results but are orders of magnitude more efficient. The common basis of biased approaches is to exploit the coherence in the lighting by caching the flux distribution in the scene by means of photon hits and reusing it for all pixels. Two well-known method of this variant are instant radiosity [Kel97] and photon mapping [Jen01]. However, even when precomputing a fixed set of light samples, computing the illumination on a pixel basis is still too costly in particular for indirect lighting. Therefore, a common approach to speed up the computation of indirect lighting further is to use irradiance caching [WRC88, TL04, KGPB05]. Irradiance caching is a well-established algorithm widely used in production rendering and lighting simulation software. The basic idea is to exploit the "smoothness" of the indirect light by sparse sampling and interpolating the irradiance in image or object space and thus reducing the intensive computation for the indirect light to only a few thousand point samples as opposed to millions. However, pure irradiance caching works only for diffuse surfaces. An extension to radiance caching on moderately glossy surfaces using (hemi-)spherical harmonics has been published in [KGPB05] and [KBPv06], which also proposes an adaptive, perceptually-inspired sampling scheme and an improved gradient-based interpolation of the cache samples. Nevertheless, those caching algorithms are difficult to control and producing artifact-free images often requires initial test-runs and user experience for setting the appropriate parameters. Little work has been devoted to the topic of how to adapt the computation of the cache records to the local lighting and scene complexity and also on how to reduce the space of user parameters.

In this chapter, we propose a method that advances in this direction. Our radiance and irradiance caching algorithm is not only one to two orders of magnitude more efficient than traditional irradiance caching but also more robust as it reduces the burden of choosing the "right" rendering parameters by the user. The performance gain is two-fold. First, we obtain a speedup by reversing the caching procedure and second, we use the lightcuts algorithm [WFA*05, WABG06] extended by our (ir)radiance gradients [WH92] for more efficient computation of the cache records. The traditional gathering scheme for irradiance caching [WRC88], where a search for feasible cache records is initiated at each pixel sample, imposes restrictions on the traversal order of the pixels. It has been proposed at a time when main memory was scarce and is not appropriate nowadays. It also does not scale well with the trend of increasing image resolution. We perform cache splatting in 3D object space, where each newly computed cache record directly extrapolates its (ir)radiance contribution to all neighboring eye samples in its ellipsoidal splatting footprint.

Using the lightcuts technique we are able to compute indirect and direct light in one pass without the need for user parameter settings like number of final gather rays (FGR) per pixel, k-nearest neighbor photons etc., as in the traditional photon mapping with (ir)radiance caching procedure. Lightcuts automatically adapts to the scene and lighting complexity and furthermore, when using our perceptually-derived visibility thresholds, to the perceptual cues in the image, which often counterbalances scene complexity (rendering errors in a complex scene with many geometric details and textures are less perceptible by the human eye than in simple scenes.) The latter case is particularly important as it allows us to relax the rendering errors and reduce computation time in cluttered image regions.

In order to compute direct and indirect light faithfully using our caching technique, we maintain a "two-layer" irradiance cache, which caches and interpolates direct and indirect light contributions in parallel. However, both cache layers interact with each other on-the-fly in terms of luminance masking to steer the cache interpolation and error thresholds in the lightcuts computation, which is impossible if we compute direct and indirect lighting separately.

Finally, we improve the (ir)radiance interpolation by using anisotropic cache splatting, which reduces the overall number of cache records per frame while maintaining the same image quality.

2 Lightcuts

Lightcuts is a deterministic and hierarchical algorithm based on *instant radiosity* [Kel97]. It is one to two orders of magnitude more efficient than photon mapping because it neither requires a costly search for the nearest photon(s) nor does it involve stochastic ray shooting for Monte Carlo final gathering. Furthermore, lightcuts' adaptive nature inherently performs "importance sampling"[1] for the product term of BRDF and lighting and not only for the BRDF term as in photon mapping [Jen01]. Instead of uniform sampling the radiance in the upper hemisphere with a constant number of final gather rays, a small set of clusters over virtual point lights (VPLs) is chosen adaptively for each eye sample point $\mathbf{x}$ such that the error introduced by the clustering is invisible for each pixel.

Let's first derive how we compute the pixel radiance contribution L from a subset of diffuse surfaces S_C in the scene for a point $\mathbf{x}$ using *instant radiosity*

$$L(\mathbf{x},\omega) = \int_{S_C} f_s(\mathbf{x},\mathbf{y},\omega)\cdot G(\mathbf{x},\mathbf{y})\cdot V(\mathbf{x},\mathbf{y})\cdot L_e(\mathbf{y}\rightarrow\mathbf{x})\cdot dA_y, \qquad (6.1)$$

where the outgoing radiance L_e at point $\mathbf{y}$ is assumed to be independent of direction, i.e., $f_s(\mathbf{y}_{-1}\rightarrow\mathbf{y}\rightarrow\mathbf{x}) = f_s(\mathbf{y})$, and can be precomputed as

$$L_e(\mathbf{y}\rightarrow\mathbf{x}) := L_e(\mathbf{y}) = f_s(\mathbf{y})\cdot E(\mathbf{y}) = \frac{\rho_d(\mathbf{y})}{\pi}\cdot\frac{d\Phi(\mathbf{y})}{dA_y}. \qquad (6.2)$$

Inserting Eq. 6.2 in Eq. 6.1 this leads to the instant radiosity algorithm when replacing the integral by a finite sum over radiant flux $\Delta\Phi$ from all N_C surfaces elements in S_C

$$L(\mathbf{x},\omega) \approx \sum_i^{N_C} f_s(\mathbf{x},\mathbf{y}_i,\omega)\cdot G(\mathbf{x},\mathbf{y}_i)\cdot V(\mathbf{x},\mathbf{y}_i)\cdot\left(\frac{\rho_d(\mathbf{y}_i)}{\pi}\Delta\Phi(\mathbf{y}_i)\right), \qquad (6.3)$$

where the last term is the precomputed intensity $I_i := I(\mathbf{y}_i)$ stored with each VPL.

The computation in Eq. 6.3 converges with increasing N_C but is not adaptive and the costs per VPL are very high. Therefore, we transform Eq. 6.3 into a hierarchical computation. We compute clusters of VPLs, where a cluster sums the intensity $\sum_i^{N_C} I_i$ of all N_C VPLs in its support but uses the form factor $G_C\cdot V_C$ and BRDF f_C of one representative VPL with index C:

$$\begin{aligned} L_C(\mathbf{x},\omega) &= \sum_i^{N_C} f_s(\mathbf{x},\mathbf{y}_i,\omega)\cdot G(\mathbf{x},\mathbf{y}_i)\cdot V(\mathbf{x},\mathbf{y}_i)\cdot I_i \\ L_C(\mathbf{x},\omega) &\approx f_s(\mathbf{x},\mathbf{y}_C,\omega)\cdot G(\mathbf{x},\mathbf{y}_C)\cdot V(\mathbf{x},\mathbf{y}_C)\cdot\sum_i^{N_C} I_i. \qquad (6.4)\end{aligned}$$

[1] The notion of *importance sampling* is not adequate here as we will cope with a completely deterministic approach

Certainly computing an optimal clustering for each individual eye sample is way too expensive (in fact it is a NP hard problem). Therefore, in order to make the clustering adaptive and efficient, VPL-clusters are precomputed hierarchically in a binary tree (the light tree), where individual VPLs are stored in the leaf nodes and interior nodes cluster the intensities of all VPLs in their sub-trees. The light tree is kept static and reused for the lighting computation at each eye sample. To obtain a pixel-adaptive set of clusters, we then solely compute a cut through the light tree, giving the algorithm its name *lightcut*. A valid lightcut is a set of nodes such that for every path from a leaf node (VPL) to the root node of the tree there exists one and only one node in the lightcut.

Lightcuts are computed as in [WFA*05] except that we also compute gradients (Section 3.3) and correct for the clamping of VPLs (Section 5) for each cache record. Since the original paper [WFA*05] does not specify explicitly how to compute and refine the lightcut, we present the basic algorithm for computing the outgoing pixel radiance with lightcuts in Algorithm 3. The basic algorithm initializes the lightcut with one cluster, the root node, and iteratively refines it by traversing down the tree while always replacing one cluster node in the current lightcut with its two children if the error introduced by this cluster is too large. Computing the exact error introduced for one cluster in Eq. 6.4 would require evaluating the entire sub-tree (primarily computing the visibility) and is suboptimal. Therefore, the basic idea is to compute inexpensive upper bounds for the error of a cluster with regard to a particular eye sample, which boils down to computing separate upper bounds of the individual terms we are approximating in Eq. 6.4. We omit an introduction to the essential computation of the upper bounds for the geometric term G_{UB} and the BRDF f_{UB} in lightcuts and refer the reader to [WFA*05] for details. The most expensive term in Eq. 6.4 is the visibility term for which it is hard to make conservative assumptions other than the trivial one $V_{UB} := 1$.

Once having computed upper bounds for BRDF and geometric term of a cluster we can compute an upper bound for the absolute error Δ_ε introduced when computing the radiance using this cluster. Note that for computing an upper bound for the absolute error we also have to consider the lower bound of the error. Fortunately, the lower bounds of the approximated terms in Eq. 6.4 are trivially zero since $V_{LB} := 0$.

1. Lower bound: all $N-1$ non-evaluated VPLs in the sub-tree of the cluster have zero contribution, i.e. $f_{LB} \cdot G_{LB} \cdot V_{LB} = f_{LB} \cdot G_{LB} \cdot 0$

$$\begin{aligned} \Delta_{LB} &= \left(0 \cdot \left(\sum_i^N I_i - I_C\right) + f_C \cdot G_C \cdot V_C \cdot I_C\right) - f_C \cdot G_C \cdot V_C \cdot \sum_i^N I_i \\ &= -(f_C \cdot G_C \cdot V_C) \cdot \left(\sum_i^N I_i - I_C\right). \end{aligned} \tag{6.5}$$

2. Upper bound: all $N-1$ non-evaluated VPLs in the sub-tree of the cluster

have maximum possible contribution $f_{UB} \cdot G_{UB} \cdot V_{UB} = f_{UB} \cdot G_{UB} \cdot 1$ within the computed bounds

$$\begin{aligned}\Delta_{UB} &= \left(f_{UB} \cdot G_{UB} \cdot \left(\sum_i^N I_i - I_C \right) + f_C \cdot G_C \cdot V_C \cdot I_C \right) - f_C \cdot G_C \cdot V_C \cdot \sum_i^N I_i \\ &= \left(f_{UB} \cdot G_{UB} - f_C \cdot G_C \cdot V_C \right) \cdot \left(\sum_i^N I_i - I_C \right). \qquad (6.6)\end{aligned}$$

Hence, an upper bound for the absolute error of a cluster is taken as the maximum over the norms of Eq. 6.5 and Eq. 6.6

$$\Delta_\varepsilon := \max(\|\Delta_{LB}\|, \|\Delta_{UB}\|). \qquad (6.7)$$

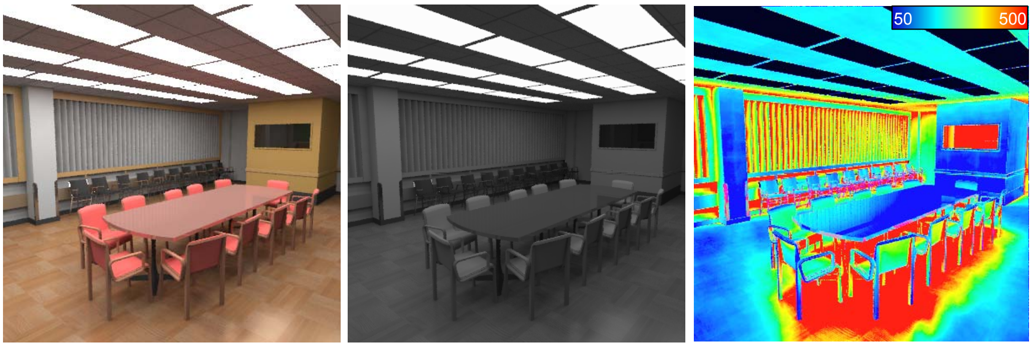
Algorithm 3 Lightcut Computation

```
Input:
    $\varepsilon_L$                lightcut refinement threshold (e.g. 1%)
    MAX_CUT_SIZE      the user defined maximum allowed lightcut size
    lightcut          priority queue (heap) sorting clusters with error
    lightTree         the binary tree of VPL clusters

Output:
    $L_{out}$              outgoing pixel radiance

$L_{out}$ := GetRadianceNotToBeComputedWithLightcuts() /* e.g., $L_e$, caustics */
cutItem.node := lightTree.GetRootNode()
/* for simplicity form factor also includes the BRDF term here */
cutItem.formFactor := ComputeClusterPotential(cutItem.node.GetVPL())
cutItem.error := MAX_ERROR
$L_{out}$ := $L_{out}$+ cutItem.formFactor * cutItem.node.intensitySum
$N_L := 1$
/* also avoid too small lightcuts due to overestimated $L_{out}$ */
while (cutItem.error > $\varepsilon_L$ * Luminance($L_{out}$) OR $N_L$ < MIN_CUT_SIZE) do
  if $N_L \geq$ MAX_CUT_SIZE then
    break /* stop lightcut refinement if its size exceeds imposed limits */
  end if
  cluster := cutItem.node
  $L_0$ := cutItem.formFactor * cluster.intensitySum
  cutItem.node := cluster.RepresentativeChild()
  upperBoundFormFactor := ComputeUpperBounds(cutItem.node)
  /* Same VPL for the representative child node → same form factor */
  cutItem.error := ComputeError(upperBoundFormFactor, cutItem.formFactor, cutItem.node)
  $L_r$ := cutItem.formFactor * cutItem.node.intensitySum
  lightcut.push(cutItem)
  cutItem.node := cluster.AdoptedChild()
  upperBoundFormFactor := ComputeUpperBounds(cutItem.node)
  /* Need to compute new form factor with visibility for adopted child node */
  cutItem.formFactor := ComputeClusterPotential(cutItem.node.GetVPL())
  cutItem.error := ComputeError(upperBoundFormFactor, cutItem.formFactor, cutItem.node)
  $L_a$ := cutItem.formFactor * cutItem.node.intensitySum
  lightcut.push(cutItem)
  $L_{out} := L_{out} - L_0 + L_r + L_a$ /* Refine the result */
  /* Get lightcut-entry with highest upper error bound and remove from queue */
  cutItem := lightcut.topAndPop();
  $N_L := N_L + 1$
end while
```

The final algorithm shown in Algorithm 3 can be summarized as follows. Initially, the light distribution in a scene is hierarchically stored in a binary tree, where each node stores the summed intensity, the angle of the surface-normal bounding-cone and the bounding-box over all VPLs in the sub-tree. During rendering the light-tree is adaptively traversed in a top-down fashion and at each visited cluster node a conservative upper error bound is computed and the node is stored in a priority queue (the lightcut), which sorts the cluster according to its error. In each iteration we take the cluster with the maximum error from the queue and refine it by computing the radiance contributions and the error bounds of its children. The process repeats until all nodes in the priority queue have an error below a threshold ε_L relative to the pixel luminance or the size of lightcut N_L exceeds a maximum threshold. For our cache computation we set $\varepsilon_L = 1\%$ and $N_L = 1000$. As example for the pixel-adaptive rendering with lightcuts see Fig. 6.1.



(a) GI result (b) error thresholds (ε_L) (c) lightcut size (N_L)

Fig. 6.1 – Pixel-adaptive rendering with lightcuts

Rendering result for the lightcuts algorithm (a) with 2% pixel error thresholds shown in (b) scaled by factor 16. The color-encoded number of evaluated point lights per pixel (avg. 281 lights) is visualized in (c).

Computing the upper error bounds for all traversed nodes is costly. We therefore refine a cluster immediately (add it to the top of the priority queue) if its energy contribution to the pixel luminance is above a threshold (we use 10%). This helps to skip needless computations of the upper error bounds for clusters in the upper part of the light-tree, which are later refined anyway.

Since the lightcuts algorithm scales well with the number of virtual point lights (VPLs), we can afford a huge number of VPLs (e.g. $N > 10^6$). Therefore, in contrast to *instant radiosity* [Kel97], the clamping of the energy contribution of close VPLs is much less objectionable in lightcuts. Such clamping also wrongly estimates gradients near corners, which is undesirable as we adapt our cache record density and the cache splatting footprint to the gradient. Nevertheless, we clamp the maximum contribution of a VPL in the computed lightcut at 1% of the pixel's luminance value to suppress the otherwise visible noise in corners. The same applies also to the gradients of the VPL. This clamping introduces bias (missing energy) but an efficient remedy is proposed in Section 5.

3 Irradiance and Radiance Caching

The original lightcuts algorithm estimates the rendering equation for each pixel sample or adaptively for at least every fourth pixel where in-between pixel samples are interpolated in image space (reconstruction cuts [WFA*05]). However, such a caching strategy of directly visible pixel radiance is only well-suited for rendering single frames and is not efficient for walk-through animations because the view-dependent BRDF term is often the source of high-frequency temporal changes. Moreover, the image-space block-interpolation for the reconstruction cuts [WFA*05] does not consider temporal flickering artifacts and has problem with reproducing the shape of glossy highlights.

Our method extends the originally proposed lightcuts algorithm [WFA*05] with (ir)radiance caching in object space [WRC88, KGPB05] for view-independent (ir)radiance caching. As in [KGPB05] the incident radiance and BRDF/material term (f_s) are separately projected onto the spherical (SH) or hemispherical harmonics [GKPB04] (HSH) basis $H_l^m(\theta,\phi)$ for the ease of interpolation on glossy surfaces. One could argue that projecting the lightcut result to a low-frequency spherical basis and throwing away all the detailed information gathered in the lightcut is not only lossy but somewhat redundant since both, the lightcut and the SH radiance coefficients, capture the incident radiance field. However, the radiance field captured by the lightcut is highly spatial coherent and storing the whole lightcut per cache record is not only wasteful in terms of memory but would also require an expensive re-projection to all eye samples within the cache footprint. Such reprojection is well approximated with our proposed gradients. For most surfaces of global illumination scenes, even when consisting of view-dependent surfaces, a (filtered) low-frequency representation of the directional radiance field is sufficient. Besides indirect illumination on high-frequency glossy surfaces is anyway badly reproduced when using the lightcuts algorithm and a Monte Carlo sampling strategy of the BRDF is more advisable and efficient in such cases.

In the following we will only refer to spherical harmonics (SH) as the basis for our computations. However, hemispherical harmonics (HSH) have very similar properties as spherical harmonics and can be treated almost interchangeable (see [GKPB04]) [2]. Although irradiance caching on diffuse surfaces can be treated the same way as radiance caching, we separate irradiance caching from radiance

[2] For hemispherical harmonics the rotation of coefficients around the Y-axis requires small changes. We also have to (pre)compute the “cut-off matrices” for a set of discrete rotation angles that delete the radiance contribution of the digon falling below the surface tangent plane after rotation [GKPB04]

caching in the way that irradiance cache records store only view-independent irradiance values whereas radiance cache records capture the full incident radiance field. The reason for this separation is performance and memory efficiency. Irradiance cache records need to store only scalar values whereas radiance cache records store for each color channel a number of SH coefficients for the radiance and radiance gradient as well. For example, not including the cosine term in the computation of the irradiance on diffuse surfaces requires to store at least 9 SH coefficients [RH04] for the incident radiance for each diffuse irradiance cache record, which is additional overhead if we assume most scenes are dominated by diffuse surfaces. Furthermore, because the alignment of the local coordinate frames is less important for view-independent diffuse surfaces, storing the cosine-projected irradiance instead of the radiance coefficients we can avoid expensive SH rotations. On the other hand, if we ignore the rotation of the radiance field on diffuse surfaces, we introduce a considerable error for curved surfaces where the change in irradiance is mainly due to the change in surface normals. Hence, following [WH92] we compute a rotational irradiance gradient (see Section 3.3.2) for all irradiance cache records, which can be computed more efficiently than a SH rotation. For all radiance cache records we exclude the cosine term in the cache computation, which instead is included in the BRDF SH coefficients, and perform a fast approximate SH rotation [KŎ5b] in replacement of a rotational gradient.

3.1 Irradiance Caching

Irradiance is cached and interpolated in object space on all view-independent diffuse surfaces. We follow Ward and Heckbert [WH92] and store the irradiance along with irradiance gradients at a sparse set of eye samples referred to as cache records and interpolate in-between samples. The 5-dimensional irradiance gradient (3D translation + 2D surface rotation) is split into a translational gradient ∇_t in the 2D tangent plane and a rotational gradient ∇_r. The irradiance $E(\mathbf{x})$ is computed with lightcuts as

$$E(\mathbf{x}) = \sum_{i=1}^{N_L} G(\mathbf{x},\mathbf{y}_i) \cdot V(\mathbf{x},\mathbf{y}_i) \cdot I_i, \tag{6.8}$$

where N_L is the size of the lightcut, $G(\mathbf{x},\mathbf{y})$ is the geometric term, $V(\mathbf{x},\mathbf{y})$ the binary visibility function, and I_i the intensity of the cluster i in the lightcut. The computation of the gradients is described in Section 3.3. The irradiance $E(\mathbf{p})$ for an eye sample is computed as a weighted sum of the extrapolated irradiance from the K nearest cache records with spatially overlapping footprints at $\mathbf{p}$

$$E(\mathbf{p}) \approx \sum_{k=1}^{K} w_k(\mathbf{p}) \cdot \{E(\mathbf{x}_k) + (\mathbf{p} - \mathbf{x}_k) \cdot \nabla_t E_k + (\vec{\mathbf{n}} \times \vec{\mathbf{n}}_k) \cdot \nabla_r E_k\}, \tag{6.9}$$

with $\sum_{k=1}^{K} w_k(\mathbf{p}) = 1$.

The cache interpolation requires to find all contributing cache records, whose performance depends on the search efficiency. We can distinguish two search strategies:

1. traditional single pass: search all over-lapping cache records for each eye sample (cache gathering)
2. two pass: search all eye samples in the footprint of each cache record (cache splatting)

We favor a variant of the second strategy as it is algorithmically more efficient in particular for higher resolution images. Our cache splatting is described in Section 4.

3.2 Radiance Caching

As opposed to irradiance caching a radiance cache record captures the (filtered) incident radiance field for each color channel in a few SH coefficients. The radiance SH coefficients λ_l^m are also computed with lightcuts:

$$\lambda_l^m(\mathbf{x}) = \sum_{i=1}^{N_L} G^{\perp}(\mathbf{x},\mathbf{y}_i) \cdot V(\mathbf{x},\mathbf{y}_i) \cdot I_i \cdot Y_l^m(\theta_i,\phi_i), \tag{6.10}$$

where m is the degree and l is the band of the SH coefficient and $G^{\perp}$ is the unprojected geometric term

$$G^{\perp}(\mathbf{x},\mathbf{y}) = \frac{\cos\theta_y}{||\mathbf{x}-\mathbf{y}||^2}, \tag{6.11}$$

since we only want to capture the incident radiance field, which can be rotated later on.

The cosine-weighted BRDF is also projected onto the SH basis:

$$f_l^m(\mathbf{x},\omega_o) = \frac{2\pi}{N \cdot M} \sum_{j=0}^{M-1} \sum_{k=0}^{N-1} f_s(\mathbf{x},\omega_{j,k},\omega_o) \cdot \cos\theta_{j,k} \cdot Y_l^m(\theta_{j,k},\phi_{j,k}), \tag{6.12}$$

where $f_l^m(\mathbf{x},\omega_o)$ are the BRDF SH coefficients precomputed for a number of discretized local viewing directions ω_o for each BRDF in the scene model [KGPB05]. Note that we separate the view-dependent BRDF from the Material. Therefore, the BRDF SH coefficient only capture the cosine-weighted gloss lobe of the BRDF and not the diffuse and specular albedo of the BRDF as this can be modulated by high-frequency spatially varying textures. Further, in our implementation the BRDF is assumed to be independent of the color channel, which allows us to store only one scalar value per BRDF SH coefficient.

Since the SH basis is orthonormal, the outgoing radiance at any point on a surface is computed as the dot product of the radiance SH coefficient vector $\Lambda(\mathbf{x})$ and the BRDF SH coefficient vector $F(\mathbf{x}, \omega_o)$.

$$\begin{aligned} L(\mathbf{x}, \omega_o) &= \sum_{l=0}^{n-1} \sum_{m=-l}^{l} \lambda_l^m(\mathbf{x}) \cdot f_l^m(\mathbf{x}, \omega_o), & (6.13) \\ &= \Lambda(\mathbf{x}) \bullet F(\mathbf{x}, \omega_o), & (6.14) \end{aligned}$$

where n is the number of bands (we use up to 8) depending on the "glossiness" of the surface [KGPB05].

With (ir)radiance caching, one cache sample contributes to many neighboring eye samples. Thus the pixel radiance L at a point $\mathbf{p}$ becomes the weighted average of the radiance SH coefficients λ_l^m from the K neighboring cache samples modulated by the BRDF SH coefficients at the point $\mathbf{p}$

$$L(\mathbf{p}, \omega_o) \approx \sum_{k=1}^{K} \left(w_k(\mathbf{p}) \cdot \tilde{\Lambda}_k(\mathbf{p}) \right) \bullet F(\mathbf{p}, \omega_o), \tag{6.15}$$

with $\sum_{k=1}^{K} w_k(\mathbf{p}) = 1$. The extrapolated radiance SH coefficients of a cache record at $\mathbf{x}$ to a point $\mathbf{p}$ are computed similar to irradiance caching except that we exclude the rotational gradient and rotate the radiance field instead

$$\tilde{\Lambda}(\mathbf{p}) = \mathcal{R}\left(\Lambda(\mathbf{x}) + d_u \frac{\partial \Lambda(\mathbf{x})}{\partial u} + d_v \frac{\partial \Lambda(\mathbf{x})}{\partial v} \right), \tag{6.16}$$

where $\mathcal{R}$ is the SH rotation matrix that aligns the local coordinate frame at $\mathbf{x}$ with the coordinate frame at $\mathbf{p}$. The translational derivative vectors $\frac{\partial \Lambda(\mathbf{x})}{\partial u}$ and $\frac{\partial \Lambda(\mathbf{x})}{\partial v}$ represent the computed SH gradient in the local tangent plane of the cache record and $d_u = p_u - x_u$, $d_v = p_v - x_v$ is the difference vector $\mathbf{p} - \mathbf{x}$ projected into the local coordinate frame $(\vec{\mathbf{u}}, \vec{\mathbf{v}}, \vec{\mathbf{n}})$ at $\mathbf{x}$.

3.3 Irradiance and Radiance Gradients

(Ir)radiance extrapolation with gradients corresponds to a first order Taylor approximation of the actual (ir)radiance at the extrapolation point. Radiance caching without gradients can result in highly visible artifacts as shown in Fig. 6.2. However, lighting is not linear and error is still introduced this way in particular at discontinuities due to visibility changes. Including higher order terms would require to compute and store more information (e.g. the Hessian) for each cache record. Moreover, the error is usually reduced if more than one cache record contribute to the interpolation in Eq. 6.9, which is achieved with our cache splatting in Section 4. Therefore, (ir)radiance caching with gradients can be regarded as a good trade-off between efficiency and error. In contrast to final gathering with integration over solid angle, lightcuts solves the integral by sampling the scene's surface-area. Therefore, a stratified gradient computation scheme as proposed by [WH92, KGPB05] is difficult to achieve. We therefore focus on computing the radiance gradients analytically.

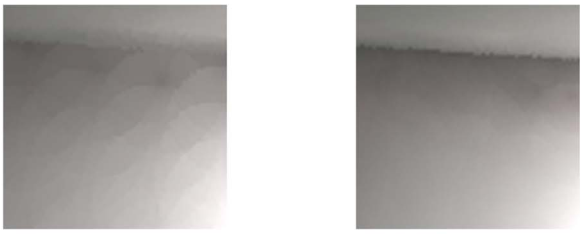


Fig. 6.2 – Irradiance Extrapolation with/without Gradients

Close-up of cache extrapolation without gradients (left), and with our gradients (right).

We choose to use simple irradiance caching with translational and rotational gradients on Lambertian diffuse surfaces [WH92] and translational gradients plus a SH rotation on glossy surfaces similar to [K0̌5a]. The gradients in the local coordinate frame defined by $\vec{\mathbf{u}}$, $\vec{\mathbf{v}}$, and $\vec{\mathbf{n}}$ are computed during the lightcut computation. As in [KGPB05] we ignore the displacement along $\vec{\mathbf{n}}$ since the cache records lie only on surfaces and the displacement along the surface normal $\vec{\mathbf{n}}$ is usually very small.

3.3.1 Translational Gradient

The translational irradiance gradient is computed by differentiating Eq. 6.8 with respect to u and v corresponding to the displacement in the tangent plane along unit-vectors $\vec{\mathbf{u}}$ and $\vec{\mathbf{v}}$ respectively

$$\frac{\partial E}{\partial u}(\mathbf{x}) = \sum_{i=1}^{N_L} I_i \cdot \frac{\partial}{\partial u}\{G(\mathbf{x},\mathbf{y}_i) \cdot V(\mathbf{x},\mathbf{y}_i)\}, \tag{6.17}$$

and in the same way for the derivative with respect to v along direction $\vec{\mathbf{v}}$.

Similarly, the translational SH radiance gradient is computed by differentiating Eq. 6.10

$$\frac{\partial \lambda_l^m}{\partial u}(\mathbf{x}) = \sum_{i=1}^{N_L} I_i \cdot \left[\frac{\partial}{\partial u}\{G^{\perp}(\mathbf{x},\mathbf{y}_i) \cdot V(\mathbf{x},\mathbf{y}_i)\} \cdot Y_l^m(\theta_i,\phi_i) + G^{\perp}(\mathbf{x},\mathbf{y}_i) \cdot V(\mathbf{x},\mathbf{y}_i) \cdot \frac{\partial Y_l^m}{\partial u}(\theta_i,\phi_i)\right]. \tag{6.18}$$

The derivatives of the SH (and HSH) basis functions $\frac{\partial Y_l^m}{\partial u}(\theta_i,\phi_i)$ are given in [K0̌5b]. Note that we ignore the visibility gradient in Eq. 6.17 and Eq. 6.18 and assume visibility is constant within the cache footprint at $\mathbf{x}$.

For simplicity we only regard a single VPL with index i and a single SH coefficient (we omit the indices of the SH coefficients) in the following equations. Note that for identification we occasionally use variable names in the subscript of parameters, which should not be confused with partial derivatives, i.e., $\theta_x \neq \frac{\partial \theta}{\partial x}$.

The gradient of the geometric term G expressed in the local coordinate frame $(\vec{\mathbf{u}},\vec{\mathbf{v}},\vec{\mathbf{n}})$ at $\mathbf{x}$ can be rearranged as follows [KGPB05] taking into account that the change of G with the displacement of $\mathbf{x}$ is opposite to its change with the displacement of $\mathbf{y}$ (see Fig. 6.3):

$$\begin{aligned}\frac{\partial G}{\partial u}(\mathbf{x},\mathbf{y}) &= -\frac{\partial G}{\partial q_u}(\mathbf{x},\mathbf{y}) \\ &= -\frac{\partial}{\partial q_u}\left\{\frac{\cos\theta_x \cos\theta_y}{||\vec{\mathbf{q}}||^2}\right\}, \end{aligned} \tag{6.19}$$

where we replace

$$\cos\theta_x = q_n/||\vec{\mathbf{q}}||, \tag{6.20}$$

$$\cos\theta_y = -(\vec{\mathbf{n}}_y \bullet \vec{\mathbf{q}})/||\vec{\mathbf{q}}||, \tag{6.21}$$

with the surface normal $\vec{\mathbf{n}}_y$ at $\mathbf{y}$ and $\vec{\mathbf{q}} = \mathbf{y} - \mathbf{x}$ expressed in the local coordinate frame of $\mathbf{x}$, i.e. $(q_u, q_v, q_n)^T = (\vec{\mathbf{u}},\vec{\mathbf{v}},\vec{\mathbf{n}})^T \bullet \vec{\mathbf{q}}$.

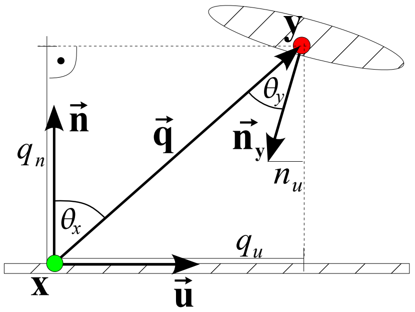


Fig. 6.3 – Quantities for computing the geometric gradient

Quantities for computing the geometric gradient at $\boldsymbol{x}$ *for a VPL at point* $\boldsymbol{y}$*.*

Hence, plugging Eq. 6.20 and Eq. 6.21 into Eq. 6.19 the derivative of G with respect to u is computed as

$$\begin{aligned}\frac{\partial G}{\partial u}(\mathbf{x},\mathbf{y}) &= -\frac{\partial}{\partial q_u}\left\{\frac{-(\vec{\mathbf{n}}_y \bullet \vec{\mathbf{q}}) \cdot q_n}{||\vec{\mathbf{q}}||^4}\right\} \\ &= q_n \cdot \left(\frac{n_u \cdot ||\vec{\mathbf{q}}||^2 - 4q_u \cdot (\vec{\mathbf{n}}_y \bullet \vec{\mathbf{q}})}{||\vec{\mathbf{q}}||^6}\right) \\ &= q_n \cdot \left(\frac{n_u \cdot ||\vec{\mathbf{q}}|| + 4q_u \cdot \cos\theta_y}{||\vec{\mathbf{q}}||^5}\right), \end{aligned} \tag{6.22}$$

where $n_u = \vec{\mathbf{n}}_y \bullet \vec{\mathbf{u}}$. And in the same way for the derivative along tangent vector $\vec{\mathbf{v}}$:

$$\frac{\partial G}{\partial v}(\mathbf{x},\mathbf{y}) = q_n \cdot \left(\frac{n_v \cdot ||\vec{\mathbf{q}}|| + 4q_v \cdot \cos\theta_y}{||\vec{\mathbf{q}}||^5}\right).$$

The gradient for the unprojected geometric term $G^{\perp}$ in Eq. 6.18 is derived in a similar fashion:

$$\frac{\partial G^{\perp}}{\partial u}(\mathbf{x},\mathbf{y}) = \frac{\partial}{\partial u}\left\{\frac{\cos\theta_y}{||\vec{\mathbf{q}}||^2}\right\} = \left(\frac{n_u\cdot||\vec{\mathbf{q}}||+3q_u\cdot\cos\theta_y}{||\vec{\mathbf{q}}||^4}\right), \tag{6.23}$$

and the same way for $\frac{\partial G^{\perp}}{\partial v}$.

3.3.2 Rotational Irradiance Gradient

The rotational gradient for the irradiance at a point $\mathbf{x}$ is computed by differentiating Eq. 6.8. Similar to [WH92] we compute the net gradient by summing the partial derivative with respect to the incident angle θ for each cluster in the lightcut. For convenience the rotational gradient is computed in the local tangent plane. Each single cluster is projected to the tangent-plane unit vector $\nu(\phi) = (\sin\phi, -\cos\phi)^T$, which is pointing in perpendicular direction[3] $(\phi - \frac{\pi}{2})$.

$$\nabla_r E(\mathbf{x}) \quad = \quad \begin{pmatrix} \frac{\partial E(\mathbf{x})}{\partial(\theta_x \sin\phi_x)} \\ \frac{\partial E(\mathbf{x})}{\partial(-\theta_x\cos\phi_x)} \end{pmatrix} = \sum_{i=1}^{N_L} \nu(\phi_i)\cdot I_i\cdot V(\mathbf{x},\mathbf{y}_i)\cdot\frac{\partial}{\partial\theta_x}\{G(\mathbf{x},\mathbf{y}_i)\} \tag{6.24}$$

where the visibility $V(\mathbf{x},\mathbf{y}_i)$ and the intensity I_i are invariant under rotation of the surface normal at $\mathbf{x}$. To show that Eq. 6.24 is a correct definition of the rotational gradient, we need to re-parametrize the geometric term G with the local incident direction (θ_x,ϕ_x) in 2D Euclidean coordinates (u,v)

$$G(\theta_x,\phi_x,l,\theta_y) = \frac{\cos\theta_x\cos\theta_y}{l^2} := g\left((\theta_x\cdot\sin\phi_x),(\theta_x\cdot-\cos\phi_x)\right) = g(u,v),$$

where $l = ||\mathbf{x}-\mathbf{y}||$ and θ_y are constant with respect to rotation of the surface normal at $\mathbf{x}$ and can thus be ignored in g. Using the chain-rule, the partial derivatives of G with respect to θ_x and ϕ_x are

$$\begin{aligned} \frac{\partial G}{\partial\theta_x} &= \frac{\partial g}{\partial u}\cdot\sin\phi_x + \frac{\partial g}{\partial v}\cdot-\cos\phi_x \\ \frac{\partial G}{\partial\phi_x} &= \frac{\partial g}{\partial u}\cdot(\theta_x\cdot\cos\phi_x) + \frac{\partial g}{\partial v}\cdot(\theta_x\cdot\sin\theta_x) = 0. \end{aligned} \tag{6.25}$$

Solving the relations in Eq. 6.25 for the partial derivatives $\frac{\partial g}{\partial u}$ and $\frac{\partial g}{\partial v}$ in the tangent plane we get

$$\begin{aligned} \frac{\partial g}{\partial u} &= \sin\phi_x\cdot\frac{\partial G}{\partial\theta_x} \\ \frac{\partial g}{\partial v} &= -\cos\phi_x\cdot\frac{\partial G}{\partial\theta_x}. \end{aligned}$$

[3] This practice is only for ease of derivation since the extrapolation direction as the result of the cross-product in Eq. 6.9 is orthogonal to the plane spanned by the two normal vectors

Thus, $\nabla g = \nu(\phi_x) \cdot \frac{\partial G}{\partial \theta_x}$. □

The derivative of the geometric term with respect to the incident angle θ_x is

$$\frac{\partial G}{\partial \theta_x} = \frac{-\sin\theta_x \cdot \cos\theta_y}{||\mathbf{x}-\mathbf{y}||^2}, \tag{6.26}$$

since only the term $\cos\theta_x$ changes with θ_x. When extrapolating the irradiance at $\mathbf{x}$ to a point $\mathbf{p}$ the rotational gradient is taken into account as a linear approximation in Eq. 6.9. The extrapolation direction $\vec{\mathbf{n}}_\perp = \vec{\mathbf{n}} \times \vec{\mathbf{n}}_k$ resulting from the cross product of the shading normal $\vec{\mathbf{n}}$ at $\mathbf{p}$ and the surface normal of the cache record $\vec{\mathbf{n}}_k$ has length $||\vec{\mathbf{n}}_\perp|| = \sin\theta$, where θ is the angle between the two normals. Using the $\sin(\theta)$ is only an approximation since our gradient is defined with respect to θ. However, for small angles as in radiance caching $\sin(\theta) \approx \theta$. Correctly scaling $\vec{\mathbf{n}}_\perp$ by $\arcsin(||\vec{\mathbf{n}}_\perp||)/||\vec{\mathbf{n}}_\perp||$ produces results that are visually indistinguishable to our approximation in Eq. 6.9.

In Fig. 6.4 the difference between irradiance caching without and with rotational gradients is shown. Note that in this special case even adaptive radiance caching (see Section 4.1) cannot circumvent the “artifacts” arising near discontinuities in the geometric normal since the computed “geometric” irradiance at the cache records does not match the irradiance for the corresponding shading normal (see Section 6.4).

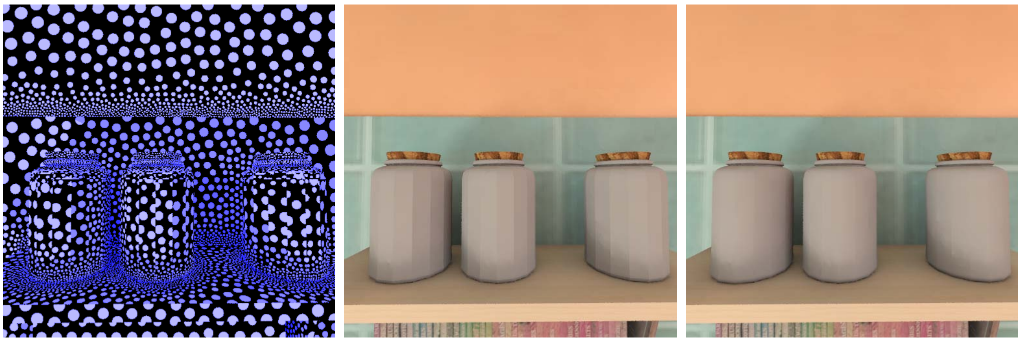


Fig. 6.4 – Rotational irradiance gradient

Indirect lighting computed with radiance caching (no adaptive caching) for a close-up view on curved objects with shading normals, left: the cache distribution and footprints, middle: ignoring the rotational irradiance gradients in the cache extrapolation reveals the discontinuous structure of the polygonal surfaces, right: cache extrapolation with rotational irradiance gradients.

3.3.3 Gradient Computation in the Lightcut

Whenever we evaluate the contribution of a visible light cluster, we also compute and store its gradient contribution in the cut according to Eq. 6.22, however, without multiplying the cluster intensity I_i since this is refined during the cut computation. Once a cut is fully converged we loop over all cut entries and sum the gradients multiplied by the final (possibly clampled) cluster intensity of each entry. The computation of the translational gradient requires clamping of the distance $||\vec{\mathbf{q}}||$ to prevent too small values for $||\vec{\mathbf{q}}||^5$ in the denominator of Eq. 6.22, which can lead to large numerical errors in the gradient computation particularly near corners.

The gradient computation only slightly influences the performance of the whole lightcut computation but significantly improves the cache interpolation quality (see Fig. 6.2). This is particularly important for the adaptive caching (Section 4.2) because interpolation without gradients creates many discontinuities resulting in an excessive cache density on flat, unoccluded surfaces.

4 Efficient Radiance Cache Splatting

In [GKBP05] an image space variant designed for the GPU of the widely used irradiance caching was presented. Besides being more suitable for a GPU implementation, it also bears several advantages, which we will discuss here. However, since the method operates in image space, it does not support splatting of indirect cache records arising from specular light transport along the eye path.

Therefore, we propose to splat cache records in object space independent of the camera. We do this via searching in a kd-tree for all feasible eye samples in the bounding sphere of the cache record. As the number of those searches in the eye sample kd-tree is relatively small compared to the number of eye samples, the search is algorithmically superior [HHS05] to the traditional irradiance caching [WRC88] where the number of searches (e.g. in an octree) is equivalent to the number of pixels in the image. The number pixels is usually about two orders of magnitude greater than the number of cache records. Thus, cache splatting [GKBP05] is less dependent on image resolution than cache gathering. Consequently, the search is not the bottleneck in the whole cache splatting algorithm and we splat each cache record one after another. This way the order of the cache-record generation and the splatting is *cache-oblivious* since the eye samples casted from the camera are spatially sorted in a kd-tree such that during splatting successive cache records access the memory in a highly coherent manner.

4.1 Anisotropic Cache Splatting

In [KBPv06] a method for adaptive irradiance caching was presented. It proposes to use individual error thresholds a per cache rather than a global error ε as in [WRC88] . This has the advantage that cache records can adaptively reduce their footprint where discontinuities in the cache interpolation are detected. Through multiple iterations eye samples are tested for visible discontinuities in the illumination extrapolated from neighboring cache records. Discontinuity causing cache-records are excluded from the interpolation and the cache record's error threshold a and hence its footprint is reduced accordingly until the solution converges (i.e. no more visible discontinuities). We follow this approach and omit the details here as they are well explained in [KBPv06] .

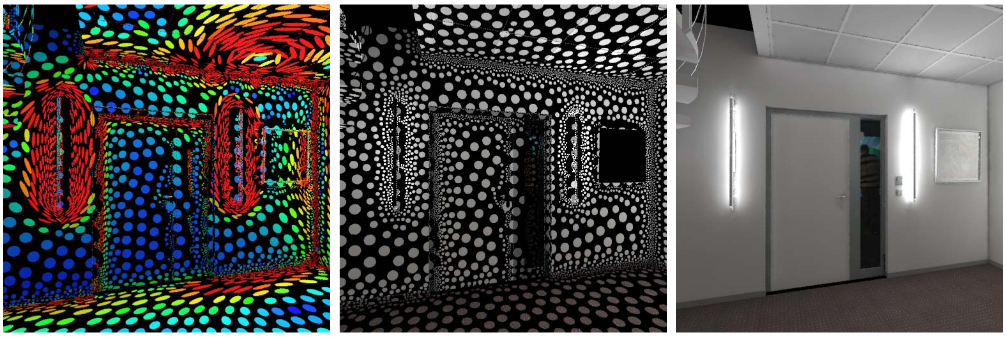


Fig. 6.5 – Anisotropic versus isotropic cache footprints

Left: color-encoded anisotropic cache-record footprints in the ICIDO *scene (red corresponds to maximum and blue to minimum anisotropy) (20881 records), middle: visualization of standard isotropic irradiance caching (23155 records), right: resulting indirect illumination ($\gamma = 2.5$, 8 f-Stops brighter), which is visually indistinguishable for both methods, although the numerical error is slightly smaller for the anisotropic reconstruction.*

We extend this method using anisotropic cache splatting. The adaptive caching of Křivánek et al. [KBPv06] increases the cache record density in regions with lighting changes, which the irradiance gradient can't compensate for. However, such strong lighting features are often of one-dimensional nature, which means that the changes in the direction orthogonal to the lighting gradient (the coherence direction) are often much smaller. This suggests to filter stronger in the coherence direction of a 2D signal and reduce interpolation along the gradient to avoid excessive "blurring" of lighting features. Such approach reduces the cache record density along edge features in the illumination (see Fig. 6.5). Anisotropic splatting and interpolation is not new but has been used frequently for example in image processing [Wei98, McC99] and point based rendering [ZPvBG01]. The proposed cache splatting imposes only small changes to the adaptive caching. Additionally, for each cache record we need the anisotropy and the direction of the gradient to perform anisotropic cache splatting.

The weights $w_k(\mathbf{p})$ are then computed similarly to traditional irradiance caching [TL04, WRC88]:

$$w_k(\mathbf{p}) = \left(\frac{||\mathbf{D} * (\mathbf{p} - \mathbf{x}_k)||}{max(R_-, min(R_+, R_k))} + \sqrt{1 - \vec{\mathbf{n}}_p \bullet \vec{\mathbf{n}}_k} \right)^{-1}, \tag{6.27}$$

where $\vec{\mathbf{n}}_p$ and $\vec{\mathbf{n}}_k$ are the surface normals at point $\mathbf{p}$ and cache position $\mathbf{x}_k$, respectively. Setting R_k to the harmonic mean distance of all visible surfaces [WRC88] computed with lightcuts does not yield a correct upper bound estimate for the indirect light gradient [WRC88] as we do not sample the hemisphere of directions uniformly (we essentially perform surface area integration). Therefore, we follow [TL04] and set R_k to the minimum distance instead. We use the distance to the nearest occluder computed from all traced shadow rays to the VPLs in the lightcut, which is clamped at a lower R_- and upper bound R_+. The bounds R_- and R_+ are computed from the projected pixel area at $\mathbf{x}_k$ [TL04] , which depends on the distance of $\mathbf{x}_k$ to the camera.

Since lightcuts is a deterministic approach, which samples all "contributing" light-emitting surfaces, we do not suffer from under-sampling of small nearby emitters (provided that small surfaces were sampled with VPLs) or ray leaking [KBPv06] as in Monte Carlo final gathering. Nevertheless, additional nearest neighbor clamping [KBPv06] should be applied since R_k is chosen as the distance to the nearest occluder rather than the distance to the nearest point light. And a small occluder, in particular when seen at a grazing angle, might not always be intersected by a shadow ray. Choosing the distance to the nearest occluder hardly changes R_k for the indirect light caches but is particularly important for direct light caches (see Section 4.3) where only very few, mostly distant surfaces emit light. Further, we would like to sample denser in shadowed regions in particular near contact shadows (see Fig. 6.6) in order to avoid under-sampling of high frequencies in the visibility function. Although such approach does not avoid light bleeding into the shadowed region, it can be detected later by the discontinuity checking of overlapping cache records (see Section 4.2).

The 3×3 matrix $\mathbf{D} = \mathbf{G} * \mathbf{A}^T$ transforms $(\mathbf{p} - \mathbf{x}_k)$ into the gradient-spanned coordinate system given by the orthogonal matrix $\mathbf{A} = \{v_{||}, v_\perp, n_k\}$, where $v_{||} = \nabla_t E(\mathbf{x}_k) / ||\nabla_t E(\mathbf{x}_k)||$ is the normalized gradient (in the local tangent plane) and $v_\perp$ is the coherence direction orthogonal to $v_{||}$, and then scales it by the diagonal matrix

$$\mathbf{G} = \begin{pmatrix} g(||\nabla_t E(\mathbf{x}_k)||^2)^{-1} & 0 & 0 \\ 0 & g(||\nabla_t E(\mathbf{x}_k)||^2) & 0 \\ 0 & 0 & 1/c_n \end{pmatrix}. \tag{6.28}$$

The matrix $\mathbf{G}$ suppresses the spherical splatting footprint along the illumination-gradient direction $v_{||}$ (see Fig. 6.7) and along the surface normal direction since those directions are likely to incur larger errors due to discontinuities. $c_n < 1$ is a constant scaling factor, which basically flattens the ellipsoidal splatting footprint in surface normal direction. We set it to 0.2. Note that we do not change the area

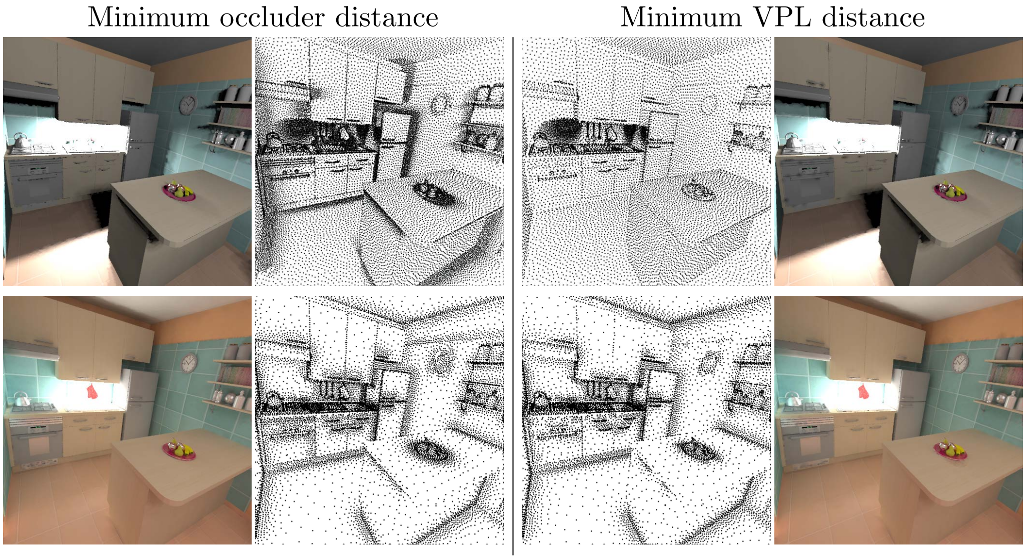


Fig. 6.6 – Visualization of cache locations for different metrics

Distribution and initial result of (top) direct light and (bottom) indirect light caching in the APARTMENT *scene. The center images show the initial cache locations when using the minimum occluder distance (middle left) and minimum VPL distance (middle right) for R_k in Eq. 6.27. The corresponding result is shown on the left and right images respectively. Note the missing contact shadows when using the VPL distance for computing the direct light.*

of a cache footprint in the tangent plane but just change its shape and orientation (see Fig. 6.7).

The monotonically decreasing function $g(x^2)$ maps the squared gradient magnitude to a value between 0 and 1 determining the anisotropy. We have chosen g as the *Perona-Malik diffusivity function* [PM90, Wei98] since it is efficiently computed as

$$g(x^2) = \frac{1}{1+x^2/\mu^2}, \tag{6.29}$$

with user defined contrast parameter μ: the smaller μ the higher is the sensitivity to the gradient magnitude yielding in more anisotropy along coherent lighting features. In order to prevent too elongated, slim footprints, we clamp $g(x^2)$ at a minimum of 0.3. One has to be aware that this diffusivity function was developed for image processing and considers absolute values of image-pixel gradients. In our case the illumination gradients depend on the scene measurement unit (e.g. inches) and lighting. Therefore, μ should be normalized to correspond to the actually perceived gradients after scaling the computed image to the desired image brightness (tonemapping).

4.2 Multi-Pass Adaptive Caching

In the final pass after all cache records have been created and splatted their contribution to all eye samples, we check at each eye sample e for discontinuities in the illumination as in [KBPv06] and reduce the footprint of the discontinuity-causing cache record such that the eye sample at which the discontinuity was detected is just excluded. Instead of resizing the footprint by reducing just the error a of a cache record (Fig. 6.7a), we reduce the footprint only in one dimension, either along the gradient $v_{||}$ or along the coherence direction $v_{\perp}$ while still keeping the symmetry and orientation of the footprint. We choose the dimension that maximizes the resulting footprint area when just excluding the eye sample e and recompute the resulting anisotropy and the new cache error a of the record (Fig. 6.7b). Note that this approach also compensates for wrongly oriented cache records due to the missing visibility gradients. One major difference to the *discernability metric* used to detect discontinuities in [KBPv06] is that our cache records are computed deterministically via lightcuts as opposed to Monte Carlo final gathering. Therefore, high-frequency noise is not present in the computed cache records and we do not need to estimate the standard deviation in the illumination to compensate for noise [KBPv06].

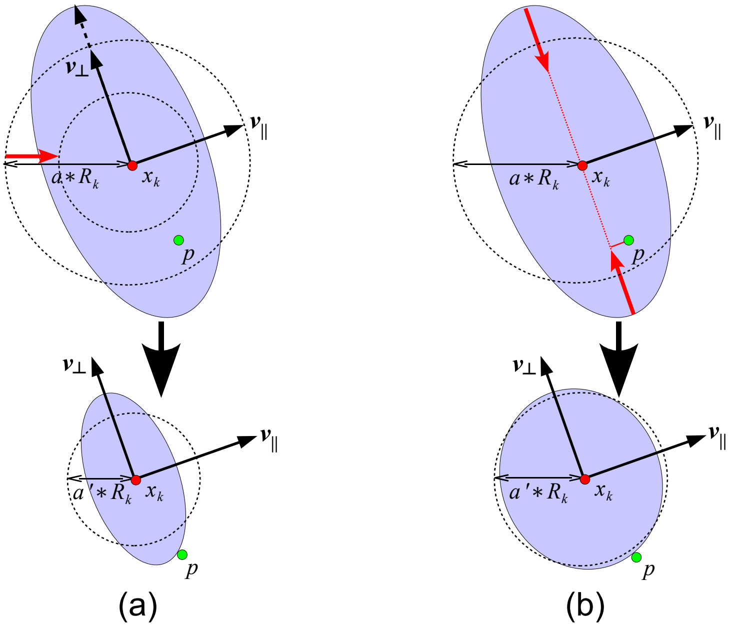


Fig. 6.7 – Anisotropic versus isotropic cache adaptation

(a) excluding the discontinuous eye sample at p from the cache footprint by reducing the cache error a while keeping the same anisotropy, (b) by shrinking the footprint only in the dimension with maximum elliptical distance to x and recomputing cache error a and the new anisotropy.

We tolerate discontinuities as long as they are invisible to a standard human observer, i.e. the lighting discontinuities are below the visibility thresholds [4]

4.3 Two-level Radiance Caching

Irradiance caching has been designed for indirect lighting and the direct light is commonly computed using a different method. However, the advantage of lightcuts is that we can seamlessly compute the direct and indirect lighting. When computing the direct and indirect light one after another in different rendering passes, we cannot obtain the full lighting information for a pixel. Since lightcuts adapts to the pixel luminance (Weber law), such an approach would result in larger cut sizes and therefore higher computation costs for the first pass rendering. This is particularly problematic when first computing the direct light in fully occluded regions where only indirect light is contributing since lightcuts will refine the cut to its maximum size. Contrarily, if we were computing the indirect light first the lightcuts algorithm would excessively refine the cache records in regions which are actually dominated by direct illumination and could thus tolerate a much higher error. Therefore, we compute direct and indirect light in parallel with our (ir)radiance caching such that the indirect light can mask the direct light during the lightcut refinement and vice versa. We maintain two cache layers and two light trees, one for direct light and one for indirect light. The main rendering loop for the cache splatting described in Section 4 needs only small modifications to test and compute both cache layers simultaneously and to weight the direct and indirect light contributions of cache records individually. A rendering pass is finished if the accumulated weights of both cache layers in each pixel are above the required user-thresholds, which can be set individually for direct and indirect light (see Fig. 6.8).

5 VPL Clamping Bias Compensation – Dealing with Weak Singularities

The lightcuts algorithm based on instant radiosity estimates the rendering equation by integrating the radiant intensity over the surface area using point samples (VPLs). This naturally leads to the problem of weak singularities, which results in low-frequency noise in corners seen as “blotchy” artifacts. Although lightcuts is highly scalable – we can use hundreds of thousands to millions of VPLs – it still requires clamping of very close VPLs resulting in darkening in corners as shown in the left image of Fig. 6.9. Those clamping “artifacts” are even more striking on glossy surfaces where individual VPLs that fall into the gloss lobe of

[4] A contribution of a cache record to an eye sample is said to be continuous if the interpolated pixel radiance including this cache record is indistinguishable from the pixel radiance without the cache record. This requires that we have at least two cache contributions per eye sample.

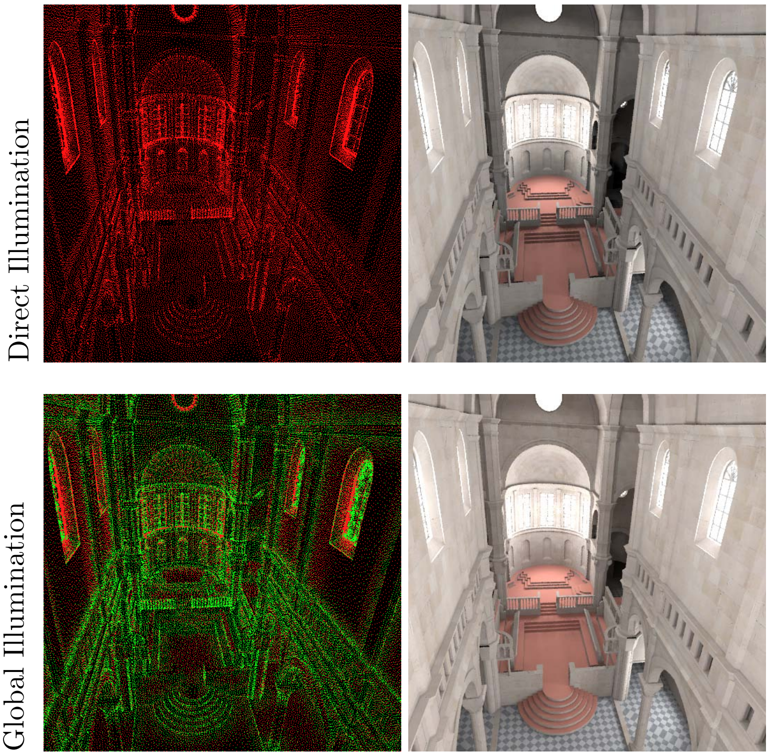


Fig. 6.8 – Two-layered radiance caching

left: distribution of the cache records in the SIBENIK *scene, red dots represent direct light and green dots indirect light caches, right: rendered results.*

the BRDF contribute relatively high to the pixel radiance and thus need to be clamped even more than for diffuse surfaces. We can compensate for the clamping of the VPL contributions by using a second estimator that integrates uniformly over the solid angle. However, instead of fixing the clamping threshold for the geometric term manually as in [KK04], we automatically compute a perceptual upper bound based upon the relative mean contribution of a VPL to its cache records's irradiance. This has two advantages. First, the clamping threshold adapts automatically to the total number of VPLs in the scene and second, it is also pixel- or cache-adaptive according to Weber law: the luminance contribution Y_c of a VPL is clamped at $T_c = 1\%$[5] of the luminance $Y\{E(\mathbf{x})\}$ of the irradiance at the cache location

$$Y_c = \min\left\{0.01 \cdot Y\{E(\mathbf{x})\}, G(\mathbf{x},\mathbf{y}) \cdot Y\{I(\mathbf{y})\}\right\}. \tag{6.30}$$

Assuming that all clamped VPLs have a similar intensity $I(\mathbf{y})$, which is reasonable if we assume VPLs where sampled using photon path-tracing with ideal BRDF

[5] Setting the clamping threshold to 1% is usually over-conservative in particular in darker regions and for many scenes a threshold of 2% leads to better visual quality with regard to the trade-off between low-frequency noise and clamping bias. However, whenever clamping bias compensation is enabled, we prefer to be more conservative initially.

importance-sampling and Russian Roulette absorption, we can compute the upper bound $b(\mathbf{x})$ for the geometric term $G(\mathbf{x},\mathbf{y})$ at each point $\mathbf{x}$ as

$$b(\mathbf{x}) = \frac{0.01}{\bar{Y}_I} \cdot Y\{E(\mathbf{x})\}, \tag{6.31}$$

where $\bar{Y}_I$ is the maximum luminance of the intensities of the clamped VPLs in the lightcut. Note that the proposed computation of the adaptive clamping threshold is only feasible for diffuse surfaces. For glossy surfaces the irradiance contribution of individual VPLs is weighted differently for varying viewing angles according to a non-uniform BRDF, which is not known during the cache record computation. Nevertheless, we compute a constant clamping threshold $b(\mathbf{x})$ per cache record the same way as for diffuse BRDFs. The irradiance of a cache record is then computed as follows:

$$\begin{aligned} E(\mathbf{x}) &= E_{cut}(\mathbf{x}) + E_{clamp}(\mathbf{x}) \\ &= \int_S \min\{b(\mathbf{x}), V(\mathbf{x},\mathbf{y})G(\mathbf{x},\mathbf{y})\} L_e(\mathbf{y} \to \mathbf{x}) dA_y \\ &\quad + \int_S \max\{0, V(\mathbf{x},\mathbf{y})G(\mathbf{x},\mathbf{y}) - b(\mathbf{x})\} L_e(\mathbf{y} \to \mathbf{x}) dA_y \end{aligned} \tag{6.32}$$

The irradiance contribution E_{clamp} from the second integral in Eq. 6.32 compensates for the clamped VPL contributions in the first integral E_{cut}, which we estimated with lightcuts. E_{clamp} is reformulated to an integral over solid angle:

$$E_{clamp}(\mathbf{x}) = \int_\Omega \frac{\max\{0, G(\mathbf{x},\mathbf{y}) - b(\mathbf{x})\}}{G(\mathbf{x},\mathbf{y})} L(\mathbf{x},\omega) \cos\theta_x \, d\omega, \tag{6.33}$$

where $L(\mathbf{x},\omega)$ is the incident radiance from direction ω. E_{clamp} can now be solved via Monte Carlo final gathering with uniform sampling of the hemisphere of directions Ω

$$E_{clamp} \approx \frac{1}{N} \sum_{i=1}^{N} \frac{\max\{0, G(\mathbf{x}, h(\mathbf{x},\omega_i)) - b(\mathbf{x})\}}{p(\omega_i) G(\mathbf{x}, h(\mathbf{x},\omega_i))} L_e(h(\mathbf{x},\omega_i), -\omega_i) \cos\theta_i, \tag{6.34}$$

where $h : \mathbb{R}^5 \to \mathbb{R}^3$ is the raytracing operator and ω_i is sampled uniformly according to the probability density function $p(\omega) = 1/(2\pi)$.

Compensating for the clamped energy using a solid angle based sampling approach diminishes the speedup we gained with our method. This is mainly due to the fact that radiance caching places most cache records near edges where the clamping actually happens. Therefore, the expensive clamping correction is often computed for the majority of the cache records. Unfortunately, importance sampling the integrand in Eq. 6.33 is difficult as we may not be able to estimate the geometric term correctly in the close neighborhood. There can be small surface emitters that were undersampled for which we actually want to compensate for

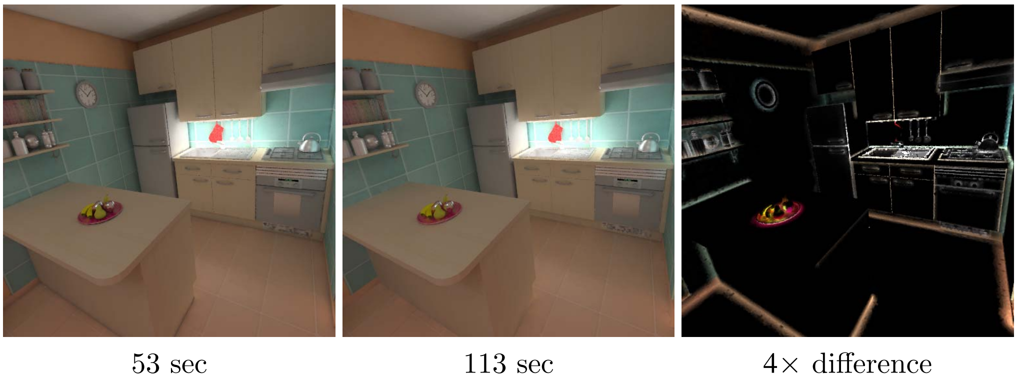


Fig. 6.9 – VPL-clamping-bias compensation

Left: clamped indirect lighting in the Apartment *scene using* 550,000 *VPLs, middle: the same scene with the proposed clamping compensation with 1000 final gather rays per cache record and 100 nearest neighbor "VPLs", right: visualization of the clamped irradiance estimate (2 f-stop brighter) only, which was added to the left image to obtain the middle image. Note that this contribution automatically decreases with the total number of VPLs distributed in the scene and when direct lighting is added (since the luminance $Y\{E(\boldsymbol{x})\}$ in Eq. 6.31 is increased). Approximately 20000 cache records with a mean lightcut size of 752 clusters were computed in 10 iterations for both methods.*

with our second estimator. However, the solid angle based estimator in Eq. 6.34 heavily relies on raytracing of nearby surfaces. In order to speedup the raytracing step, we can bound the ray intersection distance t given our geometric clamping threshold $b(\mathbf{x})$

$$t_{max}(\theta_i) = \sqrt{\frac{\cos\theta_i}{b(\mathbf{x})}}. \tag{6.35}$$

The bound t_{max} will result in very short inital rays to be traced in particular for grazing angles. This in turn culls unnecessary ray traversal steps of nodes in the acceleration data structure that are farther away and makes the raytracing part of our uniform hemisphere sampling highly cache coherent since only a few localized nodes need to be accessed by all final gather samples. And perhaps more important in most cases we avoid incoherent access and intersection tests of geometric primitives that would otherwise be needed to find the nearest ray intersection.

In order to further speed up the computation of E_{clamp} we need to find an efficient approximation for $L_e(h(\mathbf{x},\omega_i),-\omega_i)$ in Eq. 6.34 as it is the actual bottleneck. We make a few simplifying assumptions:

1. First, one can observe that the clamped irradiance $E_{clamp}(\mathbf{x})$ is non-zero in only small local regions and quickly decreases towards zero, where the clamping region reduces with the number of VPLs.

2. Second, the change in irradiance in a local neighborhood (represented by a few VPLs) is very small with respect to the solid angle (i.e., a relatively

small area projects to a large solid angle at $\mathbf{x}$).

3. Third, we only assume diffuse secondary light transport (VPLs) from $\mathbf{y} := h(\mathbf{x}, \omega_i)$ to $\mathbf{x}$, i.e., $L(\mathbf{y} \to \mathbf{x}) = L_e(\mathbf{y}) = \frac{\rho_d(\mathbf{y})}{\pi} \frac{d\Phi(\mathbf{y})}{dA}$ and the change in incident radiance $L(\mathbf{x} \leftarrow \mathbf{y})$ is dominated by the BRDF $\frac{\rho_d(\mathbf{y})}{\pi}$ (due to diffuse textures) at the secondary hit point $\mathbf{y}$.

Combining those observations we propose to compute the irradiance $E(\mathbf{y}) = \frac{d\Phi(\mathbf{y})}{dA}$ only once at a valid and contributing point $\mathbf{y}' = h(\mathbf{x}, \omega_i)$ while integrating correctly the surface material properties and geometric quantities in Eq. 6.34

$$\begin{aligned} E_{clamp}(\mathbf{x}) &= \int\limits_{\Omega} \frac{\rho_d(\mathbf{y})}{\pi} \frac{d\Phi(\mathbf{y})}{dA} \frac{\max\{0, G(\mathbf{x},\mathbf{y}) - b(\mathbf{x})\}}{G(\mathbf{x},\mathbf{y})} \cos\theta_x \, d\omega \\ &\approx \frac{d\Phi(\mathbf{y}')}{dA} \int\limits_{\Omega} \frac{\rho_d(\mathbf{y})}{\pi} \frac{\max\{0, G(\mathbf{x},\mathbf{y}) - b(\mathbf{x})\}}{G(\mathbf{x},\mathbf{y})} \cos\theta_x \, d\omega, \end{aligned} \tag{6.36}$$

where $G(\mathbf{x},\mathbf{y}') - b(\mathbf{x}) > 0$. This way we only pay for at most one additional irradiance computation per cache record. The irradiance is computed using photon (i.e flux) density estimation from the k-nearest neighbor VPLs, which are efficiently found by exploiting the light hierarchy used to compute the lightcut. We could also use lightcuts to compute $E(\mathbf{y}')$. However, besides being more expensive to compute, it suffers from discontinuities in the irradiance (we only use one sample), which we regularize when using density estimation. The error introduced by this approximation compared to the accurate irradiance computation via density estimation at every secondary ray hit point is hardly visible as shown in Fig. 6.10 and moreover decreases automatically with the total number of VPLs in the scene.

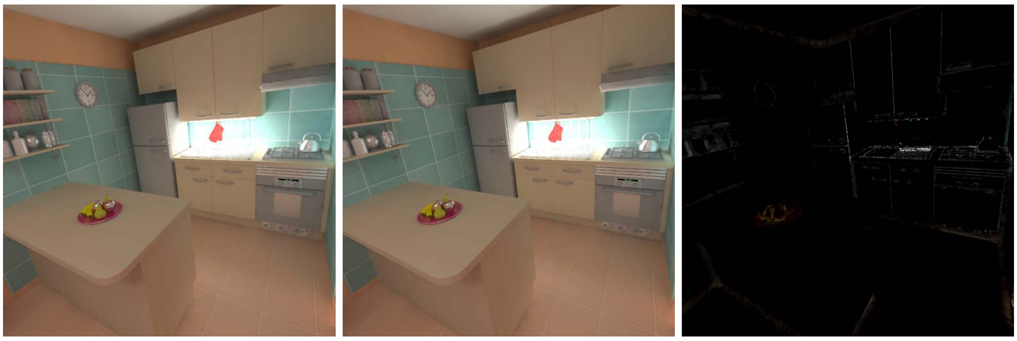


Fig. 6.10 – Approximate VPL-clamping-bias compensation

The difference between the accurate clamping bias compensation (left) and our proposed cached irradince at a single clamped secondary hitpoint (middle) is perceptually invisible as the integration domain (where $G(\boldsymbol{x},\boldsymbol{y}) > b(\boldsymbol{x})$) is usually smaller than the density estimation area. The difference image (right) is scaled by 2 f-stops.

An important point is that we should perform the clamping compensation for each cache record regardless whether it was clamped during the lightcut computation or not. The reason for this is that the "blotchy" artifacts arise because of sampling noise in corners where the VPL density is insufficient. Hence, some regions suffer from over-estimation and some from under-estimation (i.e. "missing" VPLs). In those under-estimated regions there is no clamping detected but we still have to compensate for it. A good indicator is the harmonic mean or minimum distance R_k, which is computed for each cache record, see Eq. 6.27. However, at the time the cache record has been computed with lightcuts, R_k is still prone to undersampling of small surfaces (nearest neighbor clamping is performed in the following pass). Therefore, we perform the clamping compensation for each cache record and pay the penalty of longer cache computation time, which is, depending on scene complexity and ray shooting efficiency, 10% to 15% of pure cache computation time (without interpolation) in our implementation. Note that this penalty is only due to redundant secondary ray shooting of ca. 1000 "short" rays at un-clamped cache locations.

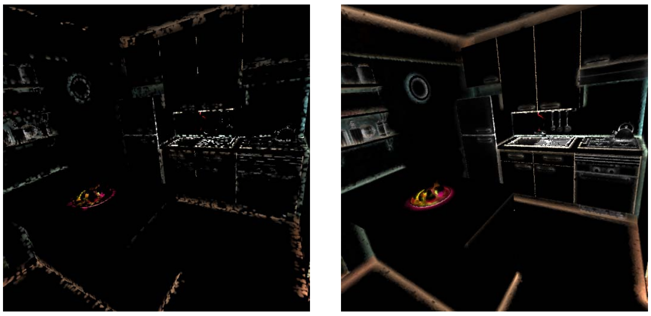


Fig. 6.11 – Conservative VPL-clamping-bias compensation

Left: erroneous clamping compensation of the indirect lighting at selected cache locations that were clamped during the lightcut computation, right: correct clamping compensation computed at all cache records.

6 Implementation Details

6.1 Light-tree Construction

A light-tree is initially constructed over the sampled VPLs in a greedy bottom-up fashion as in [WFA*05]. We also tested a simple recursive top-down clustering, which can be very efficiently computed (<1 sec for 500,000 VPLs). However, it results in poorer performance during rendering due to larger lightcut sizes arising from less precise error bounds. The naive greedy bottom-up construction has $\mathcal{O}(N^3)$ time complexity and may take a few hours for clustering half a million VPLs. To achieve a tree construction time-complexity of $\mathcal{O}(N \log N)$ is non-trivial and requires several optimizations, which are analyzed and well-explained in [WBKP08]. Since the constructed light-tree is in fact a bounding volume hierarchy over point lights (VPLs), we also exploit it for efficient search queries to find the k-nearest neighbor VPLs (see Section 5). The clustering is conservative in a greedy sense and is computed in 5 to 30 seconds for $500,000$ VPLs depending on the scene.

6.2 VPL Occlusion Caching

Our cache computation and thus also the lightcut computation is very coherent since we iterate over the spatially sorted eye samples rather than over pixels in the image. This is particularly important as we access a large amount of data (light-tree and VPLs, eye samples and cache records, the frame buffer and cache-splat buffer), which might not always fit into the CPU caches. Furthermore, we can also exploit this coherence during ray-shadow testing of clusters using a shadow cache. Assuming that the distance between successively created cache records is small with respect to the occluding primitive's size and that neighboring cache records evaluate similar light clusters during the lightcut computation, we can cache the last occluder for each evaluated VPL. Since we would like to keep the cache small and as there are usually too many VPLs, we map the index of a cluster's representative VPL to a hash index in a small hash-table, where we cache the last occluder of the VPL. We set the size of the hash-table to a power of two (e.g. 4096) in order to use a fast modulo hash-function via bit masking. Using prime numbers for the table size of a "devision rest" hash-function is usually a better choice in terms of hash collision [CLRS01]. In our tests however, it does not perform better than simple power-of-two-modulo hashing (see red "cross-haired" line in Fig. 6.12 left). Each successive shadow test for a VPL first checks the last occluder stored in the hash-table. This results in early culling of expensive shadow ray-traversals in the raytracing acceleration data structure for more than 25% of indirect shadow tests and more than 40% for direct light shadow tests for all tested scenes with negligible computational overhead. Statistics for the cache hit ratio computed from 40 frames of a walk-through in the APARTMENT scene are shown in Fig. 6.12. Note that the cache hit ratio depends strongly on cache density and the "pixel-traversal" order. For direct light the hit ratio is higher than for indirect light caching since the cache density is higher and the light tree and lightcut sizes are smaller compared to the indirect light cache. The graphs in Fig. 6.12 show that the hit ratio also increases with the distance of a cluster and the elevation angle towards the cluster. This is reasonable as distant clusters represent high levels in the light-tree and are spatially coherent in the lightcuts of nearby cache records (more likely to be included in the lightcut) whereas the distribution of nearby clusters or leaves in the lightcut varies more frequently due to the lightcuts refinement procedure (see Section 2). The same holds for the elevation angle of the direction towards the cluster. At grazing angles the refinement of clusters is stopped at a higher level in the light-tree (due to the cosine term) than for clusters seen at small elevation angles. In other words, the average hit ratio of shadow rays increases with increasing cluster coherence in the lightcut, which is roughly inversely proportional to the form factor of the cluster (the geometric term in the rendering equation), which the lightcut adapts to. These observations can be used to further improve the efficiency of the proposed occluder caching. For example we could reject all cache queries for short rays, which are more likely to be unoccluded.

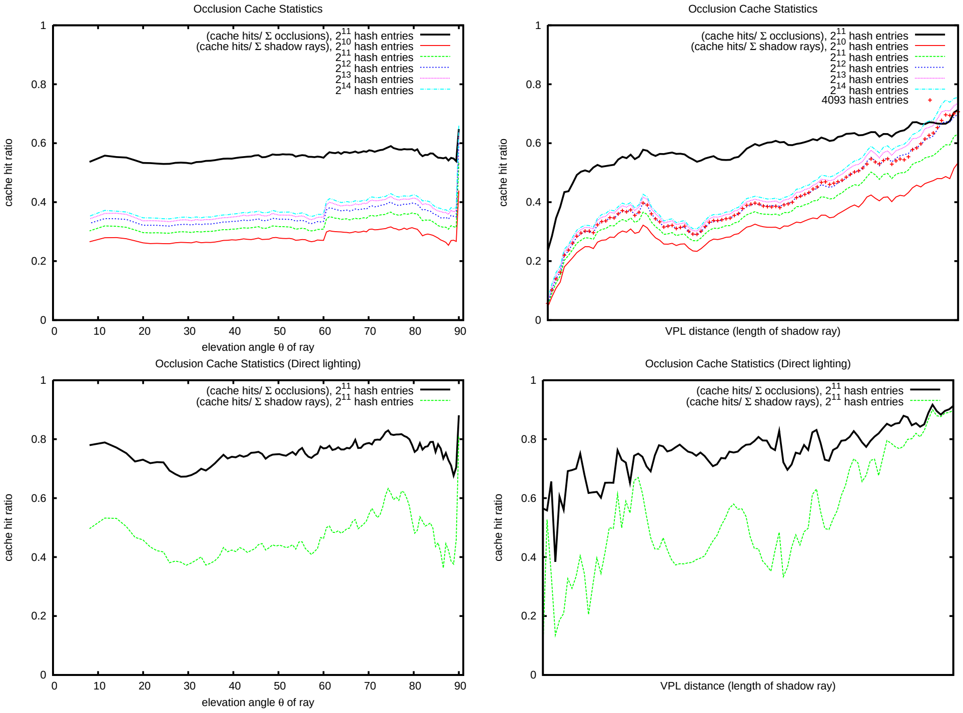


Fig. 6.12 – Statistics for VPL occlusion caching

Graphs showing visibility cache statistics with respect to incident angle (left) and length (right) of a shadow ray generated from 40 different frames in the APARTMENT *scene for different occlusion cache sizes. The graphs show the statistics for only indirect lighting (top) (550000 VPLs and 822 VPLs mean lightcut size) and only direct lighting (bottom) (5000 VPLs and 261 VPLs mean lightcut size) computed with our radiance caching. The colored curves show the successful hit ratio of cache occlusion hits to all shadow ray tests, which also includes the case where a shadow ray is not occluded as a false hit. The actual cache hit ratio of cache occlusion hits to all occluded shadow ray test is much higher as shown by the thick black curve (> 50%, > 70% for indirect light, direct light respectively).*

6.3 Setting the Irradiance Cache Error

As in traditional irradiance caching [WRC88] the user sets the initial maximum cache-interpolation error-threshold $a_k := \varepsilon$, which is then adaptively refined if discontinuities in the illumination are detected (see Section 4.1). An eye sample at point $\mathbf{p}$ receives a contribution from an extrapolated cache record k if the error $1/w_k(\mathbf{p})$ is smaller than a_k.

$$\frac{1}{w_k(\mathbf{p})} < a_k \Rightarrow \text{extrapolate.} \tag{6.37}$$

The weight $w_k(\mathbf{p})$ is computed as in Eq. 6.27. This however results in many eye samples receiving only a contribution from one cache record, which is undesirable for the adaptive cache refinement as we need at least 2 overlapping records in order to detect discontinuities (see Section 4.2). In our implementation the user is allowed to set the mean number of cache record contributions N_c per eye sample and the maximum cache-interpolation error ε separately. Then, a new cache record is computed at $\mathbf{p}$ only if:

$$\frac{1}{\sum_k w_k(\mathbf{p})} > \frac{\varepsilon'}{N_c} \Rightarrow \text{compute new record,} \tag{6.38}$$

where $w_k(\mathbf{p}) > 1/a_k$ with $a_k := \varepsilon' = \varepsilon \cdot N_c$. This modification still results in single cache contributions close to the cache record's location where the cache weight w_k is high. For most scenes setting $N_c := 2$ produces better results and moreover helps to robustly detect lighting discontinuities of overlapping cache records, which is needed for the adaptive cache splatting.

6.4 Shading Normals

Shading normals are a common way to hide discontinuities in the surface orientation at polygon boundaries (interpolated vertex normals) and to fake high-frequency geometric structure on otherwise flat surfaces (bump or normal map). However, when using shading normals two problems arise in the context of global illumination. First, a normal-perturbed surface can “generate” false energy, i.e. it may reflect more energy than it receives [Vea97] if the surface area is virtually decreased. Second, as the perturbed normals are no longer consistent with the actual surface, precise geometrical computations like for example visibility tests are biased. This may appear in form of erroneous self-intersections and light-leaking. This is particularly cumbersome for irradiance caching approaches, where the cache density adapts to the harmonic mean or minimum distance to the nearest occluder(s), which may result in very small distances due to self-intersections when using the shading normal. Further, the BRDF for normal-perturbed surfaces is non-symmetric anymore, which means that in a bidirectional algorithm light paths and eye path (importance transport) require different treatment. Tracing light particles (VPLs) for example can increase the reflected energy arbitrarily on normal perturbed surfaces [Vea97] making early path termination problematic and moreover increases the variance in the VPL's energy. In order to prevent those inconsistencies we always use the *correct* geometric surface normal for all precise visibility computations involving ray shooting as for example the previously proposed clamping bias compensation with hemisphere sampling and the particle (VPL) tracing. As we would like to capture the full hemispherical radiance field all cache records are computed using the correct geometric surface normal. Even though it is not always the best choice, we also use the geometric surface normal for the computation of the cache weights in Eq. 6.27[6]. Otherwise, the cache density increases artificially on normal perturbed surfaces and worse, holes may appear on artificially curved surfaces where the shading normal deviates strongly from the geometric normal (see Fig. 6.13). The shading normal is only used when the cache is complete and the final image radiance is computed, i.e. when the records extrapolate their (ir)radiance coefficients to the eye samples, at which the local coordinate frame is aligned with the shading normal such that the BRDF SH coefficients correspond to the desired shading normal.

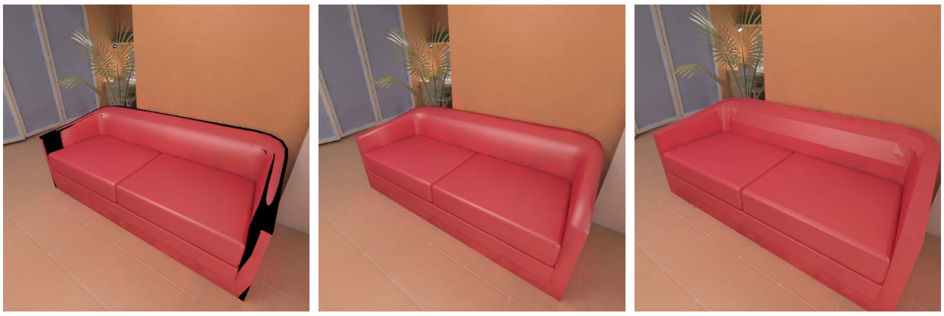


Fig. 6.13 – Shading normals and radiance caching

Cache weighting with shading normals (left) can generate holes in the cache splat buffer when geometric normals and shading normal differ significantly. Using the physically correct geometric normal in the cache record computation as well as for the cache weighting with subsequent SH radiance rotation (middle) avoids this problem and above all, generates less records on normal perturbed surfaces. Right image shows the result when ignoring shading normals.

[6] Alternatively, we could use the shading normal for the computation of the cache weights in Eq. 6.27 for both, the cache record and the eye sample, which often produces better results but increases the cache density in case of high-frequency bump-mapped surfaces.

6.5 Walk-through Animations

Radiance caching is well-suited for walk-through animations since the previously computed cache records can be reused in successive frames and only few cache records need to be added from frame to frame. However, a few changes need to be considered to prepare the algorithm for walk-throughs. First of all, to avoid exhaustion of memory all cache records need to maintain an age indicating the last frame the record contributed to. All records that did not contribute to the last n frames are deleted at the end of each frame to make space for new caches. Second, we can further exploit temporal coherence between frames to compute perceptual visibility thresholds for each pixel. These can then be reprojected and used in the next frame to detect discontinuities and steer the adaptive cache splatting (see Section 4.2). For all feasible pixel reprojections we use the computed visibility thresholds instead of the discontinuity thresholds based on the Weber law [KBPv06].

The computation of the perceptual visibility thresholds is borrowed from the well-established JPEG/MPEG compression standards [RW96]. Such a threshold computation based on the discrete cosine transform (DCT) of (8×8) image blocks can be very efficiently computed (ca. 60 ms for a 500×500 pixels on a single PC), which we describe in detail in Chapter 7. The user is able to set a scaling parameter q_{scale} (originally used in MPEG to adapt the quantization level to the bandwidth of a channel) to define the quality of the overall cache interpolation. Higher q_{scale} values raise the visibility thresholds and increase the sensitivity to luminance and contrast masking [HKMS08, RW96]. Setting $q_{scale} <= 4$ computes thresholds below the just-noticeable-difference thresholds and does not generate noticeable artifacts in our tests. In contrast to the approach discussed later in Chapter 7 we do not use the thresholds for the lightcut computation but only for the discontinuity detection in the cache interpolation. An example for a frame is shown in Fig. 6.14.

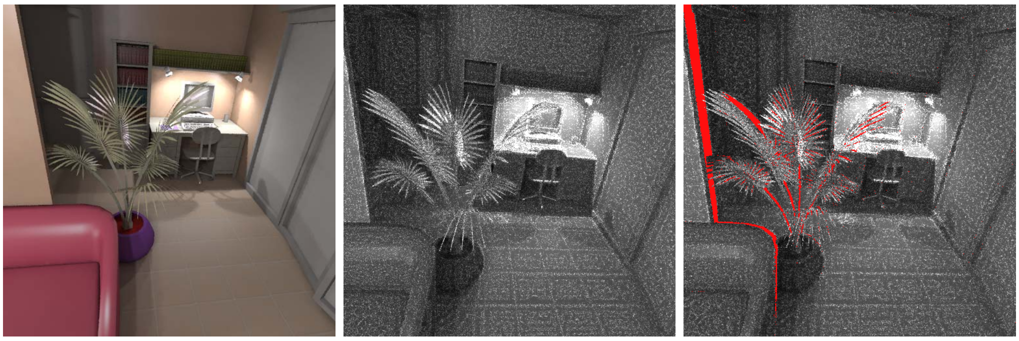


Fig. 6.14 – Warping frames in walk-through animations

(left) one frame from a camera walkthrough in the APARTMENT *scene computed with lightcut caching for direct and indirect light, (middle) computed thresholds for $q_{scale} = 5$, (right) reprojected thresholds from previous frame (red color means no correspondance). For visualization the thresholds are scaled by 32.*

7 Results

We compared our method with the lightcuts algorithm [WFA*05], which computes the radiance for each pixel and with photon mapping combined with radiance caching [KBPv06]. To be fair all methods were integrated in the same renderer using the same basic data structures and algorithms. The photon mapping implementation uses the same adaptive cache splatting procedure as for our lightcuts caching, only the computation of the caches differs. The gradients are computed numerically in a stratified manner [Kǒ5b]. For precomputing the BRDF hemispherical harmonics (HSH) coefficients and rotating the radiance HSH coefficients we included the open source code provided by J. Křivánek [Web]. All results were obtained on an AMD Athlon 64 (2.2 GHz) with 3 GByte of RAM. In order to compare with the lightcuts algorithm we provide the numerical results in Table 6.1 without pixel super-sampling while we show antialiased images in Fig. 6.15 using 4 super-samples per pixel. Since the eye-pass (raycasting the framebuffer and storing eye samples) is the same for all methods it is not explicitly shown in Table 6.1. Although there are still small differences to the reference computed with lightcuts [WFA*05], the overall quality is high while the rendering time is in the order of 20 to 100 seconds for computing a single image in high quality. Interestingly, one can observe in Table 6.1 how the direct light masks the indirect light during the cache computation. A good example is the diffuse SIBENIK scene where the computation of direct plus indirect light is even more efficient than solely computing the indirect lighting with our caching. The discontinuity detection and cache adaptation described in Section 4.2 takes about 0.2 to 0.5 seconds for 5 cache iterations and 512^2 pixels for the tested scenes.

The KNOT scene is lit by high-frequency direct and indirect light casting sharp shadows on glossy and diffuse surfaces. The KITCHEN and APARTMENT scene ©INRIA 2005 is challenging because it contains many light sources and small objects and most surfaces are slightly glossy. Also the indirect light is very strong and localized (e.g. above the chandelier) and photon mapping requires about 4000 final gather rays per cache record to avoid the otherwise distractable noise. Compared to a path-traced reference, the result computed with photon mapping is however slightly better than our result mainly due to the VPL clamping in corners. We also computed two walk-through animations, one only with indirect the other with global illumination, in the APARTMENT scene consisting of 281 frames. Using the discernibility metric described in Section 6.5 with $q_{scale} = 4$ instead of the Weber law with 2% luminance thresholds [KBPv06] reduces the total number of indirect light records by 19% in this walk-through animation. The icido scene is an example of difficult indirect lighting. Even with 5000 final gather rays per cache record the photon mapping results are too noisy and prone to artifacts. For this scene anisotropic caching pays off most and less cache records are created around the light sources and in corners than for isotropic caching (see Fig. 6.5).

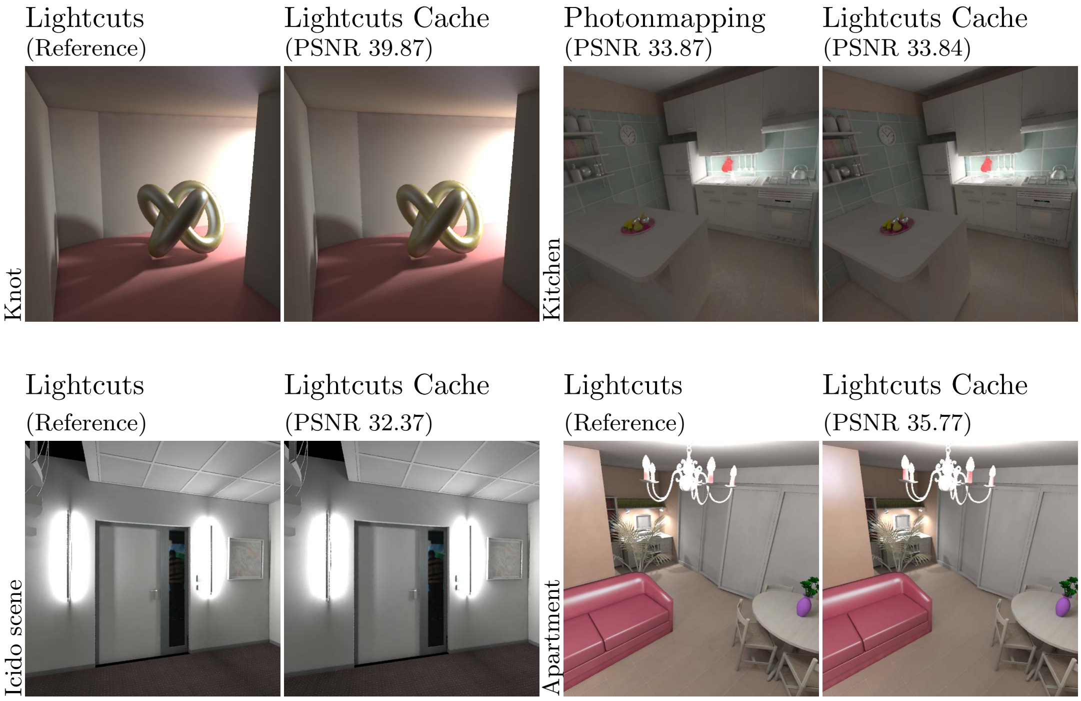


Fig. 6.15 – Result images with visual comparison for radiance cache splatting

Rendered images and peak-signal-to-noise-ratio (PSNR) for our method (lightcuts cache) compared with lightcuts on a per-pixel basis and photonmapping with radiance caching. The indirectly lit kitchen-view results are compared against an unbiased path-traced image as reference (not shown).

8 Discussion and Future Work

Although our method seems superior in terms of robustness and efficiency of the global illumination computation, it also has a few disadvantages when comparing it with traditional photon mapping. While photon mapping is able to deal with low-frequency caustics, yet on a high price, our method cannot handle any caustics because photons stored on moderately glossy surfaces are treated like diffuse VPLs. Extending the lightcuts algorithm to cluster VPLs on moderately glossy surfaces and computing the upper error bounds by taking also into account the BRDF at the clusters locations during the lightcut computation is left as future work.

Second, the clamping of close VPLs results in bias in particular on glossy surfaces near corners where only a few VPLs (in the BRDF lobe) have a very high contribution to the pixel radiance. The clamping correction described in Section 5 can always be applied but is computationally more expensive. However, also photon mapping suffers from bias in corners due to noise and overestimation of photon density since almost the same set of photons is queried by many final gather rays (see Fig. 6.16 left). Consequently, to produce high quality images, photon mapping also requires secondary final gathering [JSCK02].

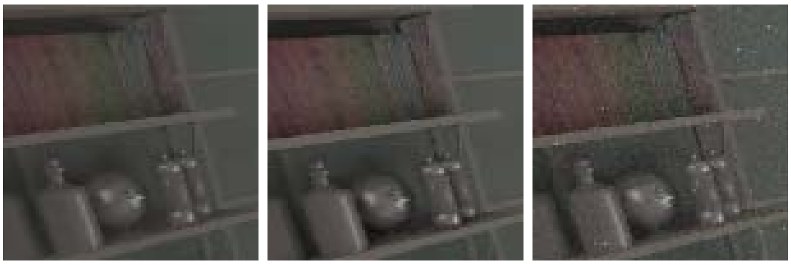


Fig. 6.16 – Visual comparison with unbiased path-tracing

Zoom into kitchen scene result of Fig. 6.15, left: photon mapping, (middle) lightcut caching, (right) pathtracing.

Third, along sharp shadow boundaries cache records do not always reduce to a pixel-sized footprint. Hence, shadow edges may appear slightly jaggy or washed out. For high-quality direct lighting a smart algorithm should detect and separate single point light and directional light sources and compute their contribution for each pixel.

8.1 Visibility Gradients

One drawback of the proposed irradiance-gradient computation in Section 3.3 is the absence of visibility gradients (we simply assume the visibility gradients are zero within a cache record's footprint) since we only know the point to point visibility). This results in errors in particular for the direct light gradients (see Fig. 6.17). While a stratified gradient computation as in [Kŏ5b] should

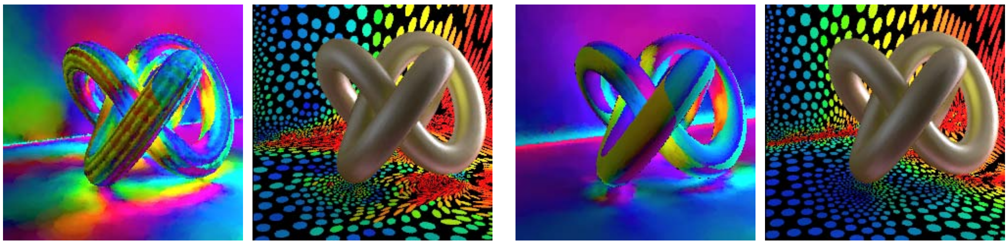


Fig. 6.17 – Visibility gradients and radiance caching

Color-encoded gradient direction and corresponding anisotropic cache footprints where color shows anisotropy, (left) stratified gradients can handle visibility, (right) our gradients fail in the presence of strong occlusions.

be feasible, it is not as trivial for irregular surface area sampling of VPLs as for uniform solid-angle based sampling. Nevertheless, we can rearrange the computed lightcut into a representation that is suitable for computing stratified gradients as in [Kŏ5b]. One possibility could be to project all VPL clusters of the computed lightcut onto the hemisphere of incident directions positioned at a cache record and constructing a voronoi diagram where each uniform stratum of the subdivided hemisphere is assigned to the nearest projected cluster point, storing the cluster's distance and the cluster's (ir)radiance contribution, which is divided by the number of strata the cluster covers. For the remaining steps of the stratified gradient computation we can proceed the same way as in [K05b]. Interestingly, this way, directions with high contribution (i.e. small number of strata covered by the cluster projection) are more accurately sampled whereas other "less important" directions are more coarsely sampled in the gradient computation.

8.2 Lightcut Visibility Evaluation

Lightcuts is a local algorithm designed for ray-tracing applications on the CPU and is therefore not well suited for current GPUs. Its main bottleneck so far is the evaluation of the visibility function between the light clusters and each eye sample. However, one can observe that the cluster refinement and traversal in the upper parts of the light tree globally affects the whole image in contrast to the lower levels of the tree where the lighting is computed sparsely on a local basis. For example all pixels need to perform a shadow test for the root node and the next 3 depth layers (i.e. 2^4-1 clusters) in the indirect light tree as shown in Fig. 6.18. Further, the number of eye samples evaluated for a specific cluster is monotonically decreasing with the depth in the light tree (see Fig. 6.18).

These observations suggest to combine global visibility computation for clusters in the upper light tree using for example shadow maps and local visibility updates for clusters/leaves in the lower light tree. To generate shadow maps we can make efficient use of the GPU whereas for local visibility evaluations we stick to raytracing (on the CPU). The question that remains is how to merge the two methods.

Let us introduce a cost function that chooses the "best" visibility algorithm for each cluster considering the whole image. By exploiting temporal coherence between successive frames we can use the recorded statistics from the previous frame of how many pixels evaluated a particular cluster in the light tree. Based on the number of pixels that evaluated the visibility for a cluster in the previous frame, a binary cost function decides whether to use raytracing in the sub-tree or to continue generating a shadow map for all pixels of the current frame. This process continues until we have found a cut through the light tree where all clusters below should be further computed locally via raytracing.

The decision function β that decides when it becomes more efficient to use raytracing in a sub-tree should take into account the complexity of the scene (i.e. number of geometric primitives) N_{prim} and the expected number of pixels N_{pix} to

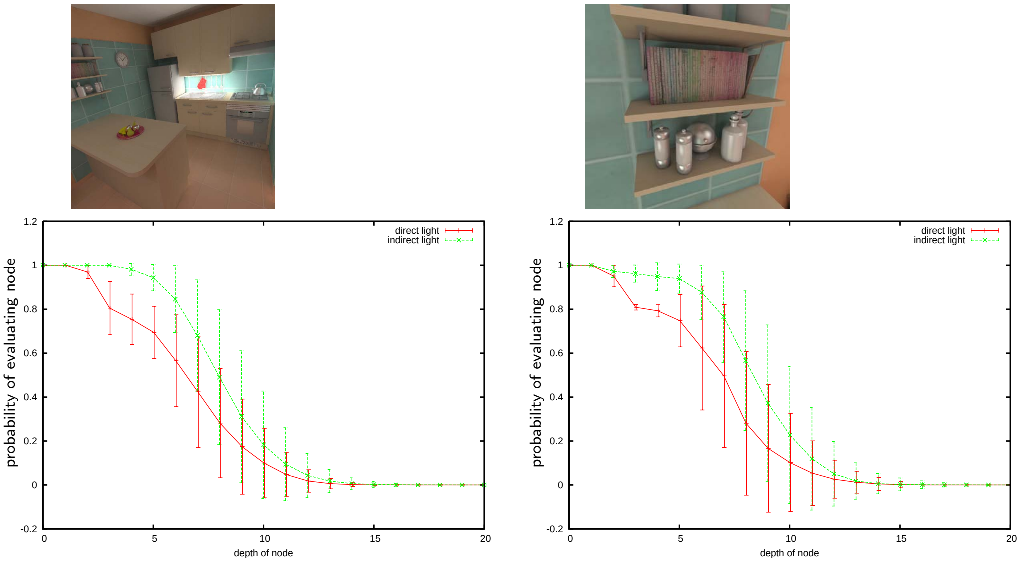


Fig. 6.18 – Statistics for visibility evaluations in lightcuts

Mean and standard deviation of visibility evaluations for cluster nodes in percent (over all computed cache records) with respect to depth in the light tree for 2 different view points in the Apartment *scene (first row), e.g. 0.5 at depth 8 means that on average only half of the cluster nodes with depth 8 in the light tree needed to evaluate the visibility where, depending on the camera view, different nodes with the same depth were evaluated more or less frequently as indicated by the standard deviation. The graph on the left is the result for a view showing almost the entire scene. The right graph corresponds to a close-up view showing only a small fraction of the scene. The indirect light tree contains 550000 VPLs and the direct light tree 10000 VPLs. The mean lightcut size is 752 clusters and 263 clusters for indirect light records and direct light records respectively (using a 1% error threshold).*

evaluate for a cluster $\mathcal{C}_l$.

$$\beta(\mathcal{C}_l) = \begin{cases} \text{true}, & \frac{C_{RT}(N_{prim}, N_{pix}(\mathcal{C}_l))}{C_{SM}(N_{prim})} < 1 \\ \text{false}, & \text{otherwise.} \end{cases} \tag{6.39}$$

where we define the shadow mapping cost as $C_{SM}(N_{prim}) = a \cdot N_{prim}$ (neglecting the dependence on shadow map resolution and shadow testing) and the ray tracing cost as $C_{RT}(N_{prim}, N_{pix}(\mathcal{C}_l)) = b \cdot \log(N_{prim}) \cdot N_{pix}(\mathcal{C}_l)$, with the ratio $\delta := a/b$ to be either set and tuned by the user or determined automatically by initial measurements. Since $N_{pix}(\mathcal{C}_l)$ is monotonically decreasing, the function β is monotonically increasing and we can stop evaluations of β in sub-trees where $\beta(L_C) = \text{true}$.

Another advantage of using shadow maps in the upper light tree is that we can compute gradients of the visibility function for all most significant light contributions in the computed lightcut.

8.3 Efficiency of Radiance Caching

Radiance caching works well on moderately glossy surfaces with a relatively small curvature. However, on highly curved and glossy surfaces radiance caching can become sub-optimal since cache records are densely spaced and need higher order spherical harmonics coefficients to faithfully reproduce the reflected lighting, which in turn increases memory and computational overhead. Also, for aligning the local coordinate frames of overlapping cache footprints on curved surfaces, a spherical harmonics rotation needs to be performed for each contributing cache record at each eye sample. The proposed approximative SH rotation [KŎ5b] is limited to small rotation angles and further increases the error on surfaces with high curvature. On the other hand, due to the lightcuts algorithm, the computation of the global illumination is relatively cheap compared with traditional approaches [KGPB05, WH92] and we can afford a higher cache density on curved surfaces. However, because our radiance caching is view-independent, on glossy surfaces we loose the advantage of lightcuts to automatically adapt to material properties of a surface. In other words, it means that the glossier the surface the more expensive is the computation and the memory consumption of a cache record in contrast to the per-pixel lightcuts algorithm. Hence, the question is when is radiance caching still beneficial and when is it better to perform pixel-exact computations without caching (or with radiance interpolation in screen space, see reconstruction cuts [WFA*05]).

8.4 Rotational Gradient versus SH Coefficient Rotation

The approximative (hemi-)spherical harmonics (SH) rotation proposed in [Kǒ5a] significantly improves the performance of the final radiance cache interpolation without noticeable errors. However, even if it is more efficient than traditional SH rotation approaches (quadratic instead of cubic complexity), it still involves complex computations for each cache record to eye sample splat, which can be considerable for high-resolution images with dense pixel super-sampling, in particular that we would like to keep the caching algorithm's complexity relatively independent of the image resolution. On the other hand, the approximative SH rotation [Kǒ5a] can also be thought off as a higher order interpolation with rotational gradients. Rotational gradients are computed "almost for free" in our algorithm, but are less precise for larger local coordinate rotations. However, even with rotational SH radiance gradients we would need to align the local coordinate frames, which would at least require a SH rotation around the surface normal.

For simple BRDF models, such as the Phong model, the shape of the BRDF lobe is invariant under view rotations. Therefore, we can rotate the BRDF SH coefficients instead of the radiance SH coefficients and avoid storing the BRDF SH coefficients for each discrete viewing direction. Or, because spherical harmonics are rotational invariant, another more efficient way would be to simply rotate the view vector at the eye sample into the local coordinate frame at each contributing cache record and looking up the corresponding BRDF SH coefficients. The final pixel radiance is computed by summing all individual contributions for each cache record to eye sample pair. Hence, an expensive rotation of the cached radiance SH coefficients to the local coordinate frame at the eye sample becomes redundant.

	Method	N_c	$a_\parallel/a_\perp$	a (converged)	FGRs	T_{light}	$T_\int$	T_{cache}	$\sum T$
Knot	PM	12479(0%)	–/0.9	–/0.35	3000(48)	3.4	328	1.0	331
	LCache	10177(0%)	–/1.0	–/0.33	–/678	3.1+8.3	26	0.7	28
	LCacheIso	34309(69%)	1.0/1.0	0.09/0.32	77/679	3.2+8.4	33	1.5	36
	LCache	33231(70%)	0.86/1.0	0.08/0.31	76/658	3.2+8.4	30	1.2	32
	LC	–	–	–	193	3.2+8.4	–	–	235
	$N_{prim} = 2936$, $N_V = 405{,}000$, $N_{glossy} = 43\%$, $\varepsilon = 0.1/0.4$, $\mu = 2.0$								
Sibenik	PM	34467(0%)	–/1.0	–/0.25	3000(30)	17.5	4697	2.0	4701
	LCache	35944(0%)	–/0.93	–/0.25	–/886	17.5+49	107	2.2	111
	LCache	109518(71%)	0.74/0.65	0.09/0.24	123/600	7.5+50	94	3.3	98
	LC	–	–	–	279	7.4+50	–	–	522
	$N_{prim} = 80394$, $N_V = 560{,}000$, $N_{glossy} = 0\%$, $\varepsilon = 0.1/0.25$, $\mu = 1.3$								
Kitchen	PM	15452(0%)	–/1.2	–/0.37	3000(29)	7.8	2686	5.0	2695
	LCache	15113(0%)	–/1.1	–/0.36	–/679	15.1+93	33	0.8	35
	LCache	39510(67%)	1.12/1.02	0.09/0.35	179/595	7.5+96	42	0.9	45
	LC	–	–	–	279	7.4+96	–	–	608
	$N_{prim} = 72365$, $N_V = 600{,}000$, $N_{glossy} = 28\%$, $\varepsilon = 0.1/0.4$, $\mu = 1.3$								
Icido	PM	14613(0%)	–/1.3	–/0.26	5000(35)	8.9	2001	7.6	2010
	LCache	18063(0%)	–/1.3	–/0.29	–/493	8.6+34	19	4.2	24
	LCacheIso	38991(69%)	1.0/1.0	0.08/0.30	149/385	4.1+34	21	8.9	31
	LCache	36520(69%)	0.65/1.1	0.08/0.30	150/381	4.1+34	23	4.7	29
	LC	–	–	–	260	4.1+34	–	–	291
	$N_{prim} = 199835$, $N_V = 515{,}000$, $N_{glossy} = 10\%$, $\varepsilon = 0.1/0.4$, $\mu = 1.3$								
Apartment	PM	20996(0%)	–/1.08	–/0.38	4000(25)	7.8	1189	1.1	1193
	LCache	18063(0%)	–/1.09	–/0.37	–/851	15+92	61	0.8	63
	LCache	45661(67%)	0.92/1.02	0.1/0.37	263/798	7.5+96	79	1.5	82
	LC	–	–	–	260	7.4+96	–	–	467
	LCwalk	3325(65%)	1.1/1.5	0.08/0.27	148/738	(7.3+95)	12	4.7	18
	LCwalk	1237(0%)	–/1.6	–/0.26	–/826	(15+92)	10	3.0	14
	$N_{prim} = 72365$, $N_V = 600{,}000$, $N_{glossy} = 62\%$, $\varepsilon = 0.1/0.4$, $\mu = 1.3$								

Table 6.1 – Statistics for radiance cache splatting

Statistics and computation times (in seconds) for the rendering with our algorithm and isotropic cache splatting (LCacheIso), anisotropic cache splatting (LCache) , lightcuts (LC) with 2% error threshold, and photonmapping with radiance caching (PM) for 5 scenes shown in Fig. 6.15 and Fig. 6.8. For the walk-through (LCwalk, 281 frames) computed with LCache the statistics for the average frame are shown. Image resolution is 512×512 pixels. N_c is the number of computed cache records, the fraction of direct light records is shown in percent. $\frac{a_\parallel}{a_\perp}$ is the mean anisotropy of the converged cache records for direct/indirect light, column FGR shows the average lightcut size for direct/indirect light or in the case of photon mapping the number of final gather rays and average found nearest neighbor photons, a is the average converged cache error for direct/indirect light in percent, T_{light} is the time for the light pass including photon shooting plus light-tree construction, $T_\int$ is the cache computation time, T_{cache} is the time for cache splatting, T_Σ is the total rendering time without the initial T_{light}. N_{prim} is the number of scene primitives, N_V is the number of VPLs/photons, N_{glossy} is the fraction of glossy cache records, which is approximately the same for all methods, ε is the global cache error for direct/indirect light (initial a), and μ is the initial anisotropy scaling parameter.

Render2mpeg: A Perception-Based Framework Towards Integrating Rendering and Video Compression

1 Introduction

In this chapter we propose a framework combining realistic rendering and MPEG video compression. Our rendering is based on the instant global illumination algorithm [Kel97,WKB*02] combined with the lightcuts data structure [WFA*05, WABG06], which we have introduced in the previous chapter (see Chapter 6). We extend these techniques to take advantage of temporal coherence between subsequent frames. The rendering quality is guided by a DCT-based quality metric, which maintains the rendering errors below the visibility level imposed by the quantization errors. At the same time the quantization errors are adapted to local image content to make the compression errors uniformly perceivable across the image space. Our prototype system shows many advantages of combining rendering and compression into a unified framework such as faster rendering and reduction of temporal artifacts.

A server-based platform with the client access through the Internet becomes a very attractive approach to access and interact with 3D graphics. To reduce the dependence on restricted computational capabilities on the client side and to simplify handling the diversity of client devices, the server can render images and stream them to the client side using a standard video compression technique. This way the client does not require any support for rendering and a standard web browser may be sufficient for video playback and backward interaction with a 3D model on the server. This is in particular important for applications that require huge 3D data, which cannot be handled by thin clients such as smart phones, PDAs, or even laptops. In such a scenario the 3D data is stored in one reliable place (possibly with guaranteed full-time access) and does not have to be transmitted to the client, which improves interaction and simplifies control over confidential data. This also ensures that in applications involving collaborative work all users always deal with the same, fully updated 3D models.

Such client-server graphics platforms are available on the market, e.g., Reality-Server developed by the *mental images*, Inc. company. The range of possible applications for such systems can be such diverse as: remote access to 3D data (e.g., for maintenance and repair purposes), industrial and architectural design, 3D navigation and tourism, interactive online (mobile) entertainment, and others. Such a client-server architecture can also be attractive in medical applications requiring 3D visualization and data access at any time and location.

To make such solutions practical and reduce the required bandwidth for video streaming, the MPEG and Motion JPEG encoding standards are used. However, in existing solutions the compression step is completely independent from the preceding rendering step. In this work, we demonstrate that by closer coupling of these two steps the computation redundancy can be reduced and the rendering quality can be well matched to the video quality, so the benefits of such coupling can be mutual.

The most obvious benefits come from the use of motion compensation vectors which, at the rendering stage, are inexpensive to derive for every pixel with very high accuracy [MB95]. On the compression side, this eliminates costly search for motion compensation vectors for pixel blocks [WKC94], which is one of the major bottlenecks in video encoding. Also, erroneous motion vectors due to approximate search algorithms are eliminated and variable block-size motion compensation [Bov05, Chapter 6.5] does not introduce any significant computational overhead. On the rendering side, pixel-level motion compensation enables pixel shading re-computation for consistent scene sample points tracked for subsequent frames, which reduces temporal aliasing in particular for textured regions. Motion compensation can be used for proper simulation of camera shutter speed and resulting motion blur [HDMS03] as well as filtering in temporal domain. All these techniques suppress temporal rendering artifacts, which otherwise cannot be distinguished from image features by the encoder and are reproduced in video at expense of extra bandwidth [BG00].

In this work our focus is on the rendering accuracy control. It is current practice that frames are rendered with many details that are later discarded due to lossy video compression. This means that the rendering quality can often be reduced without affecting the final video quality, which may lead to faster rendering, which is important in interactive applications. In MPEG compression the video quality is controlled through varying the quantization of discrete cosine transform (DCT) coefficients (used for a pixel block representation), which effectively leads to the information loss. A good match between the compressed and rendered image quality can be achieved by imposing a stopping condition on rendering, such that further computation cannot contribute to visible pixel changes as they would be eliminated by the quantization anyway. This can be facilitated by using the DCT pixel block representation both at the rendering and compression stages [BM95]. Also, such a DCT representation enables an inexpensive incorporation of a perceptual image quality metric to rendering, which can additionally adapt the quantization error to the image content by modeling important characteristics of the human visual system (HVS) such as contrast sensitivity, luminance and contrast masking [FPSG97, BM98, WPG02].

However, our goal is quite different from previous work on perceptual rendering, because we accept visible quality degradation under the condition that similar degradation is introduced by the following lossy video compression. Thus, we intend to adaptively control the rendering quality to make it synchronized with the available bandwidth (effectively the quantization error), which is important in Internet applications where the quality of services (QoS) is not guaranteed.

In contrast to previous work on perception-based rendering, we do not struggle with proper estimation of scene lighting as required to account for local luminance adaptation and visual masking. This is a typical "chicken-and-egg" problem, where the lighting knowledge is required to take the full advantage of the visual model, which is actually supposed to steer the lighting computation. In our approach, by taking into account the frame-to-frame coherence in lighting, we obtain a good prediction of the lighting distribution in the subsequent frame, whose computation is steered by our visual model.

We briefly summarize our system in Section 2 and then outline the extensions of the lightcuts and instant radiosity techniques to fit them into our Render2MPEG framework in Section 3. In Section 4 we introduce a perceptual model, which we use to steer the global illumination computation. In Section 5 we present results obtained using our techniques. Finally, we suggest directions of future research.

2 Algorithm Overview

In the following we briefly describe the whole system, which is sketched in Fig. 7.1. Our renderer is based on the lightcuts algorithm [WFA*05, WABG06], which we introduced in Chapter 6 because it is about one to two orders of magnitude more efficient than equivalent final gathering approach based on photon mapping [Chr99, Jen01, KGPB05]. Nonetheless, per pixel computation is still too costly for efficient rendering in particular for higher resolution images. Therefore, we exploit the spatial coherence of the indirect illumination. In contrast to the interpolation of pixel radiance as proposed in the *reconstruction cuts* [WFA*05], we favor irradiance interpolation of the incident lighting in object space [WRC88, KGPB05]. This decouples the interpolation from the surface material and prevents interpolation errors for high-frequency textures and surface boundaries. Since irradiance caching works only robustly for smooth lighting, we separate the high-frequency lighting components. A common way is to compute direct and indirect light separately since direct lights mostly influence the whole image whereas the weaker indirect lights often have only local impact on the rendered image. Therefore, we compute for all pixels direct lighting as well as dynamic indirect lighting using standard instant radiosity techniques [Kel97] with shadow mapping. We generate parabolic shadow maps on the GPU and use them for per pixel accurate lighting computations.

Summarizing, our system works as follows: First, a uniform grid is updated for all dynamic objects, which is used as raytracing acceleration data structure. Next, photons are traced through the scene splatting their energy to a fixed set of virtual point lights (VPLs), which we refer to as anchor lights. In the case of the first frame we construct a light tree over the anchor lights as described in Section 3.1. Otherwise all internal nodes in the tree are updated as described in Section 3.1.1. Then we shoot all primary (eye) rays for the current frame using packet raytracing [WKB*02] for 16×16 rays corresponding to one MPEG macro-block and find correspondences in the previous frame by backward reprojection. This way we can fill our error-threshold map with the previously computed visibility thresholds as explained in Section 4.3, which are then used to control the accuracy of the global illumination computation for the current frame. After computing direct and dynamic indirect lighting for each pixel using shadow maps, we compute the indirect lighting based on irradiance caching. Note that our error-threshold map is intended to steer the lightcuts computation on a per pixel basis. However, when using irradiance caching a cache sample influences a large neighborhood of pixels, which are likely to have a different rendering error-threshold. As a remedy, we filter the error threshold for a cache by averaging the tolerable rendering errors over the cache's footprint in image space. For weighting the error thresholds we use a Gaussian filter kernel. Finally, we compute the new error threshold map as described in Section 4.2, which is then used for the next frame.

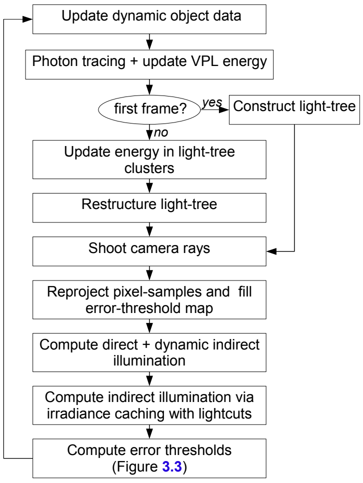

Update dynamic object data
Photon tracing + update VPL energy
first frame?
yes
Construct light-tree
no
Update energy in light-tree clusters
Restructure light-tree
Shoot camera rays
Reproject pixel-samples and fill error-threshold map
Compute direct + dynamic indirect illumination
Compute indirect illumination via irradiance caching with lightcuts
Compute error thresholds (Figure 3.3)

Fig. 7.1 – Pipeline of render2mpeg

Flow in the rendering pipeline for computing one frame.

3 Temporally Coherent Lighting Computation

Although temporally coherent global illumination is not the main focus of this work, for efficient video coding and motion compensation on MPEG side, we favor spatially and temporally coherent illumination in contrast to unbiased solutions [RPG99,BM98,BM95], which trade bias for high-frequency noise. Note that noise is well preserved in the DCT-based MPEG compression and leads to undesirable bandwidth expansion [BG00]. We chose a hierarchical version of instant radiosity [Kel97] based on *lightcuts* [WFA*05] because lightcuts is a pixel-adaptive, scalable global-illumination algorithm and perhaps more importantly, it allows us to control the computation by a perceptual rendering error. However, the original lightcuts algorithm has been developed for rendering still images [WFA*05]. In the temporal domain only short time intervals have been considered to model the effect of motion blur [WABG06]. In Chapter 6 we already introduces a version of lightcuts for walk-through animations in static scenes. However, in order to support longer animation sequences with fully dynamic scenes, we extend lightcuts into the temporal domain. This imposes several constraints to the algorithm in order to suppress temporal noise:

- A constant number of virtual point light sources (VPLs) in the scene are considered for the whole animation sequence.
- VPLs are sampled uniformly across the scene surfaces and their positions are fixed (similar to the anchor [SKDM05] and gather [HPB06] samples). We call those VPLs *anchor lights*.
- Only VPL intensities are updated from frame to frame using coherent photon splatting.
- The light hierarchy, which is built on top of the VPLs as in [WFA*05], is not rebuilt but only updated from frame to frame.

During computation of the pixel radiance only a small fraction of anchor lights is evaluated per pixel, which are chosen adaptively by computing a cut in the light hierarchy. While this lightcut is computed as in [WFA*05], the pixel error thresholds that determine the cut size adapt to the HVS and the quantization error imposed by MPEG. For details about the computation of the lightcut see Section 2 in Chapter 6.

3.1 Construction of a Light Hierarchy

As already discussed in Chapter 6, for the original lightcuts algorithm [WFA*05], the light hierarchy is constructed only once using a costly greedy bottom-up construction to cluster any two virtual point lights (VPLs) at a time that minimize a cost function (e.g., volume of associated bounding box and bounding cone weighted by their summed energy). See [WFA*05] for more details. An alternative approach using a recursive top-down construction of the point-light hierarchy is much more efficient and easier to implement but, according to our experiments, decreases the rendering performance by ca. 3% – 15% due to larger lightcuts arising from less precise error bounds. For the dynamic case, we need to be able to reconstruct and update the hierarchy efficiently. This is feasible since we assume lighting and scene geometry is temporally coherent such that only little changes in the hierarchy have to be made per frame. Besides, it is desirable to keep the lighting temporally coherent since this reduces flickering, which increases also the efficiency of the MPEG compression. As in [WFA*05] a cluster node in the light hierarchy has two children and shares the geometric properties with one child node, the representative child. The representative child is always chosen stochastically with the probability proportional to the relative light source intensity. This is important because it ensures that the induced rendering errors of individual clusters are uncorrelated and do not accumulate (see Fig. 7.7b). Because VPLs on dynamic objects violate our assumptions about temporally coherent lighting, they are not inserted into the light hierarchy but are handled separately as in standard instant radiosity [WKB*02, Kel97].

3.1.1 Updating the Light Hierarchy

At first, the intensity of all anchor lights at the leaf level of the hierarchy is updated by photon density estimation. In order to maintain temporal coherence, the raytraced photon paths are regenerated for successive frames using the same Halton number sequence when performing the random walk, i.e., in the case of only static objects no lighting changes will appear. Even in the dynamic case the energy of most anchor lights changes smoothly in time and abrupt lighting changes are rather of local nature. Therefore updates in the light tree structure and intensity affect mostly the lower levels in the tree and propagate slowly up the hierarchy, which further suppresses temporal noise in the radiance computation. Although changes in the light hierarchy are minor between successive frames, they may accumulate in the long run and eventually require a reconstruction of the entire hierarchy. Since higher levels in the light tree keep the sum of the intensities of their sub-trees, intensity changes in the anchor lights at the leaf level are simply propagated up to the cluster nodes of the hierarchy.

In the original lightcuts approach, the tree is built such that the VPL with highest energy is most likely to be at the top of the hierarchy, which cannot be ensured if we keep the tree static. In case of occlusion, e.g., a dynamic object moves in front of a cluster's representative anchor light, the corresponding node should descent the hierarchy according to the tree construction metric [WFA*05]. To model this behavior, we swap the two children of a node in the light tree if the intensity ratio of representative child to non-representative child is smaller than a threshold ($C_I = 0.7$), thus changing the representative/non-representative relationship of parent and child nodes. Only pointers to the representative child and anchor light need to be updated in a bottom-up manner. Therefore the overall tree structure is kept static and updating the tree becomes very efficient since the tree continuously reorganizes itself over time.

4 Rendering Accuracy Control

Aligning the rendering errors with the compression errors as imposed by the bandwidth of a given video streaming channel is an important goal of our Render2MPEG framework. A simple strategy is to keep the rendering errors below the quantization error imposed by the only lossy operation in video encoding: the quantization of DCT coefficients. For the 1D case this is illustrated in Fig. 7.2. Such a quantization error is determined for each DCT basis function used to represent pixel values in 8×8 pixel blocks. For a given video-stream bandwidth controlled by the value of the q_{scale} parameter, the quantization error amounts to the product of $q_{scale} \cdot q_{ij}$, where q_{ij} are predefined coefficients in the 8×8 quantization matrix Q. In our current implementation we always use the default quantization matrix Q for intra-frame coded blocks as specified in the MPEG-2 standard (ISO/IEC 13818) [MPE] and we let the user set q_{scale}.

While the matrix Q takes into account the contrast sensitivity function (CSF) attributed to the HVS (see Fig. 2.6), it does not adapt the quantization error to the frame's local content. It is well known that the quantization errors are less visible in cluttered image regions due to visual masking, which decreases the sensitivity for image details [BM95, FPSG97]. Also, the visual sensitivity in terms of detection of absolute luminance thresholds reduces in bright image regions due to luminance masking [Dal93]. We model these effects locally for each block, and we incorporate elevated sensitivity thresholds due to masking into the quantization error $q_{scale} \cdot q_{ij}$. This enables even more aggressive rendering. Note that while q_{scale} is a convenient parameter to control the compression bandwidth, it correlates poorly with the optimum visual quality that can be achieved at a given bit rate [Wat93].

This section is organized as follows. In Section 4.1 we present the derivation of quantization error with incorporated masking effects. In Section 4.2 we describe our approach to transform the resulting quantization error from the frequency domain into the spatial domain as required by our renderer. Obviously, the most

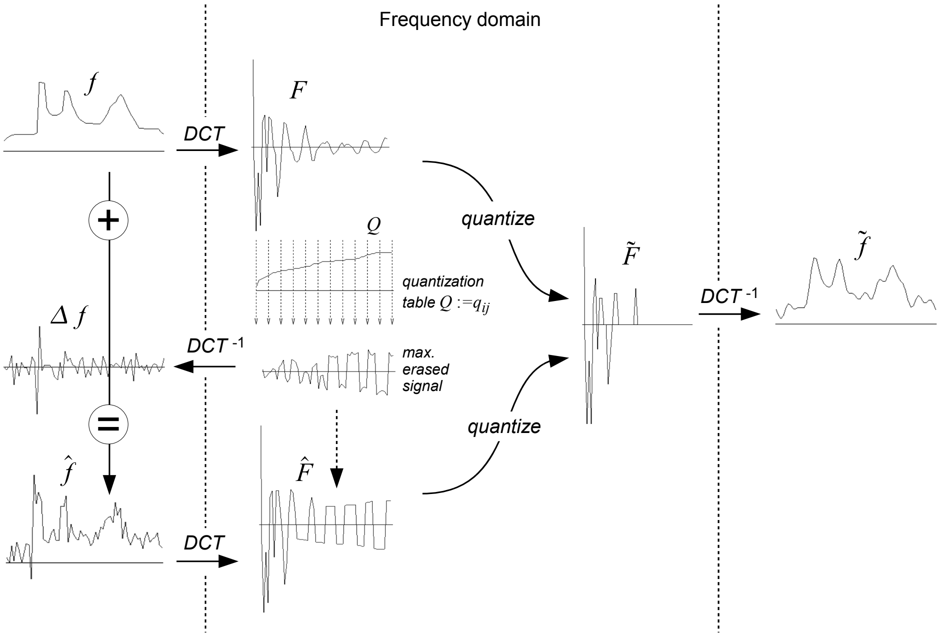


Fig. 7.2 – Quantization-dependent tolerable rendering error

An example showing the maximum possible rendering error Δf for a 1D signal f that leads to exactly the same result $\tilde{f}$ after compression with quantization table Q.

reliable estimate of quantization errors can be achieved when the image content is fully known. For this reason we apply the visual model to the previously computed frames in order to predict the quantization error to be used in the current frame rendering. We discuss various strategies of transferring the quantization error from frame to frame in Section 4.3.

4.1 Luminance and Contrast Masking Model

The visual model used by us to predict luminance and spatial masking is inspired by research in image and video compression [Wat93, ZDL00]. The *luminance masking* model as proposed in [Wat93] requires luminance values, which are dependent on the particular display characteristics such as the luminance range and gamma correction. Since the masking model needs to consider observed luminance of pixels on a particular display device, we have to consider the γ of such display (e.g. $\gamma = 2.0$) before applying the luminance masking model to our DCT coefficients. Since the luminance masking model proposed in [Wat93] is a simple power function, the display gamma is easily accounted for by multiplying γ with the luminance masking exponent (0.649) [Wat93].

$$t_{ij}^{k} = q_{ij} \cdot \left(\frac{c_{00}^{k}}{\bar{c}_{00}} \right)^{\gamma \cdot 0.649}, \tag{7.1}$$

where c_{00}^k denotes the DC coefficient of block k and $\bar{c}_{00}$ the average DC coefficient of all blocks. Intuitively, this luminance masking model increases the threshold of tolerable quantization errors q_{ij} for brighter image regions and decreases in darker regions as predicted by the Weber's law.

Then, we estimate the tolerable elevation of quantization error m_{ij}^k due to *contrast masking* [Wat93] as:

$$m_{ij}^k = t_{ij}^k \cdot \max\left(1, \left|\frac{c_{ij}^k}{t_{ij}^k}\right|^{0.7}\right), \tag{7.2}$$

where c_{ij}^k denotes the ij-th DCT coefficient of block k. Following [Wat93] we ignore contrast masking for the DC coefficient c_{00}^k, i.e., we assume that $m_{00}^k = t_{00}^k$. Intuitively, this contrast masking model increases the threshold of a tolerable quantization error for regions with high contrast patterns of a spatial frequency represented by the given coefficient c_{ij}^k.

The localized quantization error m_{ij}^k can be used to control rendering accuracy for each block in order to make the distribution of the perceptual error uniform across all blocks of the image. Note that at the compression stage we could also use the localized quantization error m_{ij}^k to modify the matrix Q for each block as proposed in [RW96], but at present optimizing video bandwidth is less important than speeding up rendering.

4.2 Maximum Tolerable Error in Rendering

From the rendering perspective the localized quantization error m_{ij}^k derived in Eq. 7.2 expresses the maximum tolerable rendering error as imposed by the quantization matrix and the local masking effects. This error must be then re-scaled by q_{scale} to mirror the user-imposed compression bandwidth. Since each block k is quantized by dividing all its coefficients c_{ij}^k by $m_{ij}^k \cdot q_{scale}$ and rounding to the nearest integer, the maximum possible quantization error is $\frac{1}{2} \cdot m_{ij}^k \cdot q_{scale}$. By exploiting the linearity property of the DCT transform and preserving the polarity of DCT coefficients using the $\mathrm{sign}(c_{ij}^k)$ function, we can conservatively construct the worst case distorted frame with DCT coefficients $\hat{c}_{ij}^k$ by adding the maximum possible quantization error to the original frame:

$$\hat{c}_{ij}^k = \begin{cases} \frac{1}{2}\mathrm{sign}(c_{ij}^k) \cdot q_{scale} \cdot \min\limits_{ij}(q_{ij}) & \text{if } c_{ij}^k < \frac{1}{2} q_{ij} \cdot q_{scale} \\ c_{ij}^k + \frac{1}{2}\mathrm{sign}(c_{ij}^k) \cdot q_{scale} \cdot m_{ij}^k & \text{otherwise,} \end{cases} \tag{7.3}$$

which after compression should be visually equivalent to the original frame (c_{ij}^k) when both are compressed with the same q_{scale} value.

When computing $\hat{c}_{ij}^k$ for the coefficients c_{ij}^k that do not vanish as the result of lossy compression (case 2 in Eq. 7.3), the quantization error including the masking effects are considered. However, the signal represented by the remaining

c_{ij}^k coefficients (case 1 in Eq. 7.3) is removed from the compressed image (i.e. $\tilde{c}_{ij}^k = 0$) and therefore contrast masking for the corresponding frequencies as predicted by Eq. 7.2 is not valid. In this case, even considering only the quantization error resulting purely from the MPEG compression (i.e. $\hat{c}_{ij}^k := \frac{1}{2} q_{ij} \cdot q_{scale}$) may lead to overestimated errors and reduced rendering quality for higher q_{scale} values. Since the largest quantization errors q_{ij} are assigned to high frequency coefficients (which usually vanish as the result of lossy compression), they tend to dominate in the tolerable rendering error estimate. Because this error estimate is finally used as upper bound for the rendering error in the spatial domain (see Fig. 7.3), its spatial frequency selectivity inherent for the DCT domain is lost. Consequently, such high quantization errors may also contribute to excessive tolerance for errors in the lower frequency signals (higher eye sensitivity), which leads to visible image distortions. Note that low-frequency quantization errors may also generate high-frequency rendering errors, which is less critical since quantization errors increase for higher frequencies. One solution is to leave those coefficients that are invisible after quantization unaltered [WPG02]. However, such approach does not scale well with higher q_{scale} values where most frequencies are removed from the signal. Therefore, we set those coefficients to the minimum quantization for AC coefficients ($\min(q_{ij}) = 16$), which is adaptively tuned to the current video bandwidth by the q_{scale} multiplier. Although this approach may still be too conservative, it produces good results in terms of rendering efficiency and robustness.

After performing the inverse DCT on $\hat{c}_{ij}^k$ and inverse tone-mapping to get the distorted luminance $\hat{Y}_{xy}$ for every pixel (x,y), we compute the maximum tolerable pixel errors e_{xy}:

$$e_{xy} = \max(0.02 \cdot Y_{xy}, |\hat{Y}_{xy} - Y_{xy}|). \tag{7.4}$$

as the absolute difference between the luminance Y_{xy} of the original undistorted pixel and $\hat{Y}_{xy}$. In order to avoid too small error thresholds, we clamp the result at a “perceptually conservative” lower bound of 2% of the pixel’s luminance Y_{xy} [WFA*05].

The error e_{xy} can change from frame to frame as a function of image content, but also can be affected by variable network bandwidth. In the following section we discuss temporal aspects of handling e_{xy}.

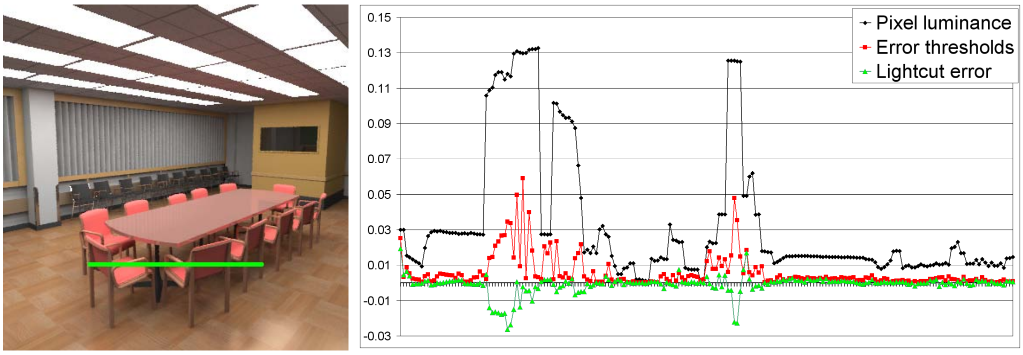


Fig. 7.3 – 1D Plot of rendering error, absolute error thresholds, and image signal along a scanline

Right image: pixel luminance (black), our imposed error thresholds (red), and actual rendering errors (green) along a scanline in the left image (green line).

4.3 Temporal Handling of Tolerable Rendering Error

In this section we discuss the problem of re-using the maximum tolerable error e_{xy} from the previous frames to steer the current frame computation. Another important issue is temporal coherence of the error between subsequent frames, which is necessary to reduce video flickering. Such temporal coherence is improved by blending the tolerable error between previous $(t-1)$ and current frame t, such that the blending weights fall-off exponentially for older frames. The final rendering error-thresholds that are then stored in the threshold map are computed as:

$$\Delta Y_{xy}(t) = \begin{cases} e_{xy} & \text{if } (x,y) \not\mapsto (x',y') \\ (1-w)e_{xy} + w\Delta Y_{x'y'}(t-1) & \text{otherwise,} \end{cases} \tag{7.5}$$

where the mapping $(x,y) \mapsto (x',y')$ to the corresponding error threshold in the previous frame is obtained through back-projecting the current frame's pixel sample (effectively the camera and object motion compensation is performed). See Fig. 6.14 for example. When occluded/disoccluded/non-existing region in frame $(t-1)$ is identified by the backprojection (case $(x,y) \not\mapsto (x',y')$) then we simply use the error e_{xy} estimated for frame t. The blending weight w is a trade-off between temporal coherence and error propagation. When using a larger blending weight our error thresholds can become less accurate for successive frames while lighting errors are kept coherent and vice versa. We set $w = 0.5$ which, according to our tests, does not seem to bias the error in our threshold map for future frames.

The resulting error thresholds $\Delta Y_{xy}(t+1)$ are then used for the future frame $(t+1)$ to decide upon the stopping condition in the lightcut computation: if the maximum upper error bound of all clusters in the pixel's current lightcut is below our error threshold or the lightcut size exceeds a maximum of 1,000 clusters, we stop refining the lightcut. The lightcut is computed as in [WFA*05] using a priority queue. A lightcut sample for one pixel with 3% error threshold is shown in Fig. 7.7b. Error thresholds and actual lightcut rendering-errors for a scanline are visualized in Fig. 7.3. Note that rendering errors (green curve) alternate around zero but are coherent for neighboring pixels with similar error threshold.

The entire flow of the threshold computation is shown in Fig. 7.4.

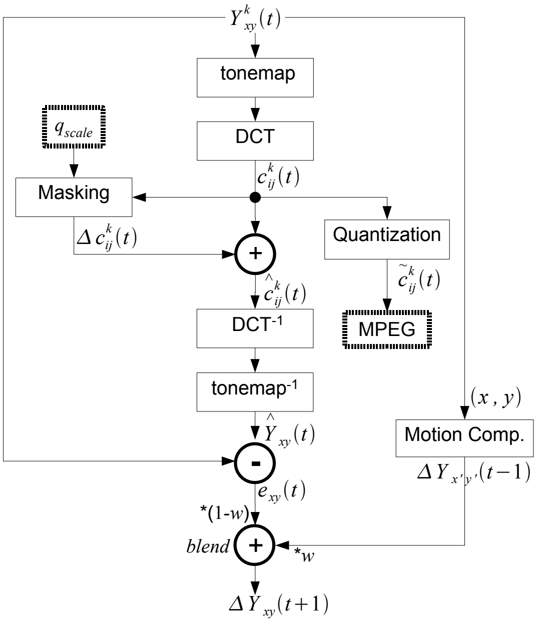


Fig. 7.4 – Flow chart for computing error thresholds in render2mpeg

The computation flow for deriving error thresholds $\Delta Y_{xy}(t)$ *for all pixels* (x,y) *in the frame computed at time* t. $\Delta Y_{xy}(t+1)$ *is used to steer rendering of the subsequent frame at time* $(t+1)$.

We applied the processing flow as shown in Fig. 7.4 to two basic strategies of the error handling with respect to the previous frames:

1. Quantization-*dependent* masking *without* motion compensation.
2. Quantization-*independent* masking *with* motion compensation.

The first case means applying the masking model in Eq. 7.1 and Eq. 7.2 to the quantized DCT coefficients $\tilde{c}^k_{ij}$, thus also considering quantization errors in the masking prediction, which results in more relaxed rendering errors. This is because quantization errors in the final image such as blocking artifacts are also included in the contrast masking. This approach is only valid for scenes with slow camera and object motions where block boundaries from the previous frame are aligned with blocks in the current frame.

The second approach is more conservative when computing the error thresholds at the expense of having smaller rendering errors in particular for higher MPEG compression settings, where masking due to quantization dominates. In this case the DCT coefficients c_{ij}^k are directly used to predict masking as it is shown in Eq. 7.1 and Eq. 7.2. Motion compensation helps in aligning masking with image details and even significant motion of camera and rigid objects can be successfully handled. The same motion compensation procedure as between frames $(t-1)$ and (t) is now applied to frames (t) and $(t+1)$. For pixels that do not find a feasible sample in the four nearest neighbor pixels of the previous frame, a maximum relative rendering error of 2% is assumed. A pixel sample is assumed to be feasible if its pixel depth and surface normal are similar. An example of motion compensated thresholds is shown in Fig. 6.14.

5 Results

We have tested our framework on 4 scenes with different lighting, frequency content, and complexity. For visualization and validation purposes the results of the CONFERENCE scene, shown in Fig. 7.5, are computed at a per pixel basis in order to compare with the original lightcuts algorithm [WFA*05] that uses a maximum luminance error per pixel of 2%, which we have used as the reference in our evaluations. Since the original lightcut algorithm has been designed for high-quality rendering with a large number of point light sources (VPLs), we compare our error metric also using a larger number (150,000) of VPLs in the CONFERENCE scene. The other scenes, CORNELL BOX, SPONZA ATRIUM, SIBENIK CATHEDRAL, were generated in a dynamic scenario and the videos are provided at [Web08]. The relevant statistics for the animated scenes shown in Fig. 7.6 are given in Table 7.1.

To speedup rendering, the irradiance caching algorithm [WRC88, KGPB05] has been used. Note that for efficiency reasons the MPEG-2 encoder [MPE] uses simple integer arithmetic in the quantization. Therefore, the provided q_{scale} values in the results, which are given as input to the encoder, should be divided by 16 to corresponds to the actual quantization errors used in Section 4.2.

Our numerical results were computed on a single PC equipped with an Athlon 64 2.4 GHz with 4 GByte of memory and a GeForce 6800 based graphics card. The statistics in Table 7.1 show (from left to right): scene settings including number of direct and indirect anchor lights (VPLs) and the number of photon splats to update the indirect VPL intensities, the q_{scale} quantization multiplier given by the MPEG encoder, the average relative error-threshold per frame (i.e. divided by pixel luminance), the average number of swapped cluster nodes in the light tree per frame, the average lightcut size with respect to the total number of anchor lights (VPLs), the average computation times in seconds per frame for photon tracing and energy splatting to anchor lights (T_{light}), for computing the shadow maps for direct and dynamic indirect light on the GPU (T_{gpu}), for primary (eye) ray casting (T_{eye}), for the lighting computation including lightcut evaluation and

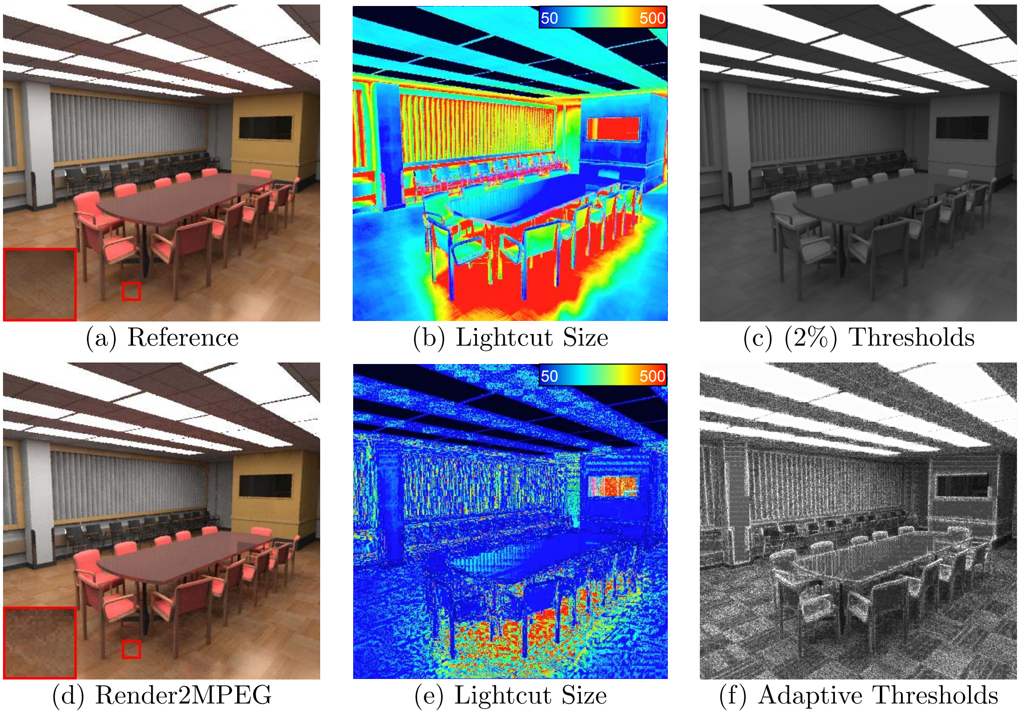
Scene+Setting	q_{scale}	$\frac{\Delta Y_{xy}}{Y_{xy}}$	LC-tree changes	LC size	T_{light}	T_{gpu}	T_{eye}	T_{LC}	Time per frame	Speedup only LC
CORNELLBOX	2	7.9%	1.1%	8.0%	2.5	0.8	0.1	3.0	6.6	×1.5
2000 VPLs	48	14.4%	1.1%	2.7%	2.5	0.8	0.1	1.6	5.2	×3.1
$3{\cdot}10^5$ *photons*	-	2.0%	1.1%	10.9%	2.5	0.8	0.1	4.0	7.5	×1.0
SPONZA	2	5.7%	2.4%	5.1%	14.2	1.1	0.5	19.0	35.0	×1.9
8900 VPLs	48	12.9%	2.4%	0.9%	14.2	1.1	0.5	9.1	25.1	×5.2
10^6 *photons*	-	2.0%	2.4%	6.8%	14.2	1.1	0.5	34.5	50.4	×1.0
SIBENIK	2	7.0%	0.0%	5.5%	(40.0)	0.0	0.7	6.5	7.4	×1.7
9100 VPLs	48	15.4%	0.0%	1.6%	(40.0)	0.0	0.7	4.4	5.3	×3.3
10^6 *photons*	-	2.0%	0.0%	7.4%	(40.0)	0.0	0.7	12.4	13.2	×1.0

Table 7.1 – Statistics and timings for render2mpeg

Statistics and computation times (in seconds) for the rendering phases of our algorithm for the 3 animated test scenes shown in Fig. 7.6. Image resolution is 512×512*. The shown speedup is only for the lightcut (LC) computation without irradiance caching.*

irradiance caching (T_{cut}), the average total rendering time in seconds per frame, and the speedup of the indirect lighting computation relative to the reference solution, which uses a constant error threshold of 2% of the pixels luminance value (see Fig. 7.5c for an example). Our error-threshold computation including a discrete cosine transforms of all blocks and motion compensation takes 0.2 sec per frame for all three animations. The memory utilization is less than 50 MBytes for the tested scenes. Since the SIBENIK scene is shown in a walk-through animation, lighting changes and updates of the light tree are not necessary. The only cost is the initial lighting computation and the light tree construction. The main bottleneck so far is the random photon walk (T_{light}), which uses the same data structure (uniform grid) for Monte Carlo ray tracing that is actually optimized for packet-ray tracing of primary eye rays (T_{eye}).

In Fig. 7.7a we show per pixel statistics of rendering/compression error and relative lightcut (LC) size for the textured CONFERENCE scene shown in Fig. 7.5 for various compression settings. The lightcut size (black dashed curve) is given relative to the reference using 281 clusters on average. It directly mirrors the saving in computation time for this particular scene, which range from approximately 30% to 60% with increasing q_{scale}. Fig. 7.7b shows the upper bounds and real lightcut errors for one pixel with a cut size of 253 and 3% error threshold. Note that the computed upper error bounds (blue) are always greater than the real cluster errors (red) but not necessarily greater than the sum over the individual cluster errors (green), whose distribution is presented in Fig. 7.8a. Fig. 7.8a shows that for the vast majority of pixels the rendering errors introduced by the



(a) Reference (b) Lightcut Size (c) (2%) Thresholds

(d) Render2MPEG (e) Lightcut Size (f) Adaptive Thresholds

Fig. 7.5 – Visual statistics and results of original lightcuts versus render2mpeg

Comparing original lightcuts sampling using 2% error metric (first row) with our MPEG-driven perceptual error metric (second row) where the rendering is adapted to compression level with $q_{scale} = 16$. *(MPEG-encoded frames are shown in Fig. 7.9.) The color-encoded number of considered lights per pixel is given in the second column with an average of 281 lights for the reference and 152 using our method, respectively. The error threshold maps (scaled by* 32 *and displayed with* $\gamma = 2.6$*) are shown in the third column. The average relative error threshold is 26% in our case.*

lightcut computation are smaller than their error thresholds (green points). Nevertheless, there are occasionally a few outliers (red points). Fortunately, outliers are mainly pixels that have a relatively small error-threshold as shown in the histogram in Fig. 7.8b, which is often too conservative anyway (see Eq. 7.4). These observations hold also for the other scenes we have tested.

In Fig. 7.9 we demonstrate how our rendering adapts to varying bit rates by means of the parameter q_{scale} provided by MPEG. In the first row the absolute error thresholds to be used for the next frame are shown. Below are given the numerical values for the average relative error-threshold $\frac{\Delta Y_{xy}}{Y_{xy}}$ in percent (i.e. absolute luminance threshold divided by pixel luminance). The second row shows our compressed rendering results using the thresholds above and their peak signal-to-noise ratio (PSNR), which are visually equivalent to the compressed reference images. We observed that even though the compressed images of reference and our solution differ slightly (blue curve in Fig. 7.7a), their RMS error (RMSE) with

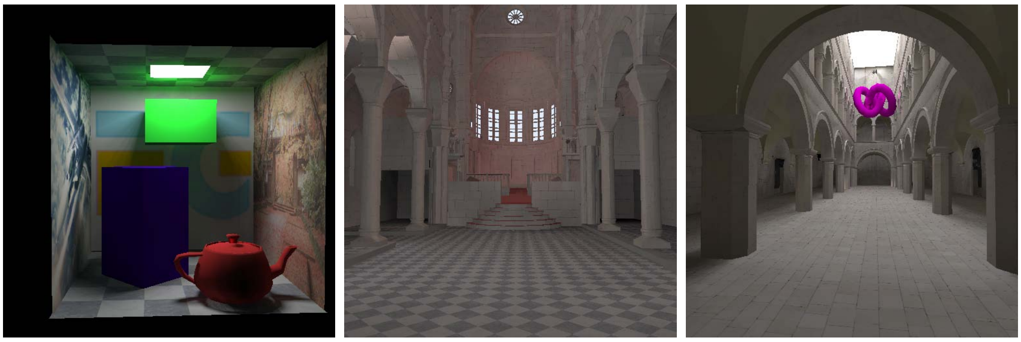
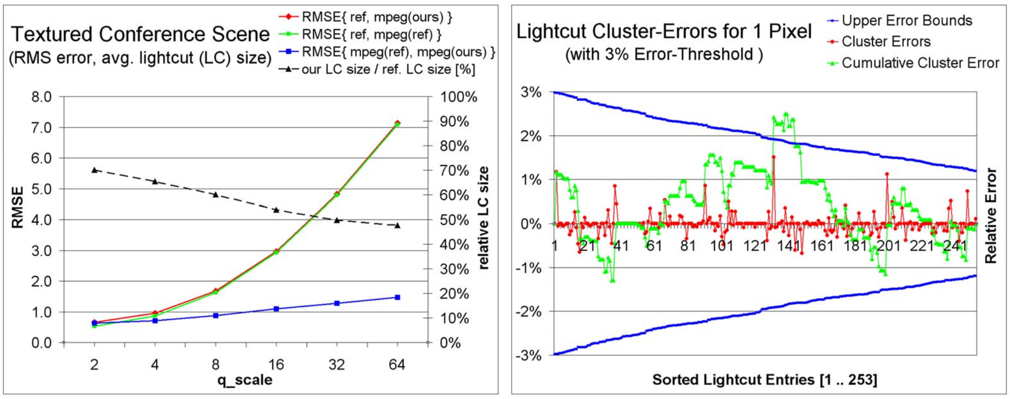


Fig. 7.6 – Animated 3D scenes for testing render2mpeg

Animated test scenes (left to right): textured CORNELL BOX*, indirectly lit* SIBENIK*,* SPONZA ATRIUM*. Videos are provided at [Web08].*

respect to the uncompressed reference is very similar (see red and green curves in Fig. 7.7a) because the quantized pixel values fluctuate around the estimated reference value.

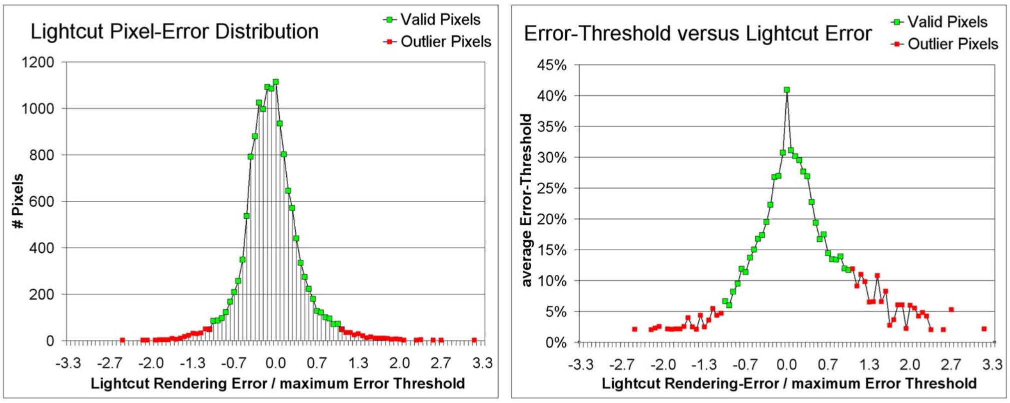


(a) (b)

Fig. 7.7 – Plots of numerical error and lighcut-size for varying MPEG quantization

(a) Root mean square errors (RMSE) and relative lightcut size for varying q_{scale} in the CONFERENCE *scene shown in Fig. 7.9. (b) Lightcut cluster-errors of a pixel in the image of the conference scene with an error threshold of 3%. The graph visualizes the sorted lightcut (size 253) with descending upper error bounds. The cumulative sum (green curve) of the individual cluster errors (red curve) shows the actual rendering error, which is close to 0 for this particular pixel (see point 253 of the green curve). Since the individual cluster errors can be considered as independent and identically-distributed random variables with finite variance, the actual rendering errors are approximately normal distributed (see Fig. 7.8a).*

The third row shows the residual (the rendering error) of the reference frame and our rendered frame before compression for visualization purposes scaled by a factor of 32. The PSNR and the average lightcut size per pixel are given below the images. One can observe that the rendering error does not scale in the same way as the compression error does. This is because the enforced error thresholds are upper bounds for the rendering error introduced by the lightcuts algorithm, which is usually much smaller for most pixels (see Fig. 7.8a).



(a) (b)

Fig. 7.8 – Histograms of the error distribution of lightcuts

(a) Histogram of the actual lightcut pixel-errors in the image of the conference scene ($q_{scale} = 32$). The x-axis represents the ratio of actual rendering error and our maximum tolerated error-thresholds for that pixel. The green points represent the valid pixels for which the rendering error is below our imposed error thresholds. Red points in the graph indicate outliers which have a higher actual error than tolerated. A plot of the lightcut-error for a single pixel is shown in Fig. 7.7b. (b) The average error threshold (y-axis) for the corresponding rendering error ratio (same domain as in chart (a)). Intuitively, this means the higher the pixel's error-threshold (y-axis), the more likely is that the rendering error will stay below its imposed error threshold (green points).

6 Discussion and Future Work

The system implementation serves as a proof of concept and there are a few limitations of our approach. The choice of MPEG-2 [MPE] instead of the more suitable for streaming MPEG-4 was driven by such factors as simplicity and availability of the source code. The choice of lightcuts imposes a few restrictions on the lighting computation such as clamping of close VPLs and missing caustics illumination. Besides, the separation of dynamic and static objects and the irradiance caching limits the generality of the lightcuts algorithm. The developed error thresholds for rendering are too conservative with respect to the MPEG compression error. We expect therefore that further investigations about the correlation between the rendering error of lightcuts and the quantization error in MPEG might enable even more aggressive rendering.

Even though our rendering is too slow for interactive streaming applications, we demonstrated that such properties as temporal coherence, low noise level, and local (pixel or irradiance cache level) accuracy control are desirable in Render2MPEG applications. By mapping our lightcuts algorithm to a multi-processor architecture as in [WKB*02] interactive performance should be feasible due to similarities between the instant global illumination and lightcuts techniques.

By closely coupling rendering errors with q_{scale} controlling the bandwidth of compressed streams, we can easily support two different encoding modes: CBR (constant bit rate, which leads to variable quality) and VBR (variable bit rate, which enables constant quality). Our error metric naturally adapts the rendering quality to the required quantization level no matter what encoding mode is used, although delayed by one frame.

Since our metric operates on tone mapped pixels, implicitly it takes into account tone mapping characteristics, which ideally should be adjusted to each display device and surrounding lighting conditions at the client side.

In the current implementation we decided to estimate the rendering error thresholds (refer to Section 4.2) for each frame. As future work we leave experimentation with computing the error map just for I-frames and propagating it along motion compensation vectors for P and B frames. It can be expected that more relaxed error estimates could be considered for P- and B-frames, but this also requires further studies.

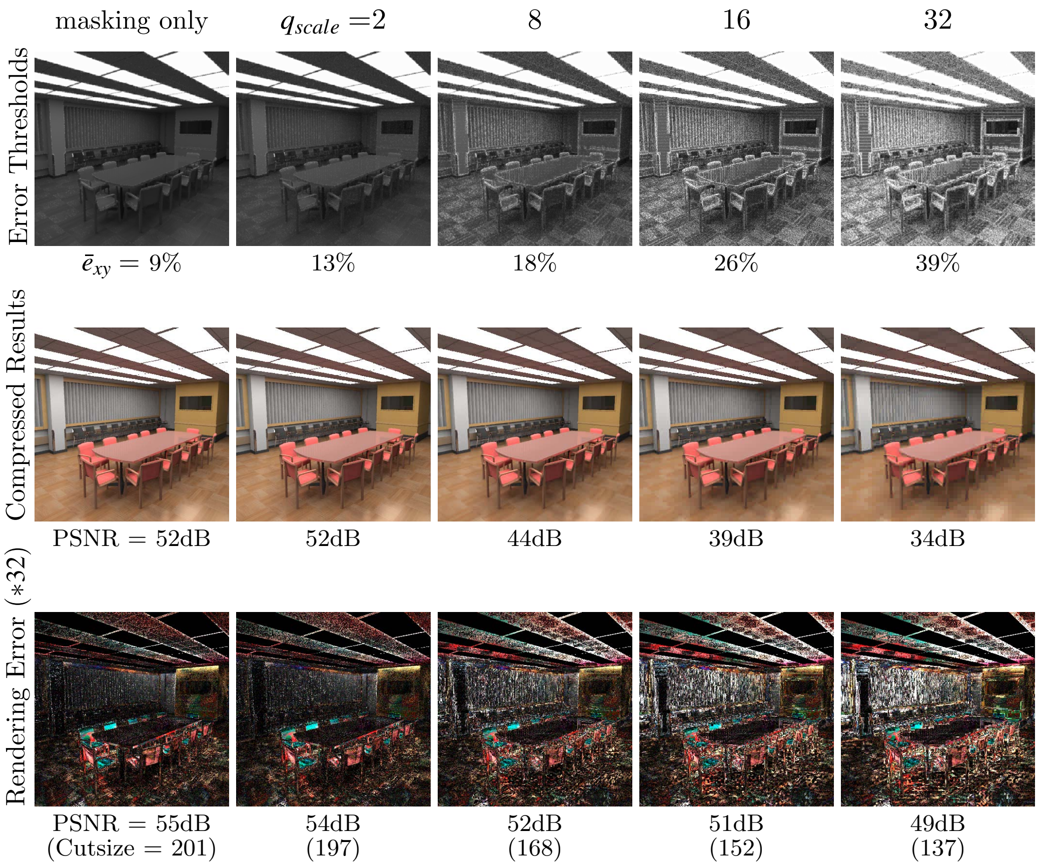


Fig. 7.9 – Visual results and error of render2mpeg for different MPEG quantization levels

Absolute rendering error thresholds (first row) for corresponding q_{scale} value (columns) driving our global illumination rendering and compression. The compressed images for varying q_{scale} with their peak signal-to-noise ratio (PSNR) are shown in the second row. While our compressed images stay visually equivalent to the compressed reference images, the rendering errors with respect to the reference solution increase steadily for higher q_{scale}. The absolute rendering error (difference between our rendered images and reference images) are shown in the third row (magnified by factor 32). The values in brackets correspond to the average resulting lightcut size.

Beyond Rendering – Spatio-Temporal Upsampling on the GPU

1 Introduction

Shader units are a substantial element of modern graphics cards and lead to significant visual improvements in real-time applications. Despite the tendency towards more general processing units, pixel shading receives a constantly increasing workload. Much visual detail today, such as shadows, ambient occlusion, procedural materials, or depth-of-field, can be attributed to pixel processing and a faster execution often leads to a direct performance increase.

With the current trend towards enhancing the image resolution in modern High Definition (HD) and forthcoming Super High Definition (SHD) imaging pipelines, one can observe that neighboring pixels in spatial and temporal domains become more and more similar. Exploiting such spatio-temporal coherence between frames to reduce rendering costs, suppress aliasing, and popping artifacts becomes more and more attractive.

Our method is driven by the observation that high quality is most important for static elements, thus we can accept some loss if strong differences occur. This has been shown to be a good assumption, recently exploited for shadow computations [SJW07]. To achieve our goal, we rely on a temporally varying sampling pattern producing a low-resolution image and keep several such samples over time. Our idea is to integrate all these samples in a unified manner.

The heart of our method is a filtering strategy that combines samples in space and time, where the time and spatial kernel can be adapted according to the samples' coherence. For static configurations, the time window can be chosen to be large to produce a high-quality frame. When drastic changes occur, our method automatically favors consistent spatial samples. The result loses some of the visual accuracy, but maintains temporal consistency.

A significant property of our algorithm is locality, meaning that a good filtering strategy is chosen according to the actual image content. Here, we differ from other recent strategies, such as [YSL08]. Although our method's overhead is small, we achieve higher quality. Our approach runs entirely on the GPU, leaving the CPU idle for other purposes which is important, e.g., for games.

2 Upsampling

In this section, we will explain our upsampling strategy. As it is inspired by previous work, we will first review spatial upsampling (Section 2.1) then temporally-amortized spatial upsampling (Section 2.2), also referred to as *reprojection caching*. Step by step, we will describe our modifications before presenting our spatio-temporal solution (Section 2.3).

2.1 Spatial Upsampling

Yang et al. [YSL08] assume that expensive shader computations are spatially slowly varying and can be reconstructed by sparse sampling followed by interpolation. This is true for many low-frequency shaders, e.g., indirect lighting, ambient occlusion. However, normally shading correlates with geometry in 3D world space, which is only partially captured in 2D image space. This means that pixel samples which are close in world space and have similar surface orientation are better interpolation candidates. Consequently, the authors want to preserve geometric discontinuities in the interpolation and achieve this with a spatial joint-bilateral filter [ED04,PSA*04] to perform the image upsampling [KCLU07]. Thereby, the filter weights are steered by geometric similarity computed from a high-resolution *geometry buffer*, which encodes 3D position and surface orientation. As opposed to "blind" image processing, we not need to estimate the geometry but get it for free as a by-product of the rendering pipeline. Furthermore, we are not only restricted to upsampling of the final result, the pixel colors, but can compute and then upsample any intermediate shading result per pixel, which we call the *payload*.

For simplicity, we will use 1D pixel indices i or j in the following derivations. Given the high-resolution geometry buffer and the low-resolution shading result l, the upsampled payload $h(i)$ can be computed as

$$h(i) = \frac{1}{\sum w_s} \sum_{j \in N\{i\}} w_s(i,j) \cdot l(\hat{j}), \tag{8.1}$$

where $N\{i\}$ is a neighborhood around i, $\hat{j}$ is the index of the nearest pixel in the low-resolution image and $w_s(i,j)$ is a spatial geometry-aware pixel weight defined as:

$$\begin{aligned} w_s(i,j) &= n(\max(0, 1-(\vec{\mathbf{n}}_i \bullet \vec{\mathbf{n}}_j))^2, \sigma_n^2) \cdot \\ &\quad d(|z_i - z_j|^2, \sigma_z^2) \cdot k(i,j) \end{aligned} \tag{8.2}$$

The weight consists of a geometric weighting function involving orientation n, linear depth d, and an image space filter k. For simple spatial upsampling, k can be chosen arbitrarily, e.g., a linear hat function [YSL08]. The σ terms are user-defined variables. Whereas we kept $\sigma_n = 0.2$, σ_z depends on the scene. For our results we used 3% of the difference between the near and far frustum plane.

For higher efficiency, we choose N to cover only the four nearest pixels in the low-resolution image l. One can choose n and d freely as long as it favors similarity, but falls off quickly. The method in [YSL08] uses Gaussian filters, which can be relatively expensive. We use a simpler yet similar function for both n and d:

$$g(x,\sigma,\gamma) = (\max(\varepsilon, 1-(x/\sigma)))^{\gamma}. \tag{8.3}$$

σ represents the filter width, and γ controls the fall-off. We choose $\gamma = 3$, which corresponds to the *tri-weight* kernel $g(x^2, \sigma^2, 3)$. A plot of the function for various σ is shown in Fig. 8.1(left). It has finite support and is clamped at a small epsilon which avoids zero weights.

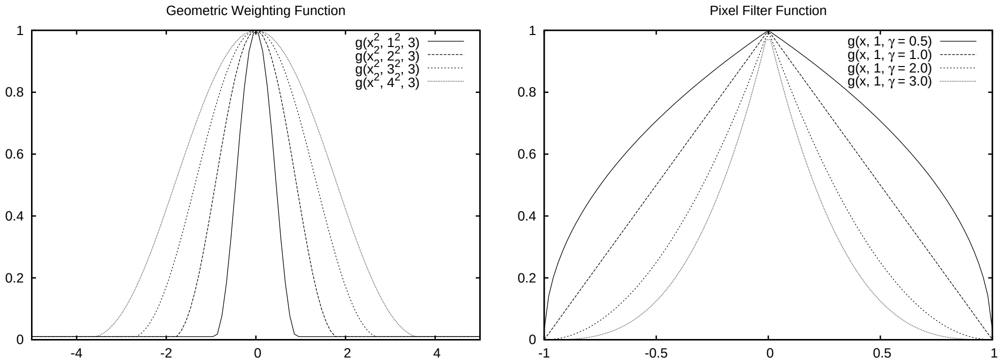


Fig. 8.1 – Joint-bilateral-upsampling filter functions

Left: geometric weighting function n, d (triweight kernel), right: image space filter function k.

2.2 Temporally-Amortized Spatial Upsampling

To increase the final image quality, one can also look back in time. Instead of interpolating within the current frame, missing pixels are resurrected from past frames. This is most beneficial if frames correspond to differing pixel subsets. Otherwise, for the static case, no new information is gained over time and accurate convergence becomes impossible.

A random pattern was proposed in [NSL*07], but it costs some efficiency because it implies a supplementary rendering pass to fill in the missing information, slightly accelerated via early-z termination. For our method, we want to avoid the computation of extra samples in order to ensure a relatively constant cost per frame, just like [YSL08]. To make updates efficient on the GPU, we use a spatially regular pattern as shown in Fig. 8.2. To render a sample set, we apply translations in post perspective space which can be encoded in the projection matrix and come at no supplementary rendering cost. The regularity will also be helpful for the spatio-temporal filtering (Section 2.3). We will concentrate on a 4×4 window for the rest of this section, but discuss other sizes in Section 3. The process is illustrated in Fig. 8.2.

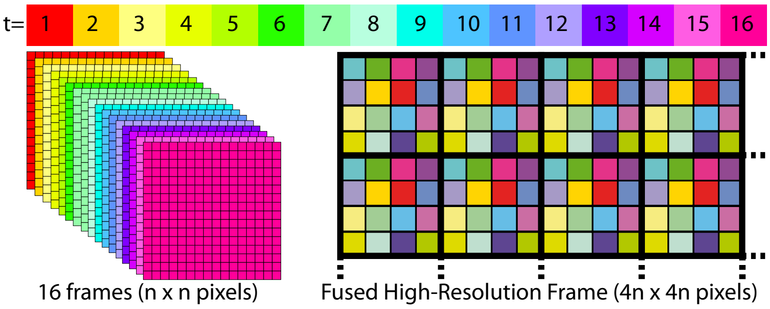


Fig. 8.2 – Interleaved spatio-temporal sampling pattern

Left: A regularly sampled low-resolution shaded image is produced for each frame. The high-resolution output is then fused from the previous frames.

If pixels were simply reused from previous frames, visible distortions are likely to appear. Any camera movement or scene changes result in ghosting trails following the objects. One can compensate for this effect by computing pixel displacements between frames, referred to as motion flow. Motion flow is inexpensive because all the necessary 3D information is available [SaLY*08a]. To improve quality, and because it is cheap, we compute the 2D motion flow in a high-resolution texture, always considering only two successive frames. For brevity we will refer to the q-frames motion-compensated pixel i at time t as $i(t-q)$ (i.e., $i(t) = i$).

In standard temporally-amortized upsampling one can keep the payload of the updated pixel set in the final output and let the other pixels search in the previous frame:

$$h(i,t) = (1 - w_t(t,i))\ h((i(t-1),t-1) + w_t(t,i)\ l(\hat{i},t), \tag{8.4}$$

where $h(i,t)$ is the high resolution image payload for the current pixel index i at time t, which is composed of the currently computed low-resolution image payload $l(\hat{i},t)$, $\hat{i}$ is the nearest low-res. pixel index of i, and the previous image payload $h(i(t-1),t-1)$ at the motion compensated position $i(t-1)$. Special care has to be taken if $i(t-1)$ is pointing to a pixel outside the screen or if the 3D point corresponding to the pixel $i(t-1)$ is disoccluded (i.e. has not been visible in the previous frame). Those pixels cannot be fetched from the previous frame $h(\cdot,t-1)$ and, for temporally-amortized upsampling, are usually recomputed. To identify disocclusions we compare our warped depth values with the depth values from previous frame [NSL*07]. Comparing only depth values we may miss certain disoccluded pixels located at contact points (see Fig. 8.3). To clear the ambiguity we also compare material IDs, which we store anyway in the geometry pass of our deferred renderer. For temporally-amortized upsampling, pixels are taken from the previous payload if they have not been computed in the current frame. The binary weight w_t determines for a given pixel whether to fetch the payload from the current or the previous frame(s), by setting $w_t(t,i) = 1$ if the pixel i has been computed in the current frame at time t, and zero respectively.

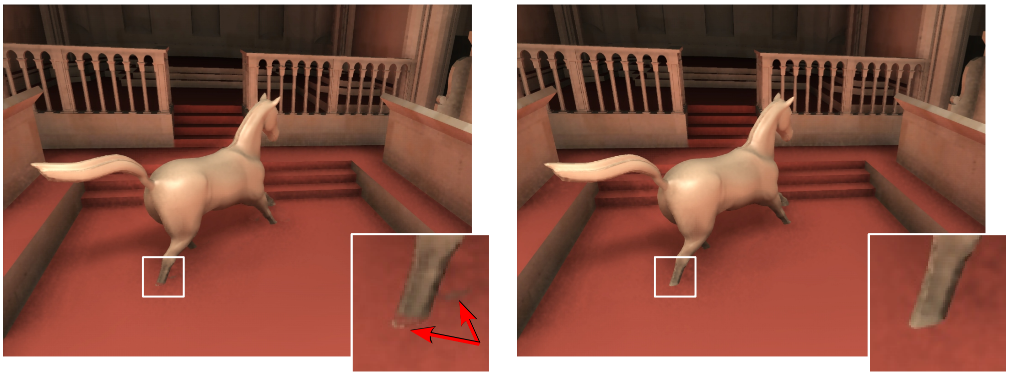


Fig. 8.3 – Conservative detection of disoccluded pixels

Dealing with disocclusions in temporal reprojection: reprojection based on z-Buffer comparison may fail at contact points (left). Additional criteria (here material IDs) help to reduce ambiguities (right).

2.3 Spatio-Temporal Upsampling

The previously described upsampling schemes are not always well suited. Constructing the image only spatially is very efficient, but prone to undersampling and blurring of sharp image features. On the other hand, temporal caching [SaLY*08a, NSL*07] is sensitive to temporal changes, but it converges if the scene is nearly static. Consequently, we would like to combine the two approaches in a spatio-temporal upsampling framework:

$$\begin{aligned} h(i,t) \quad = \quad & \frac{1}{\sum w_s w_t w_f} \sum_{q=0}^{T} \sum_{j \in N\{i(t-q)\}} \\ & w_s(i(t-q),j)\; w_t(t-q,j)\; w_f(q)\;\; l(\hat{j},t-q), \end{aligned} \tag{8.5}$$

where $w_f(q)$ is a temporal fadeout kernel to favor payloads that have been computed recently ($w_f \in [0..1]$). A good choice is an exponential falloff, e.g., $w_f(q) := 0.9^q$, but we improve upon this in Section 2.5. The other terms were explained in Equations 8.1 and 8.4. As indicated before, T is usually 16 (i.e. 4×4 upsampling), which is the amount of low-res textures needed to cover all samples of the high-resolution frame. Intuitively, we follow the sample i back over time.

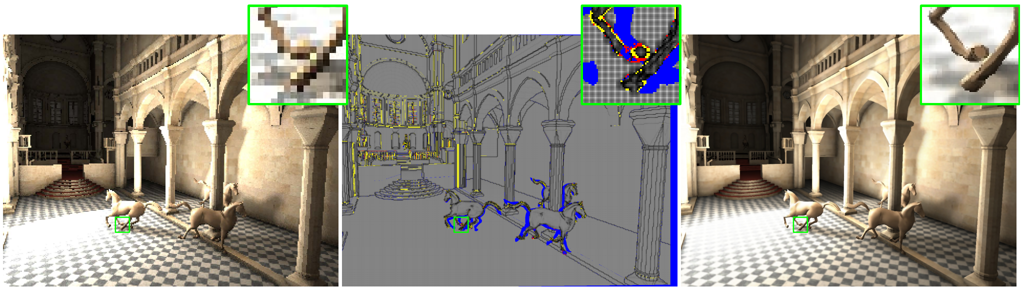


Fig. 8.4 – Visualization of spatial upsampling pipeline

Left: dynamic shading result, middle: sum of upsampling weights w_s (brightness represents confidence, yellow regions have insufficient spatial weight, blue regions are disoccluded and have no temporal weight, and red pixel are both), right: upsampled image

For each time step t, w_t ensures that only those payloads are considered that have been computed at time t. The contribution is then influenced by the weight w_s that measures the geometric difference and w_f, that penalizes age.

The pixel filter k, computed in w_s, has to be chosen carefully. Precisely, we define $k(i,j) := g(||x(i) - x(j)||, r, \gamma)$, where $x(i)$ is the pixel position on the screen, and r the low-res pixel diagonal length. For $\gamma = 0$ filter k is constant and the results approach spatial upsampling. For $\gamma = \infty$, k is a dirac-like filter ($k(i,j) := 1$ if $(i = j)$, 0 otherwise) and, hence, is equivalent to temporally-amortized upsampling. Consequently, we want to use the function k to blend between the two extrema depending on the temporal coherence, see Fig. 8.1(right). A better convergence over time, is achieved with a larger γ, while a lower value improves the temporal response. We will present a temporal gradient-guided steering of γ in Section 2.5 and one can assume for the moment that $\gamma = 3$.

Fig. 8.4(middle) visualizes the spatial upsampling weights w_s of 4×4 pixel regions. Pixels marked as blue are disoccluded. Consequently, these cannot be taken from previous frames. Yellow pixels indicate a small geometric weight for the current frame and are, thus, likely to be undersampled. Here, previous frames should be used. Pixels in red have neither spatial nor temporal confidence, making them candidates for recomputation. In general, such pixels are sparse and our filtering mechanism ensures that a plausible value is attributed, making it possible to avoid recomputation.

2.4 Exponential History Buffer

Our filtering in Eq. 8.5 leads to better results than spatial or temporal filtering alone, but it's beyond question that it is computationally expensive because we need to perform many dependent texture lookups repeatedly to produce one pixel. To improve performance, we observe that the lookups are coherent over time. It is thus possible to cache these lookup chains through previous frames. This leads to the idea of an exponential history buffer. It stores the previous upsampling result h_r, and the previous spatio-temporal upsampling weight h_w. Ignoring the temporal component in Eq. 8.5, we obtain a standard bilateral upsampling:

$$\tilde{h}(i,t) = \frac{1}{\sum w_s w_t} \sum_{j \in N\{i\}} w_s(i,j) w_t(t,j) l(\hat{j},t) \tag{8.6}$$

with weight $\tilde{w}(i) = \sum w_s w_t$. Given $\tilde{h}$, and our history buffers h_r and h_w, we compute the actual output h via weight-dependent blending:

$$h(i,t) = \frac{\tilde{h}(i)\ \tilde{w}(i) + w_f(1)\ h_w(i(t-1))\ h_r(i(t-1))}{\tilde{w}(i) + w_f(1)\ h_w(i(t-1))}. \tag{8.7}$$

The history buffer is updated for the next frame by storing $h_r(i) := h(i,t)$ and $h_w(i) := \tilde{w}(i) + w_f(1)\ h_w(i(t-1))$. If one has chosen w_f as an exponential falloff, Equations 8.5 and 8.7 are equivalent. Note that $w_f(0) = 1$. This reduces the performance penalty with respect to [YSL08] to a single texture lookup and a few algebraic operations. Furthermore, other simplifications are possible: we no longer need 16 low resolution textures (one high-resolution history buffer and one low resolution frame are enough), time dependence vanishes in general, and in particular, for the function w_f.

2.5 Temporal Weighting Function

In the previous section, we have seen an efficient way to involve a longer temporal history. Nevertheless, in some cases this might not be wanted. Fast changes, induced by, e.g., shadows, would profit from the use of spatial upsampling instead of temporal. On the other hand, to improve the quality of the shaded output, it is necessary to integrate samples from previous frames. To this extent, we will dynamically adapt the temporal weight w_f and the image space filter k locally.

Precisely, we want to address the following points:

1. Temporal flickering due to low resolution sampling of high-frequency spatial signals;
2. Temporal ghosting artifacts due to fast temporal changes;
3. Higher-quality convergence for static elements.

To choose w_f and falloff of k appropriately, we examine the variance of the temporal signal. We do this by relying on a temporal gradient. A weak gradient implies that more confidence can be granted to the evaluation over time. Consequently, w_f should be large and k should reduce image blur by giving more weight to actually computed pixels in the history buffer. Contrarily, large gradients indicate that changes occurred and that the history is no longer reliable. Accordingly, w_f should fall off faster and k should favor spatial upsampling by introducing more image blur. We achieve this by scaling the exponential fall-off γ of our filter function k defined in Section 2.3 with our confidence value w_f, where a maximum of $\gamma \cdot w_f = 4$ is a good choice and was used in all our experiments.

2.5.1 Estimating Temporal Gradients

We will assume that the payload h is one dimensional. Otherwise, one could either compute a norm of the payload, or consider separate gradients. We rely on finite differences to define a temporal gradient because deriving analytical gradients is not always feasible and besides would interfere with the shading pipeline. Further, we are not actually computing a temporal gradient as we do not divide by the time difference between frames (i.e., assume $\Delta t = 1$). This is because we want the history adaption to be sensitive to the frame rate such that the temporal weight w_f increases with the frame rate even if the "real" temporal gradient is constant. Moreover, to address the case where the value domain is unknown, we realized that it usually makes sense to compute relative gradients. In particular, for colors, such a definition tends to capture contrast (when reducing color to luminance).

$$\frac{\partial}{\partial t}h(i,t) = \frac{h(i,t) - h(i(t-1),t-1)}{max(|h(i,t)|,|h(i(t-1),t-1)|)}$$

A more general definition according to time is not needed because the previous section reduced the time dependence to a single frame. We then define:

$$w_f := a \cdot \left(1 - |\frac{\partial}{\partial t}h|\right)^2, \tag{8.8}$$

where a steers the sensitivity between spatial and temporally-amortized upsampling. Usually a is kept to 1 in our experiments. The square in Eq. 8.8 suppresses w_f more aggressively for large temporal gradients than for small fluctuations. One may notice that w_f depends on $h(i,t)$, which is the result we currently want to compute. Therefore, we approximate h based on $\tilde{h}$, the spatially upsampled result we computed to make use of the history buffer.

Because our spatial upsampling $h(i,t)$ may suffer from temporal flickering and finite differences are potentially always noisy, we decided to use a low-pass filtering in time *and* space. Simple spatial filtering, e.g., using mipmapping, would not be an option since it cannot detect spatial aliasing artifacts (which might result in low-frequency). However, due to our imposed "jittering", spatial aliasing corresponds to a periodic high-frequency temporal signal, which is greatly reduced when summing a few consecutive frames. Therefore, we address this issue by making use of the current and the three previous finite differences. These are stored in a vector $\vec{\nabla} := \left(\frac{\partial}{\partial t}h(i(t),t),\ldots,\frac{\partial}{\partial t}h(i(t-3),t-3)\right)$. To filter the gradient temporally, we compute an absolute weighted sum $||\vec{\nabla}|| := |\vec{\nabla} \bullet (0.4,0.3,0.2,0.1)^T|$. The sum, as well as the current and the two previous differences can be stored in a single 4-channel RGBA texture.

While the finite differences will only be relevant for the next frame, $||\vec{\nabla}||$ is used to control the temporal-spatial upsampling. Though filtered over time, this value can still fluctuate spatially. To additionally filter it in space, we want to use a kernel that is slightly larger than the distance between recomputed samples. Thus slightly larger than 4×4. In practice, we found that tri-linearly filtered mipmaps deliver a good quality/ performance trade-off, when values are chosen from level 2.5. However, such filtering can also reveal small artifacts due to discontinuities in the mipmap. This leads to an improved estimate, in particular, in the presence of aliasing as can be seen in (Fig. 8.6). This temporally and spatially filtered value $||\vec{\nabla}||$, is then used in Eq. 8.8.

Care has to be taken in the special case where pixel regions are disoccluded and cannot be reprojected to the previous frame. Here, computing a temporal gradient is impossible. Simply ignoring the gradients for such pixels may lead to visible discontinuities at the transition boundaries between successfully reprojected pixels and disoccluded pixels. In order to suppress these discontinuities, we set the relative temporal gradient to its maximum for all disoccluded pixels. In consequence, the temporal gradient spreads into the neighboring pixels after the spatial low-pass filtering. This leads to a gradually diminishing gradient and, hence, a smooth transition between spatial and temporally-amortized upsampling (see Fig. 8.5).

It has to be pointed out, that whenever the gradient has been falsely assumed to vary, our method does not *break* and instead falls back to spatial filtering. The adaptation of w_f only improves the quality of the results. Even though, our algorithm is not ensured to converge to the actual high-resolution solution, the increase of the temporal window improves quality significantly as illustrated in Fig. 8.6 (left) and the accompanying video.

Alternatively, temporal gradients could be obtained via joint-bilateral spatial upsampling of the known temporal gradients based on the newly computed values. In practice, this resulted in more expensive and qualitatively less convincing results. Our solution filters space and time, for very little cost and better handles disocclusions. We investigated special cases, like zero motion vectors, but found that quality remained similar, making the supplementary memory load for storing these motion vectors unjustified.

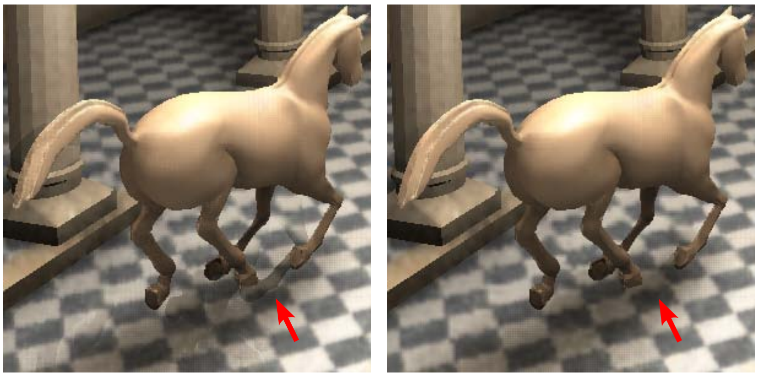


Fig. 8.5 – Dealing with disoccluded pixels in temporal gradient estimation

Disoccluded pixels in this cropped screen-shot of a running horse can cause discontinuities along motion boundaries (left), which are smoothed when setting the maximum relative gradient for all disoccluded pixels (right) since it is spread to neighbor pixels during the spatial low-pass filtering.

3 Implementation Details and Overview

So far, we considered upsampling windows of size 4×4. This is a strong approximation and up to 16 frames are needed to produce an accurate result in a static scene, although high-quality convergence is often faster. For fluctuating payloads it can make sense to decrease the window size. The algorithm easily extends to various upsampling sizes, such as 2×2 or 6×6. Plots showing the mean frame-rate and error dependency on the upsampling-window size are shown in Fig. 8.7.

The "optimal" upsampling-window size depends on many factors like frame-rate, screen resolution, scene complexity, shading frequency, error perception etc., such that choosing the best window size automatically becomes a non-trivial task. Besides the optimal choice might be also very subjective. We also found that for larger screen resolutions than 1280×1024 and frame-rates beyond 60 Hz, 2×2 windows of varying values are better addressed with simple temporally-amortized upsampling (not even spatial bilateral upsampling is needed) and windows larger than 6×6 take too much time to converge to a high quality result. Nevertheless, for even higher resolutions, this might change. Further, complex shaders often consist of several independent shading terms, some of which can be of very low frequency and high computational complexity. In fact, most high-frequency components (such as textures, direct light shadows) are not expensive and could be efficiently computed for every pixel. Hence, following [SaLY*08b], splitting the shader into individual components enables even higher gains. To illustrate the effectiveness of our approach, we refrained from such decompositions in our results.

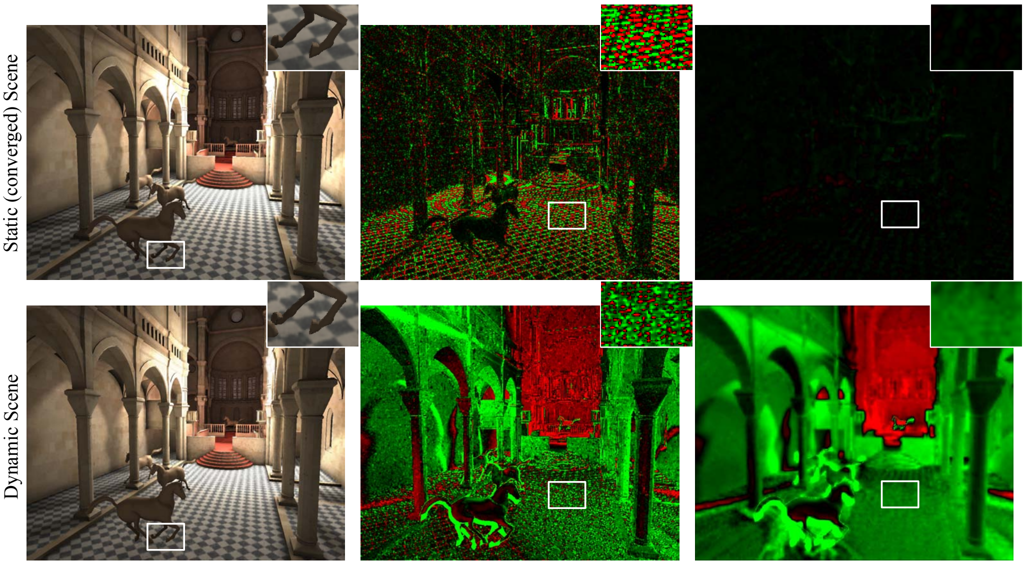


Fig. 8.6 – Visualization of temporal gradients estimation for different configurations

Temporal gradients are important to detect changes in illumination. The figure shows gradients in a static (frozen) scene (top row) and the same frame in a fully dynamic scene (bottom row) (red = negative, green = positive gradients). Simple finite differences (center) are prone to aliasing and noise. Our filtering (right) regularizes the gradient estimation (e.g., for the static scene the gradient is almost zero (top right)). It eliminates flickering and achieves high quality. Some high-frequency gradients (bottom right) might be lost, but this is almost invisible in a dynamic context. Our upsampled result is shown on the left. (For demonstration purposes, we show signed gradients. Spatial smoothing relies on absolute values $||\vec{\nabla}||$*.)*

Our spatio-temporal upsampling is integrated into a deferred shading approach. In the initial pass all necessary geometry and material information as well as the 2D motion vectors are written to the high-resolution G-buffer. In the next pass all expensive shading computations are performed at low resolution. The frustum is jittered in a coherent sampling pattern (see Fig. 8.2) corresponding to a different subset of pixels. Next, we upsample the current shading result to high resolution taking into account the high-resolution G-buffer and motion flow and solely the previous upsampling result (thanks to Section 2.4) and the (mipmapped) temporal gradients $\vec{\nabla}$. Because the temporal gradient after filtering is assumed to be spatially coherent, we do not need to compute the upsampled gradients at a high resolution, but instead compute them at an intermediate resolution (e.g., every 2×2 pixels).

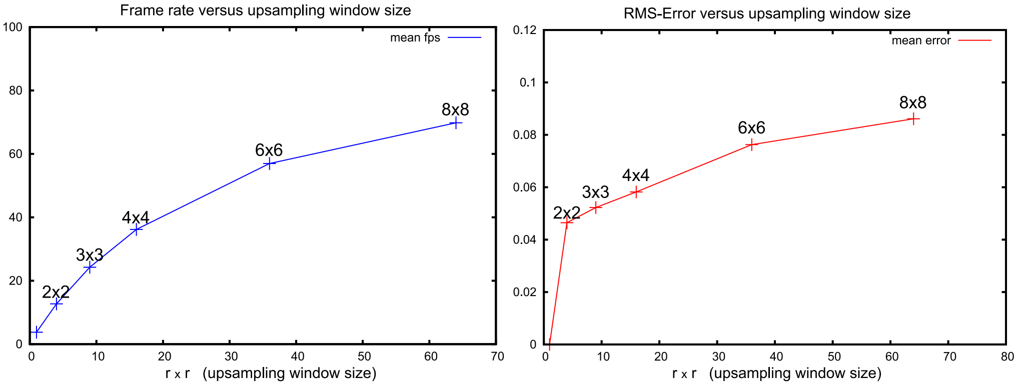


Fig. 8.7 – Frame rate versus frame error plot

The left plot shows the frame rate dependency on the upsampling window size in pixels, whereas the right plot shows the normalized root-mean square error for various upsampling window sizes ranging from 2×2 *to* 8×8 *pixels. The shown measurements are the average over all frames of the animation sequence that is shown in Fig. 8.3, where the upsampled image resolution was set to* 1280×1024*.*

3.1 Antialiasing

Our spatio-temporal upsampling is extendable to antialiasing in the spirit of [YNS*09]. A common way to achieve antialiasing with deferred shading is to compute the results in a $m \times m$ higher resolution and then merge $m \times m$ pixels to obtain the average pixel color, which corresponds only to regular pixel supersampling. Nevertheless, such scheme nicely fits to our proposed upsampling. We only need to compute the geometry and material buffer and the intermediate upsampling buffers at a higher resolution than the display buffer and down-sample the upsampled result to the display resolution as shown in Fig. 8.8. To some extent this follows the hardware-based antialiasing which computes shading values at a lower resolution. Nevertheless, the current state-of-the-art hardware antialiasing cannot involve previous frames, nor is it compatible with deferred shading, which results in a significant overhead and its quality depends on the shader's spatial frequency. Although the shading pass is computed at the same resolution, the upsampling performance drops down for larger upsampling windows (see Fig. 8.7) and besides, the initial geometry pass also needs to output all geometry and material data at sub-pixel precision, which penalizes memory bandwidth. Clearly, we would like to reduce the upsampling computation at sub-pixel level since upsampled sub-pixels are not displayed directly but instead are downsampled before being displayed anyway. Hence, we can approximate the upsampled color of individual sub-pixels as long as their statistical mean value is correct. However, although this seems to be an interesting track, it has not yet been investigated and is considered as future work.

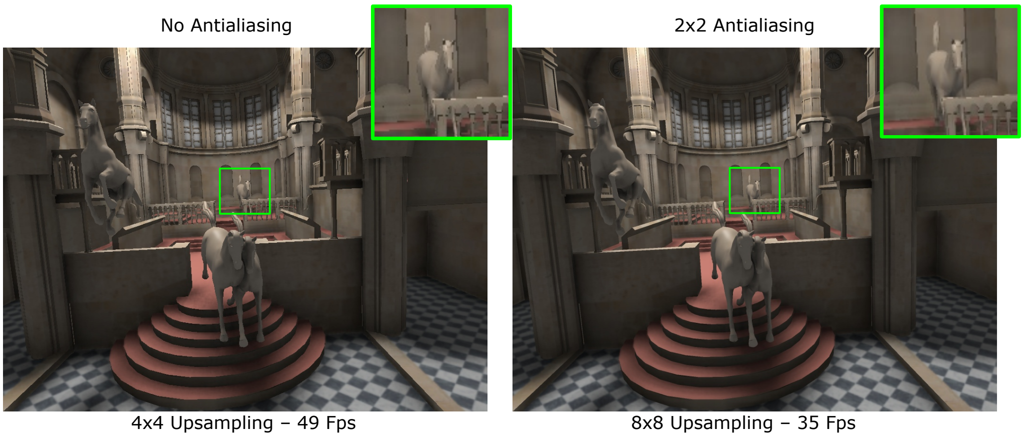


Fig. 8.8 – Spatio-temporal upsampling and antialiasing

(left) our 4×4 *spatio-temporal upsampling, (right) antialiasing via regular sub-sampling can be achieved with our method by first upsampling to a higher resolution (e.g.,* 8×8 *upsampling) and then downsampling to smaller resolution.*

4 Results

For comparison, we implemented a simple temporally-amortized upsampling as described in Section 2.2 with second-pass hole filling [SaLY*08a] and spatial upsampling [YSL08]. We used OpenGL with GLSL on a GeForce GT 280 and performed tests on various challenging scenes at a resolution of 1280×1024. The timings are giving in Table 8.1 and the visual results with statistical error measures (PSNR) are shown in Figure 8.9. The initial geometry pass of our deferred renderer deviates only slightly throughout the different methods and is not explicitly shown in Table 8.1. The overhead for computing the motion vectors was always less than 1 ms in our experiments.

The first scene is a jeep with little geometry, but a moving light source, challenging texture, percentage-closer filtering (PCF) [RSC87] with a 6×6 kernel, Fig. 8.9(top). The second scene, contains animated horses, Fig. 8.9(third row), with a static camera and applies screen-space ambient occlusion (SSAO) [BS08] with 8×8 randomized horizon samples. In a different scenario, Fig. 8.9(fourth row), we added instant global illumination by evaluating a large number (2000) of virtual point lights (VPLs) without visibility generated with reflective shadow maps [DS05]. In this setup the camera is moving and the indirect lighting is changing quickly. Our approach even suppresses the flickering due to temporal noise of the VPL sampling. Please refer also to the video material provided at [Web10]. Finally, the last scene shows a running elephant with a large animated body which is rendered offline at fixed frame rate (30 fps) with SSAO (32×24 samples), a spot-light with PCF, and screen-space directional occlusion

		temporal		spatial		spatio-temp.		reference
	T_{shade}	T_{up}	fps	T_{up}	fps	T_{up}	fps	fps
Jeep (PCF)	1.6	1.0	244	1.1	239	2.1	191	171
stdev.	0.02		2.4		1.6		1.5	
Horses (AO)	11.3	5.3	47.5	1.1	62.5	2.2	55.9	7.6
stdev.	0.09		0.69		0.36		0.26	
Horses (AO+GI)	35.7	45.2	11.4	1.6	24.5	3.0	21.8	2.2
stdev.	0.27		3.2		0.56		0.78	
Elephant (AO+DO)	80.3	76.1	5.9	1.8	12.3	2.9	11.9	0.7
stdev.	39.7		4.5		5.1		4.9	

Table 8.1 – Timings of different GPU upsampling methods

Final average fps and standard deviation (stdev.) for rendering a frame with spatial, temporally-amortized with disocclusion-hole filling, spatio-temporal upsampling and reference for image resolution of 1280×1024 *and a* 4×4 *upsampling window. The time for rendering the low-res shading input,* T_{shade}*, and the time spent on upsampling only,* T_{up}*, is given in milliseconds.*

(SSDO) [RGS09] with direct light sampling (128 samples) from an environment map. This scene is challenging as the viewpoint changes very quickly and reveals large surfaces making it difficult to reproject old samples. Furthermore, the shading is of high-frequency and cannot be accurately reproduced by neither method. To favor spatial upsampling, we set $a = 0.7$ for this scene. Temporally-amortized upsampling completely fails in this scenario whereas our method, similar to spatial upsampling, still produces decent results.

In Tables 8.2 and 8.3 we compare the errors and corresponding frame rates of the different upsampling methods for an entire animation in a mostly static and almost converged scene and in a fully dynamic scene with high-frequency shading, respectively. The frame error is computed for each frame separately using a simple error metric (normalized root mean square error) with respect to the reference frames. Although not always outperforming spatial upsampling, our method still produces visually more pleasing results, which cannot be captured by an image quality metric disregarding the animation context. The reason for this is that the frames produced by our methods are also low-pass filtered in time resulting in a slight delay in the dynamic shading of pixels.

Results generally benefit from temporally-amortized *and* spatial upsampling. The horse scene shows that in static regions the SSAO converges nicely whereas in dynamic regions, near the running horses, more values are taken from the current spatial upsampling, trading-off response for quality. The jeep's texture (even when not separating it from the payload) faithfully converges and the shadow shows only minor artifacts that are hidden by its motion. Temporally-amortized upsampling shows many ghosting artifacts and spatial upsampling does not converge to a high-quality solution as shown in Table 8.2(left). This is visible near geometric details, e.g., in the background of the cathedral where geometric discontinuities become smaller and aliasing artifacts emerge. Moreover, flickering appears when the camera moves because of high-frequency details that were undersampled by the low-resolution payload. Temporally-amortized upsampling cannot assure a constant frame-rate because the supplementary rendering to fill disocclusion holes can have very differing cost (see Table 8.3(right)) even requiring to compute every pixel in the worst case. Our approach is similar in performance to spatial upsampling, as illustrated in Table 8.1 and Table 8.3 while maintaining a good image quality overall. This makes our solution a good choice for dynamic and static scenes.

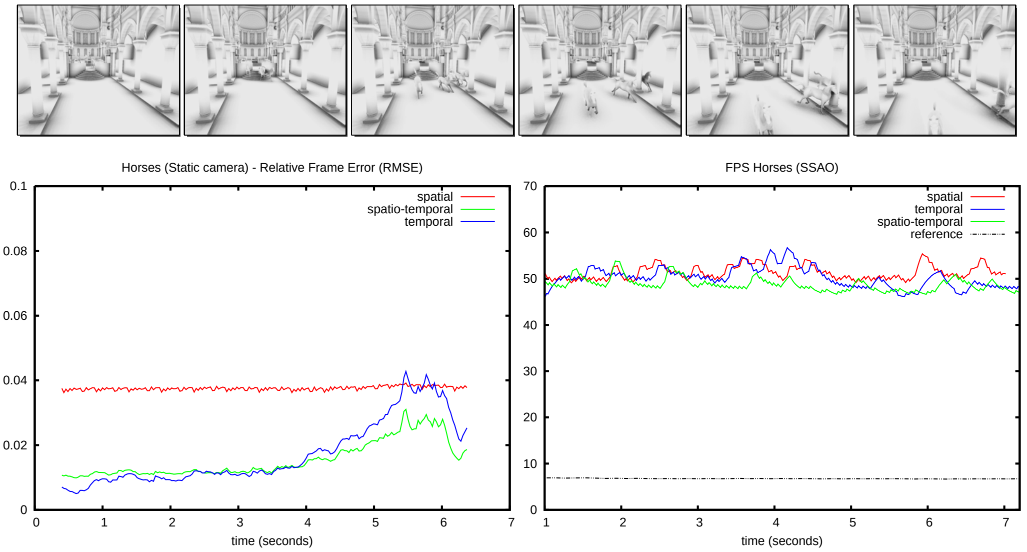


Table 8.2 – Errors and frame-rate for an animation sequence with mostly static shading

(top) a few frames extracted from the animation sequence rendered using screen-space ambient occlusion (SSAO) for a relatively static (converged) scene, (left) frame errors (normalized root mean square error) for spatial, our spatio-temporal, and temporally-amortized (temporal) upsampling. (right) corresponding frame rate in frames per second (FPS) for the three upsampling schemes and the reference.

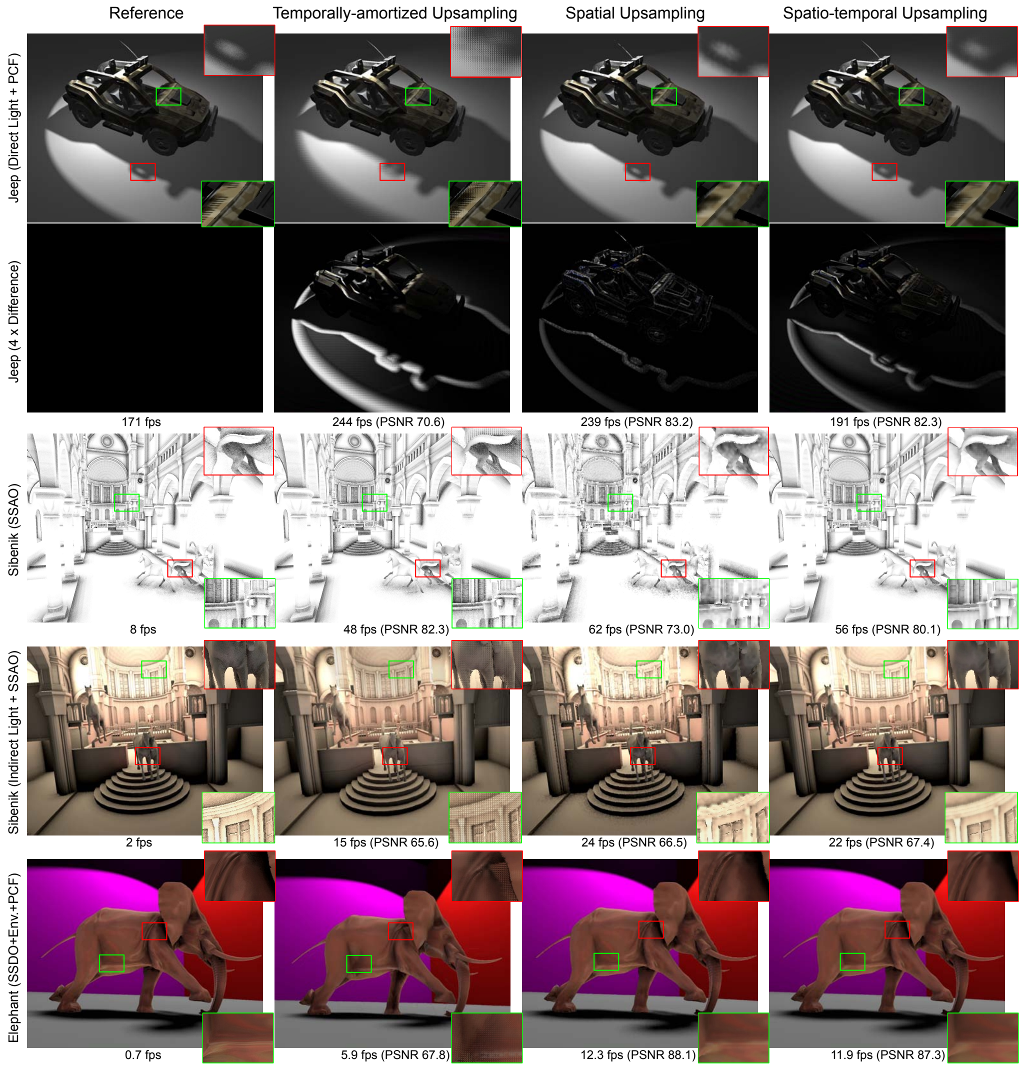


Fig. 8.9 – Comparison of spatial, temporally-amortized, and spatio-temporal upsampling.

The first two rows show screenshots and difference images (4 times scaled) of simple shading (static geometry and dynamic light). This is a difficult case for our method, but it performs better than amortized temporal upsampling. Although the PSNR is smaller than for spatial upsampling, we obtain a smoother difference image with less aliasing. Geometric discontinuities are better handled leading to a more visually pleasing result. The third row shows results of a more expensive shader (SSAO) (dynamic scene and static camera). The background exhibits the high quality achieved by temporal convergence. The PSNR reflects the positive influence of our method. The fourth row shows screenshots from a fully dynamic scene including camera motion. Our method also filters high-frequency flickering artifacts due to random VPL sampling. The bottom row shows another dynamic scene with high-frequency shading (screen space ambient (SSAO) and directional occlusion (SSDO) with environment map lighting and PCF shadows).

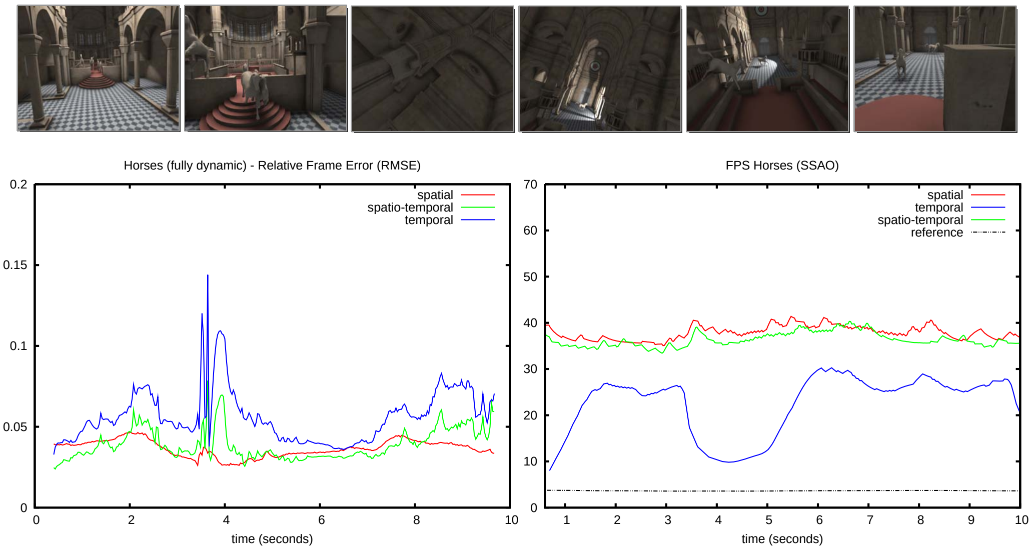


Table 8.3 – Errors and frame-rate for an animation sequence with dynamic shading

(top) a few frames extracted from the animation sequence rendered with dynamic direct lighting and screen-space ambient occlusion (SSAO), (left) frame errors (normalized root mean square error) for spatial, our spatio-temporal, and temporally-amortized (temporal) upsampling. (right) corresponding frame rate in frames per second (FPS) for the three upsampling schemes and the reference.

5 Discussion and Future Work

Although our algorithm produces generally better and more robust results than previous real-time upsampling techniques [YSL08] for the GPU there are situations where it may also fail. First limitation is that our reprojection caching is solely based on fast computable "geometric" flow which might differ from *optical flow*. As a worst case scenario imagine a fast moving object with a camera attached to it moving at the same speed. A shadow cast by the object on a static floor is completely decorrelated with the world-space pixel positions and temporal reprojection can not help. In such case mainly geometry-aware spatial upsampling is influencing the final pixel color as temporal gradients become large.

Further, our algorithm relies on the assumption that temporal changes in the shading are smooth and spatially coherent and can therefore be low-pass filtered. And all quickly varying signals are only due to aliasing or noise in the shading[1]. Otherwise, we could not reliably estimate our temporal weighting coefficient w_f, trading spatial with temporally-amortized upsampling, as our gradients would

[1] Ideally, to avoid aliasing we would like to low-pass filter the input signal to the shading instead of the shading result itself. However, this way we would have to interfere with the shader algorithm, which we want to treat as a black-box decoupled from the upsampling

flicker. Hence, fine details in the shading that are also quickly changing in time are hard to detect. Fortunately, fast temporal changes are also hard to track by the human visual system and may appear blurry.

A second minor shortcoming is the influence of apparent motion blur in the temporally-amortized upsampling, which arises from the discretization in the bilinearly-filtered reprojection accumulated over the history of reprojected frames as illustrated in Fig. 8.10. Even though our motion vectors are computed at high resolution, blur still appears when for example moving the camera because small deviations from the pixel center in the reprojection accumulate over several frames (see Fig. 8.11). A simple remedy also proposed in [YNS*09] is to increase the resolution of our exponential history buffer to sub-pixel accuracy, which we do for antialiasing purposes (see Section 3.1). A more sophisticated approach would replace the hardware-accelerated bilinear interpolation with higher-order (e.g., cubic) interpolation. On the other hand, our spatio-temporal upsampling reduces the temporal influence of old pixels and shortens the lookup-chain leveraging the spatial coherence (even when shading is static pixels always have a small influence on the spatial neighborhood).

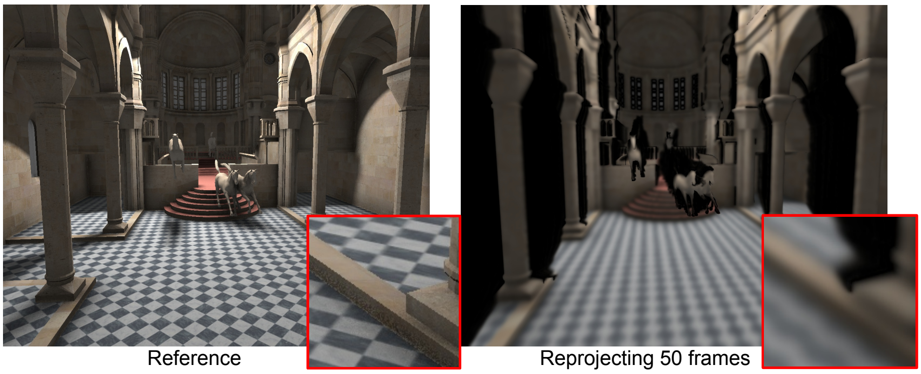


Fig. 8.10 – Reprojection blur

When only reprojecting frames, more and more blur is introduced due to the bilinearly filtered history (right image).

Our work relies on temporal gradients solely to adapt the temporal filter weights. Ideally, we would have wanted to extrapolate information based on these gradients, but this did not proof to be robust enough due to accumulated errors in those gradients.

Our algorithm produces better results with increasing frame-rates because incorrect pixels are quickly refreshed by newly computed ones. A downside are the temporal patterns, that at least in screen shots, are clearly recognizable as the deterministic temporal sampling pattern. However, more sophisticated adaptive methods would easily annihilate our gained speedup. A dynamic change in sampling patterns could help to hide the problems behind noise, but we believe that more complex adaptations of the spatial-temporal weighting might be a better

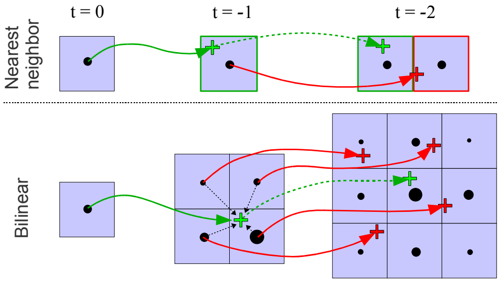


Fig. 8.11 – Precision loss in temporal reprojection

(top) nearest neighbor reprojection results in discontinuities when the correct continuous motion flow (green arrows) and the approximate nearest-neighbor pixel-flow (red arrows) point to different pixels. (Bottom) bilinear texture filtering trades discontinuities with blur since an ever-growing neighborhood influences the final result.

choice that we keep as future work.

At last, an interesting track for future work, is how to apply our upsampling method for real-time streaming of rendered images, which has recently become more and more attractive since all data needed for rendering is always up-to-date and kept confidential on the server-side, while the client solely receives a video stream of rendered frames and therefore needs no powerful hardware. Our proposed upsampling framework nicely fits to this concept of streaming video data. A simple, yet effective approach to imagine, would be to stream the MPEG-encoded motion flow, which is highly coherent and compressible, as well as the encoded low-res shading image with additional geometric edge information to the client side. The client then reconstructs the high-res image using the upsampling method proposed in this chapter.

Conclusion

This thesis concludes with this chapter. Here, we summarize the major contributions and applications of this work and discuss ongoing and possible future work in addition to what has already been pointed out in the dedicated chapters.

Photon Ray Splatting

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 by means of a volumetric search along entire 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 [HBHS05]. Second, our method does not suffer from discontinuities in surface orientation. Since we estimate the density over photon rays, 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 geometry 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 [Jen01] since the costly k-NN search can be avoided. Fourth, we have shown how to adapt the classical irradiance caching algorithm [WRC88] for 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 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 method often yields satisfying results, it 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. However, we proposed a remedy for approximating the visibility using discretized volumetric occlusion tests in Chapter 5. And second, only low-frequency lighting can be reconstructed with our method, but final gathering can still be performed similar to photon mapping [HHS05].

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 [Chr99].

Radiance Cache Splatting

In Chapter 6 we presented a high-quality global illumination algorithm with its strength in robustness and automatic adaptation to scene and lighting complexity with relatively little user intervention – setting the standard parameters works well for most scenes – in contrast to traditional photon mapping with (ir)radiance caching. The method extends the original lightcuts algorithm [WFA*05] to be used for (ir)radiance caching and improves the adaptive cache interpolation [KBPv06] by anisotropic cache splatting driven by perceptual visibility thresholds, which could also replace the traditional (ir)radiance caching for photon mapping. We further proposed several optimizations and implementation details to make the algorithm efficient and practical for various scene and light settings. We achieve computation times for a single image in the order of seconds rather than minutes on a standard PC.

Render2MPEG

In Chapter 7 we investigated the problem of simultaneous control of accuracy for rendered and compressed video frames. We demonstrated that by taking into account MPEG's quantization mechanisms and basic HVS characteristics in deriving the perceptual error thresholds, we could significantly improve the rendering performance by relaxing rendering errors, while obtaining video streams with frames surprisingly similar to the compressed high-quality reference frames. By exploiting temporal coherence in rendered frames we could acquire information required by our HVS model and successfully predict the tolerable error map for frames to be rendered, which is a notorious problem for many of existing perception-based rendering solutions. Our results clearly show that stronger integration of rendering and compression software is desirable to avoid redundant computation by existing frame-by-frame standalone renderers. In this context, algorithms for temporally coherent global illumination computation become important and our extension of the lightcuts algorithm aims in this direction.

Spatio-Temporal Upsampling on the GPU

Although spatio-temporal processing has been explored in different contexts, it has received less attention in terms of GPU rendering. In fact, GPUs inherently exploit spatial coherence in the SIMD structure of the massively parallel processing pipeline. However, modern GPUs benefit very little from temporal coherence. In Chapter 8, we proposed a relatively simple framework for spatio-temporal rendering as a trade-off between efficiency and quality. Compared to joint-bilateral spatial upsampling our algorithm produces higher quality and introduces only a small performance overhead and keeps memory demands small. For dynamic scenes, our algorithm is more robust than temporal reprojection caching. It combines benefits from both methods. We demonstrated on relatively complex shaders that our upsampling technique can reduce the GPU's shading costs. Our solution is well suited for increasing screen resolutions and beneficial for various algorithms, e.g., global illumination, soft shadows, or procedural textures.

1 Ongoing Work

Aside from the specific directions for future research proposed in the previous chapters, our key idea that draws upon all presented developments in this thesis is the extension of the render2mpeg and spatio-temporal upsampling framework to real-time streaming of rendered frames. Remote rendering with streaming has recently become more and more attractive since all data needed for rendering frames is kept up-to-date and confidential on the server-side, while the client solely receives a video stream. In Chapter 7 we already proposed a method to adapt expensive global illumination computation to the quantization level of lossy MPEG compression steered by the transmission bandwidth of the MPEG-stream. However, in this context we did not profit in terms of MPEG encoding nor in bandwidth reduction. Therefore, we started to investigate how the upsampling framework introduced in Chapter 8 can be used to not only save computation time on the server-side but also how to better utilize the limited bandwidth while at the same time speeding up MPEG encoding. Currently, we achieve this by feeding additional knowledge from rendering like motion flow, geometry into a custom-designed video compressor essentially freeing the MPEG encoder from redundant motion compensation. In contrast, we favor more expensive decoding on the client. This way, we better exploit the computational power of the clients and shift the computational load from the server to the client side.

Bibliography

[AH95] Adelson S. J., Hughes L. F.: Generating Exact Ray-Traced Animation Frames by Reprojection. *IEEE Computer Graphics & Applications 15*, 3 (1995), 43–53. 61

[BDT99] Bala K., Dorsey J., Teller S.: Ray-Traced Interactive Scene Editing Using Ray Segment Trees. In *Proceedings of the 10th Eurographics Workshop on Rendering* (1999). 61

[BFMZ94] Bishop G., Fuchs H., McMillan L., Zagier E. J. S.: Frameless rendering: double buffering considered harmful. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1994), ACM, pp. 175–176. 60, 61

[BG00] Border P., Guillotel P.: Perceptually adapted MPEG video encoding. In *IS&T/SPIE Conf on Human Vision and Electronic Imaging V* (2000), Proceeding of SPIE, volume 3959, pp. 168–175. 162, 165

[BGB08] Brouillat J., Gautron P., Bouatouch K.: Photon-driven irradiance cache. *Computer Graphics Forum (Proc. of Pacific Graphics) 27*, 7 (2008). 64

[BGH05] Burke D., Ghosh A., Heidrich W.: Bidirectional Importance Sampling for Direct Illumination. In *Rendering Techniques 2005* (2005), Eurographics Symposium on Rendering, pp. 147 – 156. 46

[Bli77] Blinn J. F.: Models of light reflection for computer synthesized pictures. *Computer Graphics 11*, 2 (1977). 38, 39

[BM95] Bolin M. R., Meyer G. W.: A frequency based ray tracer. In *Proceedings of SIGGRAPH* (1995), pp. 409–418. 64, 162, 165, 167

[BM98] Bolin M. R., Meyer G. W.: A perceptually based adaptive sampling algorithm. In *Proceedings of SIGGRAPH* (1998), pp. 299–310. 52, 53, 63, 162, 165

[BM05] Bennett E. P., McMillan L.: Video enhancement using per-pixel virtual exposures. *ACM Transactions on Graphics 24*, 3 (2005), 845–852. 63

[Bov05] Bovik A. (Ed.): *Handbook of Image and Video Processing*. Elsvier, Academic Press, 2nd ed., 2005. 162

[BPC*03] Bekaert P., Philipp S., Cools R., Havran V., Seidel H.-P.: *A custom designed density estimation method for light transport*. Research Report MPI-I-2003-4-004, Max-Planck-Institut für Informatik, September 2003. 57, 69, 108

[BS08] Bavoil L., Sainz M.: Image-space horizon-based ambient occlusion. In *SIGGRAPH 2008, Talk Program* (2008). 41, 193

[CB04] Christensen P. H., Batali D.: An Irradiance Atlas for Global Illumination in Complex Production Scenes. In *Rendering Techniques 2004* (2004), Proceedings of Eurographics Symposium on Rendering, pp. 133–141. 24

[Chr99] Christensen P. H.: Faster Photon Map Global Illumination. *Journal of Graphics Tools 4*, 1 (1999), 1–10. 103, 105, 163, 202

[CLRS01] Cormen T. H., Leiserson C. E., Rivest R. L., Stein C.: *Introduction to Algorithms, Second Edition*. MIT Press, 2001. 148

[CM98] Calle D., Montanvert A.: Super-resolution inducing of an image. *Image Processing, International Conference on 3* (1998), 232. 62

[CNL*09] Crassin C., Neyret F., Lefebvre S., Sainz M., Eisemann E.: Beyond triangles : Gigavoxels effects in video games. In *SIGGRAPH 2009 : Technical talk* (2009). 41, 58

[CT81] Cook R. L., Torrance K. E.: A reflectance model for computer graphics. In *Proceedings of SIGGRAPH '81* (1981), ACM, pp. 307–316. 38

[Dah04] Dahmen T.: *Multi Resolution Ray Tracing of Point Data*. Master's thesis, Universität des Saarlandes, Computer Graphics Group, 2004. 79

[Dal93] Daly S.: The Visible Differences Predictor: An algorithm for the assessment of image fidelity. In *Digital Images and Human Vision* (1993), Cambridge, MA: MIT Press, pp. 179–206. 52, 63, 167

[DBB03] Dutré P., Bekaert P., Bala K.: *Advanced Global Illumination*. A K Peters, Natick, MA, 2003. 46

[DCB*04] Dong Z., Chen W., Bao H., Zhang H., Peng Q.: Real-time voxelization for complex polygonal models. *Pacific Graphics* (2004), 43–50. 41, 58, 112, 113

[DDM03] Damez C., Dmitriev K., Myszkowski K.: State of the art in global illumination for interactive applications and high-quality animations. *Computer Graphics Forum 22*, 1 (Mar. 2003), 55–77. 63

[Dis98] Dischler J.-M.: Efficiently Rendering Macro Geometric Surface Structures with Bi-Directional Texture Functions. In *Rendering Techniques '98* (1998), pp. 169–180. 36

[DLW93] Dutré P., Lafortune E., Willems Y.: Monte carlo light tracing with direct computation of pixel intensities. In *Proceedings of Compugraphics '93* (1993), pp. 128–137. 57

[DS05] Dachsbacher C., Stamminger M.: Reflective shadow maps. In *I3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games* (New York, NY, USA, 2005), ACM, pp. 203–231. 58, 193

[DS06] Dachsbacher C., Stamminger M.: Splatting indirect illumination. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2006). 57

[DWWL05] Dayal A., Woolley C., Watson B., Luebke D. P.: Adaptive frameless rendering. In *Rendering Techniques* (2005), pp. 265–275. 60, 61

[ED04] Eisemann E., Durand F.: Flash photography enhancement via intrinsic relighting. In *ACM Transactions on Graphics (Proceedings of Siggraph Conference)* (2004), vol. 23, ACM Press. 62, 63, 182

[ED06] Eisemann E., Décoret X.: Fast scene voxelization and applications. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2006), pp. 71–78. 41, 58, 112, 113

[ED08] Eisemann E., Décoret X.: Single-pass gpu solid voxelization and applications. In *GI '08: Proceedings of Graphics Interface 2008* (2008), vol. 322, pp. 73–80. 58

[Eve01] Everitt C.: Introduction interactive order-independent transparency. 59

[FD09] Fabianowski B., Dingliana J.: Interactive global photon mapping. *Computer Graphics Forum (Proc. of EGSR) 28*, 4 (2009), 1151–1159. 64

[FPSG97] Ferwerda J. A., Pattanaik S. N., Shirley P. S., Greenberg D. P.: A model of visual masking for computer graphics. In *Proceedings of SIGGRAPH* (1997), pp. 143–152. 52, 162, 167

[GKBP05] Gautron P., Křivánek J., Bouatoch K., Pattanaik S.: Radiance cache splatting: A gpu-friendly global illumination algorithm. In *Rendering Techniques* (2005), Eurographics Symposium on Rendering, pp. 55–64. 60, 86, 89, 137

[GKD07] Green P., Kautz J., Durand F.: Efficient reflectance and visibility approximations for environment map rendering. In *Computer Graphics Forum (Proc. of Eurographics)* (Prague, Czech Republic, 2007), Cohen-Or D., Slavik P., (Eds.), vol. 26(3), pp. 495–502. 108

[GKPB04] Gautron P., Křivánek J., Pattanaik S. N., Bouatouch K.: A novel hemispherical basis for accurate and efficient rendering. In *Rendering Techniques* (2004), Eurographics Symposium on Rendering, pp. 321–330. 60, 129

[GSHG98] Greger G., Shirley P., Hubbard P. M., Greenberg D. P.: The Irradiance Volume. *IEEE Comput. Graph. and Appl. 18*, 2 (1998), 32–43. 60, 104

[Hac05] Hachisuka T.: High-quality global illumination rendering using rasterization. *GPU Gems 2* (2005), 615–633. 59

[HAM07] Hasselgren J., Akenine-Möller T.: PCU: the programmable culling unit. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers* (New York, NY, USA, 2007), ACM, p. 92. 61

[HBHS05] Havran V., Bittner J., Herzog R., Seidel H.-P.: Ray maps for global illumination. In *Rendering Techniques* (Konstanz, Germany, June 2005), Bala K., Dutre P., (Eds.), Eurographics Symposium on Rendering, pp. 43–54,311. 56, 65, 66, 67, 70, 77, 78, 107, 201

[HDMS03] Havran V., Damez C., Myszkowski K., Seidel H.-P.: An efficient spatio-temporal architecture for animation rendering. In *Eurographics Symposium on Rendering* (2003), pp. 106–117. 162

[HEMS10] Herzog R., Eisemann E., Myszkowski K., Seidel H.-P.: Spatio-Temporal Upsampling on the GPU. In *Proceedings ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D) 2010* (February 2010), pp. 91–98. 25, 27

[HHK*07a] Herzog R., Havran V., Kinuwaki S., Myszkowski K., Seidel H.-P.: Global illumination using photon ray splatting. In *Computer Graphics Forum (Proc. of Eurographics)* (Prague, Czech Republic, 2007), Cohen-Or D., Slavik P., (Eds.), vol. 26(3), The

European Association for Computer Graphics, Blackwell, pp. 503–513. 25, 66

[HHK*07b] Herzog R., Havran V., Kinuwaki S., Myszkowski K., Seidel H.-P.: *Global Illumination using Photon Ray Splatting*. Research Report MPI-I-2007-4-007, MPI Informatik, Saarbrücken, Germany, May 2007. 85

[HHS05] Havran V., Herzog R., Seidel H.-P.: Fast final gathering via reverse photon mapping. *Computer Graphics Forum (Proc. of Eurographics) 24*, 3 (2005), 323–333. 56, 66, 77, 81, 95, 98, 137, 202

[HJ09] Hachisuka T., Jensen H. W.: Stochastic progressive photon mapping. *ACM Transactions on Graphics 28*, 5 (2009). 96

[HJJ10] Hachisuka T., Jarosz W., Jensen H. W.: A progressive error estimation framework for photon density estimation. *ACM Transactions on Graphics (Proc. of SIGGRAPH Asia '10)* (2010). 95, 96, 97

[HKMS08] Herzog R., Kinuwaki S., Myszkowski K., Seidel H.-P.: Render2MPEG: a perception-based framework towards integrating rendering and video compression. In *Computer Graphics Forum (Proc. Eurographics)* (Crete, Greece, 2008), Scopigno R., Gröller E., (Eds.), vol. 27(2), Blackwell, pp. 183–192. 25, 26, 152

[HKWB09] Hašan M., Křivánek J., Walter B., Bala K.: Virtual spherical lights for many-light rendering of glossy scenes. *ACM Transactions on Graphics (Proc. of SIGGRAPH Asia '09) 28*, 5 (2009). 58

[HMS09] Herzog R., Myszkowski K., Seidel H.-P.: Anisotropic radiance-cache splatting for efficiently computing high-quality global illumination with lightcuts. In *Computer Graphics Forum (Proc. Eurographics)* (München, Germany, 2009), Stamminger M., Dutré P., (Eds.), vol. 28(2), Wiley-Blackwell, pp. 259–268. 25, 26

[HOJ08] Hachisuka T., Ogaki S., Jensen H. W.: Progressive photon mapping. *ACM Transactions on Graphics (Proc. of SIGGRAPH Asia '08) 27*, 5 (2008). 56, 95

[HPB06] Hašan M., Pellacini F., Bala K.: Direct-to-indirect transfer for cinematic relighting. *ACM Transactions on Graphics 25*, 3 (July 2006), 1089–1097. 165

[HS07] Herzog R., Seidel H.-P.: Lighting details preserving photon density estimation. In *The 15th Pacific Conference on Computer Graphics and Application* (Maui, Hawaii, USA, October 2007), Alexa M., Ju T., Gortler S., (Eds.), IEEE Computer Society, IEEE Computer Society, pp. 407–410. 25, 26

[HW02] Haumont D., Warze N.: Complete polygonal scene voxelization. *Journal of Graphics Tools 7*, 3 (2002), 27–41. 58

[Jen96] Jensen H. W.: Global Illumination using Photon Maps. In *Rendering Techniques '96* (1996), Proceedings of the Seventh Eurographics Workshop on Rendering, pp. 21–30. 39, 55, 57

[Jen97] Jensen H. W.: Rendering caustics on non-lambertian surfaces. *Computer Graphics Forum* (March 1997), 57–64. 81, 97

[Jen01] Jensen H. W.: *Realistic Image Synthesis Using Photon Mapping.* AK, Peters, 2001. 31, 55, 56, 60, 65, 66, 69, 70, 77, 80, 93, 96, 104, 114, 121, 123, 125, 163, 201

[JSCK02] Jensen H. W., Suykens F., Christensen P. H., Kato T.: A practical guide to global illumination using photon mapping. 93–121. SIGGRAPH Course Note. 155

[JZJ08] Jarosz W., Zwicker M., Jensen H. W.: The Beam Radiance Estimate for Volumetric Photon Mapping. *Computer Graphics Forum (Proc. of Eurographics) 27*, 2 (4 2008), 557–566. 64

[Kaj86] Kajiya J. T.: The rendering equation. In *Proceedings of ACM SIGGRAPH* (1986), ACM Press, pp. 143–150. 44, 120

[KBPv06] Křivánek J., Bouatouch K., Pattanaik S. N., Žára J.: Making radiance and irradiance caching practical: Adaptive caching and neighbor clamping. In *Rendering Techniques 2006, Eurographics Symposium on Rendering* (Nicosia, Cyprus, June 2006), Eurographics Association, pp. 127–138. 60, 88, 123, 138, 139, 141, 152, 153, 202

[KBW06] Krüger J., Bürger K., Westermann R.: Interactive screen-space accurate photon tracing on GPUs. In *Rendering Techniques* (2006), Eurographics Symposium on Rendering, pp. 319–329. 41

[KCCo00] Koltun V., Chrysanthou Y., Cohen-or D.: Virtual occluders: An efficient intermediate pvs representation. In *11th Eurographics Workshop on Rendering* (2000), pp. 59–70. 61

[KCLU07] Kopf J., Cohen M. F., Lischinski D., Uyttendaele M.: Joint bilateral upsampling. *ACM Transactions on Graphics 26*, 3 (July 2007), 96:1–96:5. 62, 63, 182

[Kel96] Keller A.: Quasi-Monte Carlo Radiosity. In *Rendering Techniques '96* (1996), pp. 101–110. 47, 58

[Kel97] Keller A.: Instant radiosity. In *Proceedings of SIGGRAPH* (1997), pp. 49–56. 57, 59, 75, 111, 121, 123, 125, 128, 161, 163, 165, 166

[KGPB05] Křivánek J., Gautron P., Pattanaik S., Bouatouch K.: Radiance caching for efficient global illumination computation. *IEEE TVCG 11*, 5 (2005), 550 – 561. 26, 60, 87, 123, 129, 131, 132, 133, 134, 158, 163, 173

[KK04] Kollig T., Keller A.: Illumination in the presence of weak singularities. *Monte Carlo and Quasi-Monte Carlo Methods* (2004), 245–257. 58, 143

[K0̌5a] Křivánek J.: *Radiance Caching for Global Illumination Computation on Glossy Surfaces.* Ph.D. thesis, Université de Rennes 1 and Czech Technical University in Prague, December 2005. 23, 133, 159

[K0̌5b] Křivánek J.: *Radiance Caching for Global Illumination Computation on Glossy Surfaces.* Ph.d. thesis, Université de Rennes 1 and Czech Technical University in Prague, December 2005. 67, 83, 86, 87, 90, 91, 130, 133, 153, 155, 156, 158

[KW86] Kalos M., Whitlock P.: *The Monte Carlo Method, Volume 1: Basics.* John Wiley and Sons, New York, 1986. 46

[KW00] Keller A., Wald I.: Efficient Importance Sampling Techniques for the Photon Map. In *Proc. Vision, Modelling and Visualization* (2000), pp. 271–279. 46

[LC04] Larsen B. D., Christensen N. J.: Simulating photon mapping for real-time applications. In *Rendering Techniques 2004: 15th Eurographics Workshop on Rendering* (2004), pp. 123–132. 60

[LFTG97] Lafortune E., Foo S.-C., Torrance K., Greenberg D.: Non-linear approximation of reflectance functions. In *Proceedings of SIGGRAPH '97* (1997), ACM, pp. 117–126. 39

[LP03] Lavignotte F., Paulin M.: Scalable photon splatting for global illumination. In *GRAPHITE 2003* (2003), ACM SIGGRAPH, pp. 1–11. 56, 66, 81

[LURM02] Lastra M., Urena C., Revelles J., Montes R.: A particle-path based method for monte carlo density estimation. In *In Poster Papers Proceeding of the 13th Eurographics Workshop on Rendering* (June 2002), pp. 33–40. 56, 65, 66, 67, 77

[LV00] Lokovic T., Veach E.: Deep shadow maps. In *Proceedings of ACM SIGGRAPH '00* (2000), pp. 385–392. 59

[LW85] Levoy M., Whitted T.: *The Use of Points as a Display Primitive.* Tech. Rep. 85-022, Computer Science Department, University of North Carolina at Chapel Hill, January 1985. 41

[LW93] Lafortune E. P., Willems Y. D.: Bi-Directional Path Tracing. In *Compugraphics 93* (1993), pp. 145–153. 57

[LZT*08] Lehtinen J., Zwicker M., Turquin E., Kontkanen J., Durand F., Sillion F., Aila T.: A meshless hierarchical representation for light transport. 41

[Mac48] MacRobert T.: *Spherical Harmonics; An Elementary Treatise on Harmonic Functions, with Applications.* Dover Publications, 1948. 60, 82

[Mal02] Malgouyres R.: A discrete radiosity method. In *DGCI '02: Proceedings of the 10th International Conference on Discrete Geometry for Computer Imagery* (2002), Springer-Verlag, pp. 428–438. 59

[MB95] McMillan L., Bishop G.: Plenoptic modeling: An image-based rendering system. In *Proceedings of SIGGRAPH* (1995), pp. 39–46. 162

[McC99] McCool M. D.: Anisotropic diffusion for monte carlo noise reduction. *ACM Transactions on Graphics 18*, 2 (1999), 171–194. 67, 85, 138

[ML09] McGuire M., Luebke D.: Hardware-accelerated global illumination by image space photon mapping. In *Proceedings of the 2009 ACM SIGGRAPH/EuroGraphics conference on High Performance Graphics* (New York, NY, USA, August 2009), ACM. 64

[MMB97] Mark W., McMillan L., Bishop G.: Post-rendering 3D warping. In *1997 Symposium on Interactive 3D Graphics* (1997), ACM SIGGRAPH, pp. 7–16. 61

[MMS*05] Müller G., Meseth J., Sattler M., Sarlette R., Klein R.: Acquisition, Synthesis, and Rendering of Bidirectional Texture Functions. In *Computer Graphics Forum* (2005), pp. 83–109. 36

[MPE] MPEG-2: Free mpeg-2 encoder software. http://www.mpeg.org/MPEG/video/. 167, 173, 177

[MTAS01] Myszkowski K., Tawara T., Akamine H., Seidel H.-P.: Perception-guided global illumination solution for animation rendering. In *Proceedings of SIGGRAPH '01* (2001), ACM, pp. 221–230. 52, 63

[Mys97] Myszkowski K.: Lighting reconstruction using fast and adaptive density estimation techniques. In *Rendering Techniques* (1997), pp. 251–262. 66

[Nie92] Niederreiter H.: Random Number Generation and Quasi-Monte Carlo Methods. In *CBMS-NSF Regional Conference Series in Appl. Math.* (Philadelphia: SIAM, 1992), vol. 63. 47

[NKGR06] Nayar S. K., Krishnan G., Grossberg M. D., Raskar R.: Fast separation of direct and global components of a scene using high frequency illumination. *ACM Trans. on Graph. (Proc. of SIGGRAPH '06) 25*, 3 (2006), 935–944. 23, 36

[NSL*07] Nehab D., Sander P. V., Lawrence J., Tatarchuk N., Isidoro J. R.: Accelerating real-time shading with reverse reprojection caching. In *Graphics Hardware* (2007). 41, 61, 62, 183, 184, 185

[Oat05] Oat C.: Irradiance volumes for games (game developers conference), 2005. http://ati.amd.com/developer/gdc/GDC2005_PracticalPRT.pdf. 60, 104

[PH04] Pharr M., Humphreys G.: *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann, 2004. 40

[Pho75] Phong B. T.: Illumination for computer generated pictures. *Communication of the ACM 18*, 6 (1975), 311–317. 38, 39

[PM90] Perona P., Malik J.: Scale-space and edge detection using anisotropic diffusion. In *IEEE Transactions on Pattern Analysis and Machine Intelligence* (1990), vol. 12, pp. 629–639. 140

[Poy98] Poynton C.: The rehabilitation of gamma. *Human Vision and Electronic Imaging III (Proc. of SPIE)* (1998), 232–249. 51

[PSA*04] Petschnigg G., Szeliski R., Agrawala M., Cohen M., Hoppe H., Toyama K.: Digital photography with flash and no-flash image pairs. *ACM Transactions on Graphics (Proceedings of Siggraph Conference) 23*, 3 (2004), 664–672. 62, 63, 182

[REG*09] Ritschel T., Engelhardt T., Grosch T., Seidel H.-P., Kautz J., Dachsbacher C.: Micro-rendering for scalable, parallel final gathering. *Proceedings of SIGGRAPH Asia '09* (2009). 41

[RGK*08] Ritschel T., Grosch T., Kim M. H., Seidel H.-P., Dachsbacher C., Kautz J.: Imperfect Shadow Maps for Efficient Computation of Indirect Illumination. *ACM Trans. Graph. (Proc. of SIGGRAPH ASIA 2008) 27*, 5 (2008). 41, 58, 59, 108

[RGS09] Ritschel T., Grosch T., Seidel H.-P.: Approximating Dynamic Global Illumination in Image Space. In *Proceedings ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D) 2009* (2009). 41, 194

[RH04] Ramamoorthi R., Hanrahan P.: A signal-processing framework for reflection. *ACM Transactions on Graphics 23*, 4 (2004), 1004–1042. 82, 85, 130

[RPG99] Ramasubramanian M., Pattanaik S. N., Greenberg D. P.: A perceptually based physical error metric for realistic image synthesis. In *Proceedings of SIGGRAPH* (1999), pp. 73–82. 52, 63, 165

[RS91] Rao A. R., Schunck B. G.: Computing oriented texture fields. In *Proceedings of CVGIP* (1991), vol. 53, pp. 157–185. 93

[RSC87] Reeves W. T., Salesin D. H., Cook R. L.: Rendering antialiased shadows with depth maps. In *Proc. of Siggraph'87* (1987). 41, 193

[RW96] Rosenholtz R., Watson A. B.: Perceptual adaptive JPEG coding. In *IEEE International Conference on Image Processing* (1996), pp. 901–904. 152, 169

[RWPD05] Reinhard E., Ward G., Pattanaik S., Debevec P.: *High Dynamic Range Imaging: Acquisition, Display, and Image-Based Lighting*. Morgan Kaufmann, 2005. 54

[SaLY*08a] Sitthi-amorn P., Lawrence J., Yang L., Sander P. V., Nehab D.: An improved shading cache for modern gpus. In *Graphics Hardware* (2008). 41, 61, 184, 185, 193

[SaLY*08b] Sitthi-amorn P., Lawrence J., Yang L., Sander P. V., Nehab D., Xi J.: Automated reprojection-based pixel shader optimization. In *SIGGRAPH Asia '08: ACM SIGGRAPH Asia 2008 papers* (2008), ACM, pp. 1–11. 62, 190

[SBS94] Sain S. R., Baggerly K. A., Scott D. W.: Cross-validation of Multivariate Densities. In *J. Am. Statist. Assoc.* (1994), vol. 89, pp. 807–817. 56

[Sch94] Schlick C.: An inexpensive BRDF model for physically-based rendering. *Computer Graphics Forum 9*, 13 (1994), 233–246. 38

[Sch03] Schregle R.: Bias compensation for photon maps. *Computer Graphics Forum 22*, 4 (2003), 729–742. 55, 56, 65, 66, 73, 107

[SGwHS98] Shade J., Gortler S., wei He L., Szeliski R.: Layered depth images. In *Proceedings of ACM SIGGRAPH '98* (1998), pp. 231–242. 59

[Shi93] Shinya M.: Spatial Anti-aliasing for Animation Sequences with Spatio-temporal Filtering. In *Proceedings of SIGGRAPH '93* (1993), pp. 289–296. 63

[Sil85] Silverman B.: *Density Estimation for Statistics and Data Analysis*. Chapmann and Hall, London, 1985. 55, 56, 65, 66, 69, 72, 73, 74, 76

[SIMP06] Segovia B., Iehl J. C., Mitanchey R., Péroche B.: Non-interleaved deferred shading of interleaved sample patterns. In *Graphics Hardware* (2006), Olano M., Slusallek P., (Eds.). 60, 63

[SJW07] Scherzer D., Jeschke S., Wimmer M.: Pixel-correct shadow maps with temporal reprojection and shadow test confidence. In *Rendering Techniques* (2007), pp. 45–50. 61, 181

[SKALP05] Szirmay-Kalos L., Aszódi B., Lazányi I., Premecz M.: Approximate ray-tracing on the gpu with distance impostors. *Computer Graphics Forum* (2005), 695–704. 57

[SKDM05] Smyk M., Kinuwaki S., Durikovic R., Myszkowski K.: Temporally Coherent Irradiance Caching for High Quality Animation Rendering. In *Proceedings of Eurographics* (2005), vol. 24, pp. 401–412. 103, 165

[SKP98] Szirmay-Kalos L., Purgathofer W.: Global ray-bundle tracing with hardware acceleration. In *Rendering Techniques* (1998), pp. 247–258. 59

[SKP06] Shah M. A., Konttinen J., Pattanaik S. N.: Caustics mapping: An image-space technique for real-time. *IEEE Transactions on Visualization and Computer Graphics* (2006). 57

[SM02] Smyk M., Myszkowski K.: Quality improvements for indirect illumination interpolation. *Proceedings of the International Conference on Computer Vision and Graphics* (2002), 685–692. 59

[ST87] Scott D. W., Terrell G. R.: Biased and Unbiased Cross-validation in Density Estimation. In *J. Amer. Statist. Assoc.* (1987), vol. 82, pp. 1131–1146. 56

[Stü98] Stürzlinger W.: Calculating global illumination for glossy surfaces. *Computers & Graphics 22*, 2-3 (1998), 175–180. 81, 83

[SW00] Suykens F., Willems Y. D.: Adaptive filtering of progressive monte carlo image rendering. In *Proceedings of WSCG 2000* (2000). 57, 75

[SW01] Suykens F., Willems Y. D.: Path differentials and applications. In *12th Eurographics Workshop on Rendering* (2001), Springer-Verlag, pp. 257–268. 64

[SWZ96] Shirley P., Wang C., Zimmerman K.: Monte Carlo Techniques for Direct Lighting Calculations. In *ACM Transactions on Graphics* (1996), pp. 1–36. 46

[Tek95] Tekalp A. M.: *Digital video Processing*. Prentice Hall, 1995. 63

[TL04] Tabellion E., Lamorlette A.: An approximate global illumination system for computer generated films. In *Proceedings of SIGGRAPH* (2004), vol. 23, pp. 469–476. 24, 60, 87, 123, 139

[Vea97] Veach E.: *Robust Monte Carlo Methods for Light Transport Simulation*. PhD thesis, Stanford University, December 1997. Available from http://graphics.stanford.edu/papers/veach_thesis/. 46, 150, 151

[WABG06] Walter B., Arbree A., Bala K., Greenberg D. P.: Multidimensional lightcuts. *ACM Transactions on Graphics 25*, 3 (2006), 1081–1088. 60, 124, 161, 163, 165

[War92] Ward G. J.: Measuring and Modeling Anisotropic Reflection. In *Computer Graphics* (1992), vol. 26, Proceedings of SIGGRAPH '92, pp. 265–272. 38

[Wat93] Watson A.: DCT quantization matrices visually optimized for individual images. In *Human Vision, VisualProcessing, and Digital Display IV* (1993), SPIE, volume 1913-14, pp. 202–216. 167, 168, 169

[WBKP08] Walter B., Bala K., Kulkarni M., Pingali K.: Fast agglomerative clustering for rendering. In *IEEE Symposium on Interactive Ray Tracing (IRT)* (2008). 148

[WD06] Wyman C., Davis S.: Interactive image-space techniques for approximating caustics. In *Proceedings of the ACM Symposium on Interactive 3D Graphics and Games* (2006), pp. 153–160. 57

[WDP99] Walter B., Drettakis G., Parker S.: Interactive Rendering using the Render Cache. In *Proceedings of the 10th Eurographics Workshop on Rendering* (1999), pp. 235–246. 60, 61

[Web] Webresource: Hemispherical harmonics source code. http://www.cgg.cvut.cz/. 153

[Web08] Website: Render2mpeg project, 2008. http://www.mpi-inf.mpg.de/resources/anim/EG08/. 173, 176

[Web10] Website: Spatio-temporal upsampling project, 2010. http://www.mpi-inf.mpg.de/∼rherzog/. 193

[Wei98] Weickert J.: *Anisotropic diffusion in image processing*. Teubner, Stuttgart, 1998. 93, 138, 140

[WEV02] Ward G., Eydelberg-Vileshin E.: Picture Perfect RGB Rendering Using Spectral Prefiltering and Sharp Color Primaries. In *Thirteenth Eurographics Workshop on Rendering* (2002). 50

[WFA*05] Walter B., Fernandez S., Arbree A., Bala K., Donikian M., Greenberg D.: Lightcuts: A scalable approach to illumination. In *Proceedings of ACM SIGGRAPH* (2005), pp. 1098 – 1107. 26, 28, 58, 60, 110, 119, 120, 122, 124, 126, 129, 147, 153, 158, 161, 163, 165, 166, 167, 170, 171, 173, 202

[WGS04] Wald I., Günther J., Slusallek P.: Balancing Considered Harmful – Faster Photon Mapping using the Voxel Volume Heuristic. vol. 22, pp. 595–603. (Proceedings of Eurographics). 77

[WH92] Ward G. J., Heckbert P.: Irradiance Gradients. In *Rendering Techniques '92* (1992), Eurographics Symposium on Rendering, pp. 85–98. 59, 91, 124, 130, 132, 133, 135, 158

[WHSG97] Walter B., Hubbard P. M., Shirley P., Greenberg D. P.: Global illumination using local linear density estimation. *ACM Trans. Graph. 16*, 3 (1997), 217–259. 66

[WJ95] Wand M., Jones M.: *Kernel Smoothing*. Chapman and Hall, 1995. 56

[WKB*02] Wald I., Kollig T., Benthin C., Keller A., Slusallek P.: Interactive global illumination using fast ray tracing. In *Eurographics Workshop on Rendering* (2002), pp. 15–24. 60, 63, 67, 85, 104, 161, 164, 166, 178

[WKC94] Wallach D. S., Kunapalli S., Cohen M. F.: Accelerated mpeg compression of dynamic polygonal scenes. In *Proceedings of SIGGRAPH* (1994), pp. 193–197. 162

[WMM*04a] Weber M., Milch M., Myszkowski K., Dmitriev K., Rokita P., Seidel H.-P.: Spatio-temporal photon density estimation using bilateral filtering. In *Computer Graphics International (CGI 2004)* (2004), Cohen-Or D., Jain L., Magnenat-Thalmann N., (Eds.), pp. 120–127. 63

[WMM*04b] Weber M., Milch M., Myszkowski K., Dmitriev K., Rokita P., Seidel H.-P.: Spatio-temporal photon density estimation using bilateral filtering. In *Computer Graphics International* (2004), pp. 120–127. 103

[WPG02] Walter B., Pattanaik S. N., Greenberg D. P.: Using perceptual texture masking for efficient image synthesis. *Computer Graphics Forum 21*, 3 (2002), 393–399. 52, 64, 162, 170

[WRC88] Ward G. J., Rubinstein F. M., Clear R. D.: A ray tracing solution for diffuse interreflection. In *Computer Graphics* (1988), Proceedings of ACM SIGGRAPH '88, pp. 85–92. 26, 59, 60, 86, 87, 88, 89, 90, 91, 94, 95, 123, 124, 129, 137, 138, 139, 150, 163, 173, 201

[YNS*09] Yang L., Nehab D., Sander P. V., Sitthi-amorn P., Lawrence J., Hoppe H.: Amortized supersampling. *ACM Transactions on Graphics (Proc. of SIGGRAPH Asia 2009) 28*, 5 (2009), 135. 41, 62, 192, 198

[YPG01] Yee H., Pattanaik S., Greenberg D.: Spatiotemporal Sensitivity and Visual Attention for Efficient Rendering of Dynamic Environments. *ACM Transactions on Graphics 20*, 1 (January 2001), 39–65. 63

[YSL08] Yang L., Sander P. V., Lawrence J.: Geometry-aware framebuffer level of detail. In *Rendering Techniques* (2008). 62, 181, 182, 183, 184, 187, 193, 197

[ZDL00] Zeng W., Daly S., Lei S.: Visual optimization tools in JPEG 2000. In *IEEE Intern. Conf. on Image Processing* (2000), pp. 37–40. 168

[ZPvBG01] Zwicker M., Pfister H., van Baar J., Gross M.: Surface splatting. In *Proceedings of SIGGRAPH* (2001), pp. 371–378. 108, 138

Index