FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Interactive, Progressive Photon Tracing using a Multi-Resolution Image-Filtering Approach

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Visual Computing

eingereicht von

Katharina Krösl, BSc
Matrikelnummer 0325089

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Mitwirkung: Dipl.-Ing. Christian Luksch, BSc

Wien, 24. März 2016

Katharina Krösl	Michael Wimmer

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Interactive, Progressive Photon Tracing using a Multi-Resolution Image-Filtering Approach

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieurin

in

Visual Computing

by

Katharina Krösl, BSc
Registration Number 0325089

to the Faculty of Informatics

at the Vienna University of Technology

Advisor: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Assistance: Dipl.-Ing. Christian Luksch, BSc

Vienna, 24th March, 2016

Katharina Krösl | Michael Wimmer

Erklärung zur Verfassung der Arbeit

Katharina Krösl, BSc
Gentzgasse 14-20/7/8, 1180 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 24. März 2016

Katharina Krösl

Acknowledgements

First of all, I want to thank VRVis Zentrum für Virtual Reality und Visualisierung Forschungs-GmbH for the opportunity to develop the prototype for my thesis in an applied research project bridging academia and industry.

I also want to thank my supervisor Michael Wimmer for always sending me very helpful and constructive feedback, even during his free time and on holidays, motivating me to never stop improving my thesis. My special thanks go to Christian Luksch for his ongoing support during the implementation phase of my prototype and numerous brainstorming sessions when I got stuck.

Furthermore, I want to thank my colleague Bernhard Steiner for taking the time to proof read early stages of my thesis and for giving me important advice right from the start. A big thank you goes to Thomas Rausch, for his constant moral support, that kept me going even through very stressful times, and the endless hours of proof reading my thesis. Last but not least, I want to thank my parents for always supporting me throughout my studies and providing me with countless home-cooked meals that prevented me from starving during the end phase of writing my thesis.

Kurzfassung

Moderne Arbeitsabläufe in Architektur und professioneller Gebäudebeleuchtung erfordern physikalisch korrekte Lichtsimulationen für detaillierte und komplexe 3D-Modelle. Aktuelle Arbeitsabläufe für Leuchten-Entwicklung und Beleuchtungsplanung sind nicht aufeinander abgestimmt. Bei der Leuchten-Entwicklung werden CAD-Programme verwendet, um 3D-Modelle von Leuchten zu erstellen und Offline-Rendering-Tools, um die Lichtverteilung dieser Modelle zu visualisieren. Mit Hilfe einer interaktiven Lichtplanungssoftware können während der Beleuchtungsplanung Lichtkonzepte entworfen werden, indem Modelle von zuvor erstellten Leuchten in einer 3D-Szene platziert werden. Zu diesem Zeitpunkt ist es nicht mehr möglich, die Lichtverteilung der während der Leuchten-Entwicklung erstellten Lichtquellen zu modifizieren.

In dieser Diplomarbeit wird ein interaktiver, globaler Beleuchtungs-Algorithmus vorgestellt, der die Lichtverteilung einer Leuchte simuliert. Der Algorithmus erzeugt optisch ansprechende Zwischenresultate bei interaktiven Bildwiederholraten, bevor er zu einer physikalisch plausiblen Lösung konvergiert. Das Endresultat beschreibt die Lichtausbreitung einer Leuchte und kann als solches direkt in eine Lichtplanungssoftware importiert werden. Wir kombinieren einen interaktiven, progressiven Photon-Tracing-Algorithmus mit einem Multi-Resolution Bild-Filter. Unser Algorithmus emittiert Photonen in eine 3D-Szene eines Leuchten-Modells und verbessert die Ergebnisse mit jeder Iteration. Wir verwenden Mipmaps, um verschiedene Auflösungen einer Textur zu erzeugen und integrieren Filter-Techniken, um optisch ansprechende Zwischenergebnisse zu erhalten.

Evaluierungen auf Grundlage objektiver Qualitätsmetriken zeigen, dass der präsentierte Ansatz die Bildqualität im Vergleich zu ungefilterten Ergebnissen erhöht. Der vorgestellte Algorithmus ermöglicht eine schnelle Vorschau auf die Endresultate und erlaubt es, Geometrie und Materialeigenschaften einer Leuchte in Echtzeit zu editieren. Dies reduziert die Zeit zwischen einzelnen Modifikatios-Iterationen und macht Leuchten-Entwicklung damit zu einem interaktiven Prozess, wodurch die Gesamtproduktionszeit verringert wird. Darüber hinaus integriert der vorgestellte Ansatz Leuchten-Entwicklung in Beleuchtungsplanung und stellt somit eine neue Methode dar, um zwei ehemals getrennte Arbeitsabläufe zu verbinden.

Abstract

Modern workflows in architectural planning and lighting design require physically reliable lighting simulations for detailed and complex 3D models. Current workflows for luminaire design and lighting design are not tailored to each other. During luminaire design, CAD programs are used to create 3D models of luminaires, and offline rendering tools are used to visualize the light distribution. In lighting design, light concepts are explored by placing light sources - previously created during luminaire design - in a 3D scene using an interactive light-planning software, but it is not possible to modify the light sources themselves.

This thesis presents an interactive global-illumination algorithm to simulate the light distribution of a luminaire. The algorithm produces visually pleasing intermediate results at interactive frame rates, before converging to a physically plausible solution that can be imported as a representation of a light source into a light-planning software. We combine an interactive, progressive photon-tracing algorithm with a multi-resolution image-filtering approach. Our algorithm iteratively emits photons into a 3D scene containing the model of a luminaire and progressively refines results. We use mipmaps to create a multi-resolution approach and incorporate image-filtering techniques to obtain visually pleasing intermediate results.

Evaluations based on objective quality metrics show that the presented image-filtering approach increases image quality when compared to non-filtered results.

The proposed algorithm provides fast previews and allows interactive modifications of the geometry and material properties of the luminaire in real time. This reduces time between modification iterations and therefore turns luminaire design into an interactive process that reduces overall production time. Furthermore, the presented approach integrates luminaire design into lighting design and therefore provides a new way to combine two former decoupled workflows.

Contents

CHAPTER 1

Introduction

1.1 Motivation

Photorealism has become more and more important in modern computer graphics, for example in industrial applications that rely on high realism and physically correct simulations. Modern workflows in architectural planning and lighting design require physically reliable lighting simulations for very detailed and complex 3D models like shown in Figure 1.1. Therefore, offline rendering is common practice in this area, especially in luminaire design. Due to the high complexity of offline rendering algorithms, they are time consuming and thus increase production time and financial cost. Interactive solutions would make the luminaire and lighting design process more flexible and faster and so reduce production time.

There has been a lot of research in the area of global illumination and realistic lighting, and some real-time algorithms to simulate light distributions produce visually convincing results. However, for physically plausible solutions, only offline rendering algorithms are sufficiently accurate, which makes the development of an interactive yet physically plausible lighting solution a challenging task.

1.2 Problem Statement

In architectural planning and lighting design, light concepts are explored by placing light sources in a 3D scene using an interactive light-planning software. The light sources are represented by measured light-distribution curves of luminaires and are used as black boxes that can be positioned in a scene, but not edited in their light-distribution behavior.

The current workflows for luminaire design and lighting design are not tailored to each other. During the lighting-design process, it is not possible to modify the light sources themselves.

Figure 1.1: 3D scenes rendered with the light-planning software *HILITE*

During luminaire design, elaborate interactions of optics and complex materials have to be considered. The advent of light emitting diodes (LEDs) in industrial lighting solutions further increased the complexity, amount of LEDs that are built into a single luminaire, where each should be processed as an individual light source to get precise results for lighting simulations. 3D representations of luminaires are created using CAD programs for modeling, and raytracing tools are used to visualize the results. Because these visualizations have to be very accurate, they are usually slow and can thus be categorized as offline rendering methods. After each change to the 3D model of a luminaire, it is necessary to wait for the visualization result in order to see the corresponding effects. So the iterative development process of a luminaire, consisting of modeling and visualizing, is a time-consuming, challenging task.

1.3 Contributions

Physically correct lighting and shading is key to photorealistic rendering. As we aim to achieve more and more photorealism in computer graphics, global illumination gains in importance and is still an ongoing research topic. This thesis contributes mainly in the field of interactive global illumination, solving a practical problem for luminaire design and lighting design applications. The combination of image-filtering techniques that are used in computer vision, image processing and pattern recognition with state-of-the-art

rendering methods for global illumination, like photon tracing, unite two main topics of Visual Computing.

In this thesis we present an interactive global-illumination algorithm that simulates the light distribution of a light source and produces visually pleasing intermediate results at interactive frame rates before it converges to a physically correct solution, resulting in a visualization of the light distribution of a luminaire. To achieve this, we combine multi-resolution image-filtering techniques with an interactive, progressive photon-tracing algorithm.

Based on the concept of photon tracing [Jen01], the algorithm emits photons into a 3D scene containing one or multiple light sources. Using an importance function allows to direct the photons into directions with a lot of detail, like shadow borders, and use fewer photons for very homogeneous regions. Results are progressively refined until a physically accurate outcome is reached. Furthermore, the algorithm utilizes mipmaps to create a multi-resolution approach where state-of-the-art image-filtering techniques based on bilateral filtering [TM98] are applied. This gives our algorithm the ability to upsample image regions that have not yet received enough photons, according to a pre-defined threshold, and are therefore considered "not stable". Filtering techniques are used to interpolate from information on higher mipmap levels to create visually pleasing intermediate results at low photon counts and interactive frame rates. This approach is integrated into a light-source design and visualization tool that enables the user to interactively adapt parameters of the material or the geometry of the model, change properties of the reflectors or lenses and get the intermediate results rendered at interactive frame rates. Early results of this rendering process provide a preview of the light distribution of a specific luminaire and the possibility to instantly change the model without having to wait for the simulation to finish and converge to the ground truth. A comparison of our final results to the state of the art for physically correct global illumination shows that our method achieves comparable results in terms of quality in less computation time.

With our method we can reduce the development time of new luminaires for light manufacturers and create possibilities to develop novel workflows for companies that supply lighting solutions. To that end, our approach transforms the time-consuming, iterative light-source development into an interactive process. The results of our algorithm are representations of luminaires, that can directly be imported into and used as virtual light sources in a light-planning software.

Since our approach offers the possibility to modify light sources parallel to or after positioning them in a 3D scene, it combines the luminaire design and lighting-design process.

1.4 Structure of the Work

The following Chapter 2 gives an introduction to lglobal illumination techniques in computer graphics. Furthermore, the two major concepts, raytracing and radiosity, are presented, including some more advanced algorithms that build upon these basic concepts.

In Chapter 3, related work and current state-of-the-art approaches in photon mapping as well as image filtering are discussed and analyzed and we give an introduction to lighting design.

Chapter 4 gives an overview of our approach. The basic idea of our algorithm is presented and we describe the main components of our solution.

The first part of our algortihm is presented in detail in Chapter 5. We explain how our photon tracer operates and discuss the implemented material model as well as an importance sampling scheme that we explored.

Chapter 6 describes the multi-resolution image-filtering approach of our algorithm.

In Chapter 7, we explain the architecture of our application. Additionally, implementation details regarding the used languages, graphic APIs and frameworks are presented.

The results of our approach are shown and discussed in Chapter 8.

Finally, Chapter 9 gives a conclusion based on the presented results and an outlook of future work.

CHAPTER 2

Background

This chapter gives an introduction global-illumination methods and their evolution in computer graphics. Global illumination in computer graphics has always been a challenging problem. Numerous different approaches have been explored over the years, the two most prominent concepts being radiosity and ray tracing [Jen96]. Most of today's lighting approaches are based on one of these two concepts.

There have been several enhancements of these methods, some of which will be discussed in this chapter. In the 1990s researchers started to try to combine these two approaches, stating that a combination of both would give the best results, but there were some problems to overcome. Monte Carlo ray tracing, an enhancement to ray tracing, was considered as time-consuming, while producing noisy results. Radiosity on the other hand needed a lot of memory and was not capable of rendering specular reflections. Peter Shirley [Shi90] first presented a method to use ray tracing for shadow rendering as well as light raytracing for caustics in 1990. In 1991 Chen et al. [CRMT91] published their multi-pass method for global illumination using path tracing for all diffuse reflections that are visible from the view point, and radiosity only for rendering soft indirect illumination. This way, it was possible to get rid of artifacts that usually appeared when using radiosity. Another improvement shown by Rushmeier et al. [RPJV92] was a geometric simplification of the model, to reduce memory and speed up the rendering when using radiosity. Jensen [Jen96] then presented his approach which was to simplify the representation of the illumination instead of simplifying the geometry by using photon mapping to optimize the sampling directions, reduce the number of shadow rays, render caustics, and limit the number of reflections traced. A lot of other methods have been invented and research in this area is still proceeding. To give an overview about global illumination we will describe some of the most important models and some enhanced methods in the following.

2.1 Radiosity

The calculations for radiosity are based on the interaction between small surface patches that are assumed to be flat and diffuse [SAS92]. The main idea behind radiosity is to calculate bounces of light between diffuse surfaces to enable more realistic rendering of interreflections. Assuming that light bounces around in a scene, each surface is considered to be a light source, that can cause effects like color bleeding for example. To evaluate the light transfer between two patches a geometric value, called "form factor", that has a range from 0 to 1 is used. This value describes how much light travels directly from one patch to another. It is affected by the area of the patches, their distance and relative orientation. If one patch faces away from the other, the form factor is 0. In addition, a visibility factor is used to describe if two patches have a direct line of sight, or are blocked by another surface. The radiosity equation can then be expressed as a square matrix consisting of form factors among all patches in the scene. It is possible to solve this matrix using Gaussian elimination, which is very time consuming. There has been a lot of research to simplify the solution of this matrix [AMHH08].

Radiosity is a global solution, that has to be computed only once, because it is view independent. Furthermore it can simulate color bleeding. That also allows to pre-calculate it [SAS92].

The downside is that radiosity algorithms have problems with sharp edges like shadow borders, and they represent their result in a tessellated representation of the model [Jen01]. Therefore, disadvantages of radiosity arise in complex scenes due to the trade-off between scene complexity and speed. Also, there is only one radiosity value calculated for each patch, so this method only works for perfect diffuse materials [SAS92].

2.1.1 Progressive Refinement

To speed up radiosity, progressive refinement, as described by Smits et al. [SAS92], can be used. This method shoots energy from light sources into the scene. This energy is collected at each patch. The patch that received the most energy becomes the new light emitter and starts shooting energy back into the scene. Then the second brightest patch shoots. Patches, that have already shot all their energy back into the scene, might be refilled during the shooting of other patches. This process continues until a certain threshold is reached. Apart from less form factors, that need to be computed (only for one column of radiosity equation), a big advantage of this method is that it can produce preliminary results and then refine the solution over time.

2.1.2 Hierarchical Radiosity

Hierarchical Radiosity Algorithms [SAS92] subdivide patches until their form-factor is small. A tree-structure is generated and patches in this tree-structure are linked, if they are on the same level and their form-factor is small. Light transport is then only calculated between these patches. Radiosity is propagated through the whole tree-structure.

2.1.3 Importance-Driven Radiosity

The Algorithm presented by Smits et al. [SAS92] aims to speed up and improve the accuracy of radiosity. The authors point out two problems in traditional radiosity algorithms, they tried to overcome: Surfaces that are not contributing any additional information to the scene and that are still taken into account slow an algorithm unnecessarily down. This is over-solving globally. Also, there is the problem of undersolving locally. That means, that the algorithm has too low resolution to preserve small details in a scene. Under close inspection there might appear shadowing artifacts or false color bleeding. This problem exists due to the uniform precision radiosity algorithms usually use. So there is a trade-off between speed, or environment complexity, and accuracy.

The basic idea of Smits et al. [SAS92] to solve these problems is as follows. First, an adaptive subdivision scheme is used. Directly visible surfaces are refined in high accuracy, whereas surfaces that have only indirect effects are calculated in an accuracy based on their contribution. This whole process is guided not only by radiosity. The authors also introduce "importance" to guide the refinement steps.

This algorithm is an extension to hierarchical radiosity. It refines interactions, that are contributing the most to the scene. In order to detect these interactions, importance functions are used and an importance driven subdivision is performed. One challenge is to identify which interactions are significant enough. Hanrahan et al. [HSA91] introduced a brightness-weighted hierarchical radiosity algorithm, to meet this challenge. Their algorithm approximates insignificant interactions, but is view-independent, which still makes the calculation for complex scenes slow. To overcome this shortcoming Smits et al. [SAS92] incorporate a notion of view-dependence. Their algorithm uses visibility preprocessing, considering all potential interactions, but only computing them to an appropriate level of accuracy. To determine the level of accuracy needed, the algorithm uses two strategies, radiosity and importance and solves them simultaneously. The algorithm refines estimates of the transport equations where radiosity and importance are highest.

2.2 Ray Tracing

Ray tracing is a rendering method where rays are cast from the view point through the pixel grid into the scene to determine the visibility and lighting of objects. For the closest object a ray hits, the intersection point is calculated. To evaluate whether this point is directly illuminated by a light source or in shadow, so called "shadow rays" are cast directly to the light source to see if any objects block the light or not. Depending on the surface, the ray might also be reflected or refracted, in which case the reflected or refracted ray picks up the color of the next surface it hits and the new intersection point is again tested for shadows. This process is repeated until the ray hits a diffuse surface or a pre-defined threshold, that specifies the maximum number of bounces, is reached. This approach of tracing the rays further into the scene is an enhancement to classical

ray tracing, called Whitted ray tracing. This method can be used for direct illumination, but is not capable of rendering glossy and diffuse interreflections [AMHH08].

Ray tracing methods are widely used in offline rendering as they produce physically accurate results, but take a long time to render.

2.2.1 Monte Carlo Ray Tracing

Monte Carlo ray tracing is another enhancement to classical ray tracing that can also handle glossy and diffuse interreflections in a scene. A random direction is chosen for each reflection or refraction ray, influenced by the bidirectional reflectance distribution function (BRDF) of the surface [AMHH08]. Pure Monte Carlo ray tracing is unbiased, but very time consuming. The variance of Monte Carlo methods [Jen01], depicted as noise in the renderings, is reduced by an increasing number of samples. To half the error it is necessary to quadruple the samples. For standard Monte Carlo methods, best results are achieved with sampling formulas that yield random directions, which are distributed proportional to the cosine weighted BRDFs.

Importance Sampling

Another improvement is importance sampling [AMHH08], which weights the random directions of a monte carlo ray tracer according to their probability. For example, for a glossy surface, rays are more likely to be reflected in or close to the direct reflection direction.

Russian Roulette

During photon scattering, it is probabilistically decided if and what way a photon is reflected or transmitted, based on material properties. Russian Roulette [Jen01] is an importance sampling technique where the PDF (probability density function) is used to eliminate unimportant photons and therefore reduce work. The PDF is used to decide if another radiance estimate is evaluated. A percentage of the photons is reflected with full power, instead of reflecting all photons with a percentage of the power. The advantage is that we get photons with similar power in the photon map, but the disadvantage is that we increases variance, because we only sample reflection and transmission values.

2.2.2 Photon Mapping

Photon Mapping is a two-pass method for global illumination. In the first pass the photon map is created. The light sources emit packets of photons towards all objects into the scene. If a photon hits a diffuse surface it is stored in the photon map. If a photon hits a glossy or specular surface a Monte Carlo method like Russian Roulette is used to determine if this photon reflects on to another surfaces. The BRDF of a surface defines the reflection direction of photons. They then reflect further until they hit a diffuse surface and are stored in the photon map. In the end, a large number of photons

are stored, which gives an approximate representation of light in the scene [Jen96]. In the second pass the scene is rendered. Rays from the view point are traced through a pixel into the scene until they hit a surface. Around the intersection point the photon density is estimated. In classical photon mapping, a nearest neighbor search is performed to find the n nearest photons. The distance to the n^{th} photon represents the radius of a sphere centered at the intersection point. The flux evaluated by the sum of the n photons is then set in relation to the projected area of this sphere, to calculate an approximate indirect illumination [JC98].

2.3 Hybrid Methods

Bi-Directional Path Tracing

Path Tracing is a global-illumination technique, introduced by Kajiya [Kaj86], that uses radiosity for indirect illumination and Monte Carlo ray tracing for all other illumination. Monte Carlo ray tracing is generally slow to converge and entirely dependent on the viewing point. Ideally, an algorithm should also take the light sources into account on the same basis as the viewing point. Lafortune and Willems [LW93] presented a bi-directional path tracing algorithm to fulfill this requirement. The basic idea of their method is to shoot rays from the viewing point and from the light source at the same time into the scene. All intersection points are then connected with shadow rays. Using this method allows to take various lighting contributions and also secondary or tertiary light sources into account.

2.3.1 Instant Radiosity

Algorithms that are based on final gathering, like Instant Radiosity [Kel97], are alternatives to the photon density estimation. Similar to photon mapping, photons are emitted into the scene and stored where they hit a diffuse surface, after a couple of bounces. The difference is the second step, where every photon is treated as a virtual point light (VPL) and a shadow map or shadow volume is computed for each. Instant radiosity methods suffer from singularity problems that arise when calculating the contribution of a VPL to a pixel in the proximity of the VPL. This calculation involves a division by the squared distance between the VPL and the pixel. While in radiosity, the light transport between finite areas is calculated, instant radiosity replaces the per-area quantities by infinitely small points with intensity, so this term becomes unbound. As a solution, the term is clamped to a small number and therefore the method becomes biased [RDGK12].

CHAPTER 3

Related Work

In this chapter, we first discuss different photon mapping techniques, to solve global illumination, second, we present some related work concerning image filtering and third, we will give an introduction to lighting design at the example of a modern light-planning software.

3.1 State-of-the-Art Photon Mapping Approaches

According to a survey by Kang et al. [KWXM16], state-of-the-art approaches in photon mapping, like radiance estimation; photon relaxation; progressive photon mapping; or adaptive photon tracing, face three major challenges: smoothness of images; feature preservation; and space-savings in terms of low memory consumption, as shown in Figure 3.1. In the following we will discuss different approaches to tackle these challenges.

Radiance Estimation

Radiance-estimation methods evaluate the radiance that is reflected at a surface point by integrating over the hemisphere of incoming directions. Photon-mapping methods use k nearest neighbors (in the photon map) to estimate the radiance of a point:

$$L_r(x, \vec{\omega}) \approx \frac{1}{\pi r^2} \sum_{p=1}^{N} f_r(x, \vec{\omega}_p, \vec{\omega}) \Delta\Phi_p(x, \vec{\omega}_p)$$

The reflected radiance L_r at a point x in outgoing direction $\vec{\omega}$ is calculated by summing up the product of the bidirectional reflectance distribution function (BRDF) f_r and the power Φ_p of each photon p in an area πr^2 around x, that represents the projection of the hemisphere onto the surface [Jen01].

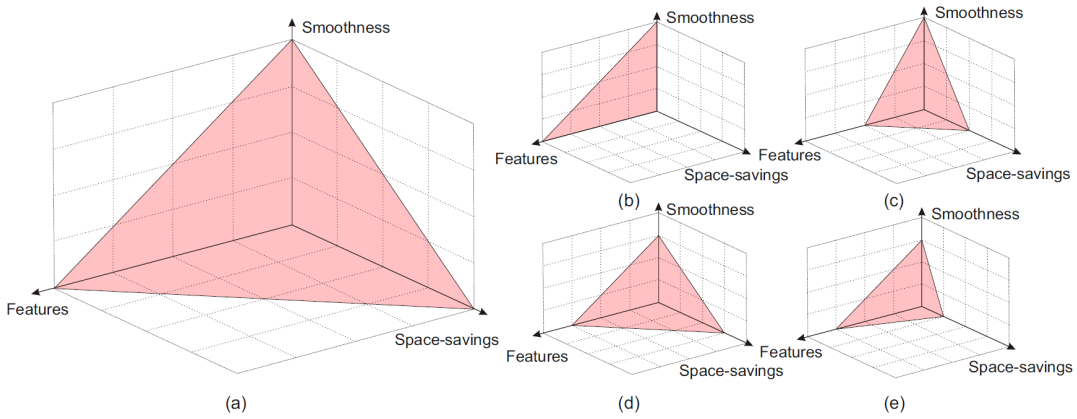


Figure 3.1: "Analysis of the optimized methods based on three challenges: (a) ideal solution; (b) radiance estimation methods; (c) photon relaxation methods; (d) progressive photon mapping methods; (e) photon tracing methods" Reprinted from [KWXM16].

Typically, improved photon mapping algorithms add a filter to the nearest neighbor density estimator in order to assign weights to photons according to their distance to the center point [Jen96] or other properties. The search bandwidth has an important role in these algorithms, because it balances bias and variance (which is visible as noise in the resulting images). By crossing boundaries of lighting features during a nearest neighbor density estimation, the bias is increased. To reduce this problem Jensen et al. [JC95] introduce differential checking. Other approaches use different shapes of filter kernels or adaptive kernels for nearest neighbor the density estimation. García et al. [GHUPS14] analyze common filter kernels for density estimation and present new estimators with improved filter kernels. Despite all improvements in this area, most radiance estimate methods still increase the computation time of photon mapping, due to elaborate computations of the radiance estimate.

Photon Relaxation

Photon-relaxation approaches, like presented by Spencer and Jones [SJ13a, SJ13b], try to distribute photons uniformly in a local area. Iterative point repulsion is used to move photons , relative to neighboring photons, and according to a displacement vector that depends on photon flux and photon direction. During photon relaxation, feature detection methods preserve high-frequency details in illumination information. It is possible to reduce noise in the results, by applying photon relaxation after photon tracing and before the illumination is calculated. Photon relaxation algorithms need only a small number of photons, and therefore reduce rendering time. However, because feature detection is difficult, photon relaxation is only used for caustic photon maps, which have very distinct feature.

Progressive Photon Mapping

Progressive photon mapping (PPM) was introduced by Hachisuka et al. [HOJ08] in order to reduce memory consumption when casting a large amount of photons. Even though billions of photons are cast during many iterations, it is not necessary to store the whole photonmap. After each photon tracing pass, the contribution of every photon is recorded by accumulating photons in an small area around each hit point. Then these photons are discarded. Therefore, the memory consumption is only dependent on the amount of photons that are cast during one iteration.

Adaptive Photon Tracing

Keller and Wald [KW00] introduce an importance driven algorithm affecting the deposition of photons by storing them only in areas of high visual importance. This reduces memory cost and computation time by maintaining the same image quality as in previous photon tracing approaches. The adaptive importance shooting technique, presented by Liu and Zheng [LZ14], calculates error values of neighbors in order to construct an adaptive cumulative distribution function, which is used to reflect photons, upon surface intersection, into directions of areas of interest.

Summary

A number of global-illumination approaches are based on radiosity; raytracing; or hybrid methods like instant radiosity. The objective of these methods is to render in real time or at least at interactive frame rates; under the constraint of image quality and speed. Therefore, approximations are used instead of physically accurate calculations to achieve solutions in real-time rendering. In recent years there has also been a lot of research towards real time photon mapping [MLM13] involving screen space optimizations or culling.

All of the presented state-of-the-art photon-mapping approaches have advantages and disadvantages. When using radiance-estimation methods, there is always a trade-off between bias and variance. Feature detection in photon relaxation is difficult, especially in sparse textures.Most enhanced adaptive photon tracing methods increase rendering time and memory consumption, but work well for small parts of illuminated regions [KWXM16]. Progressive photon mapping is currently the best step towards interactive frame rates or real-time solutions, but needs thousands of iterations to obtain an accurate result [KWXM16]. However, many progressive photon-tracing methods have reached interactive frame rates on the GPU.

3.2 State-of-the-Art Image Filtering

There has been a lot research in the field of image processing involving guidance images for edge-preserving smoothing algorithms that achieve visually promising results. Petschnigg et al.[PSA+04] present a variety of techniques to enhance images and synthesize new

images with improved quality from a pair of photographs, one taken with flash to capture details, and the other one taken without flash to accurately capture ambient illumination. They introduce a joint bilateral filter for ambient image denoising using the flash image as guidance image to reduce noies in the no-flash image. He et al. [HST13] use the concept of this joint bilateral filter to develop a guided image filtering technique that can use the input image itself as guidance image. Zhang et al. [ZSXJ14] developed a "rolling guidance filter" to smooth images by using a scale-aware image filter that also takes a guidance image into account. Their approach works iteratively. First, the input image is smoothed with a Gaussian filter and then serves as guidance image for a joint bilateral filter that is applied on the input image. From then on, the result of one iteration is used as guidance image in the next iteration.

Cho et al. [CLKL14] propose a structure preserving image decomposition operator based on the bilateral filter, introduced by Tomasi and Manduchi [TM98], incorporating "relative total variation" (RTV). The authors aim to remove texture, but preserve structure. In extension to the original bilateral-filtering method, a local patch-based analysis of texture features is performed in order to obtain a texture description image as input for the range filter kernel of the bilateral filter. This guidance image substitutes the color or intensity differences in Tomasi and Manduchi's [TM98] formula to construct a joint bilateral filter:

$$J_p = \frac{1}{k_p} \sum_{q \in \Omega_p} f(||q - p||) g(||G_q - G_p||) I_q$$

For each pixel, a patch shift operation is done to find a texture patch that contains the current pixel, is least likely to contain structure and is the best representation for the texture region containing the pixel. A certain parameter is chosen to specify the patch size. For each possible and distinct patch that contains the current pixel, a minimum and maximum intensity is calculated to compute the intensity range of this patch as the difference of maximum and minimum intensity. From these results, the patch with the smallest tonal range is chosen, because it has a small likelihood of containing structure edges. Once this patch is found, its average intensity is used as value in the guidance image at the position of the pixel, that is currently processed.

A very high tonal range in certain regions can lead to bad results. To target this problem, the authors suggest using modified relative total variation (mRTV). The algorithm first computes an average image using a box filter. In the next step the tonal range of each patch centered on a pixel is calculated.

Then patch shift is used (as described before) to find the patch with minimum tonal range. For this minimum tonal range patch, a gradient at each pixel covered by this patch, is calculated and the magnitude of the maximum gradient is selected. This maximum gradient magnitude is then normalized by the sum of all gradient magnitudes of this patch and scaled by the tonal range, to obtain the mRTV value, as shown in this formula.

$$mRTV(\Omega_q) = \Delta(\Omega_q)\frac{\max\limits_{r\in\Omega_q}|(\partial I)_r|}{\sum\limits_{r\in\Omega_q}|(\partial I)| + \epsilon}$$

Patch shift can then be used to find the patch with the minimum mRTV value. Because small mRTV values in very smooth regions may become sensitive to image noise, further improvements are done. The mRTV value of the patch centered on the current pixel is compared to the previously computed minimum mRTV value. If these values are similar, the pixel centered patch is preferred and its corresponding value from the average image is copied to the guidance image. Otherwise, if the minimum mRTV value is considerably smaller, the average value corresponding to this patch is used for the guidance image. To implement this improvement, Cho et al. [CLKL14] suggest blending both average values together, using the difference in mRTV values as weight.

$$G'_p = \alpha_p G_p + (1 - \alpha_p)B_p$$

where

$$\alpha_p = 2(\frac{1}{1 + exp(-\sigma_\alpha(mRTV(\Omega_p) - mRTV(\Omega_q)))} - 0.5)$$

Kniefacz and Kropatsch [KK15] introduced a similar approach to reduce noise or unimportant details from images by first smoothing the image with a blurring filter and then restoring edges with a guided edge-aware filter. They propose using a separable Gaussian range filter or a symmetric nearest neighbor filter as guided filter, to produce faster results in comparison to the rolling guidance filter [ZSXJ14].

Summary

The presented image-filtering approaches perform very well in noise reduction scenarios. The proposed smooth and restore method by Kniefacz and Kropatsch [KK15] seems to be the most promising in terms of image quality and speed. However, since all of the presented image-filtering techniques require more calculations than a simple bilateral filter, they also have a higher computation time.

3.3 Lighting Design with HILITE

This master thesis was created in context of “HILITE”, a light-planning software developed at VRVis. HILITE is modern software tool in this field and considered state of the art in lighting design.

Figure 3.2: Complex scene with many light sources, rendered by HILITE.

3.3.1 HILITE - a Light Planning Software

HILITE is "[...] a dynamic, interactive, realistic real-time lighting simulation for complex architectural environments" [fVRuVFGb]. The software allows light designers to interactively develop a lighting concept for architectural scenes by loading 3D models and placing and changing light sources in the scene while the illumination is being calculated and progressively refined. The manipulation of light sources causes the simulation to restart and yield a physically plausible result in a few seconds. This modifiability not only allows immediate walk-throughs, it also enables light designers to refine their concepts interactively, together with their customers.

As a light-planning software, HILITE has to deal with very complex scenes, which include many light sources, like shown in Figure 3.2, or highly reflective materials, as can be seen in Figure 3.3. In order to handle such complex scenarios and calculate correct lighting simulations, HILITE uses light maps.

3.3.2 Global Illumination with Light Maps in HILITE

In order to develop a many-light global-illumination solution, Luksch et al. [LTH+13] use light maps and virtual polygon lights for a fast light-map computation.

In the context of light planning, it is necessary to provide interactive solutions. To allow the user to move around in a 3D scene while the light contribution is being refined, the

Figure 3.3: Scene with highly reflective materials, rendered by HILITE.

simulation has to be view-independent. One way to create a view-independent lighting solution is to use light maps. Lighting information of surfaces in a scene is pre-calculated and stored in a texture, called "light map" for later use. The downside is that the generation of light maps for large scenes is usually slow and has to be performed again once a light source changes position or light distribution behavior. That is the reason why light maps are not feasible for frequently changing lighting setups. However, Luksch et al [LTH+13] managed to speed up the light-map computation by creating virtual polygon lights.

At first, rays from the primary light sources are traced along their light paths over several bounces to generate virtual point lights (VPLs) at each intersection point. By using a RANSAC-based method, planar surfaces are detected in the VPL set, resulting in connected planar subsets and some remaining VPLs that represent arbitrarily formed surfaces. To reduce the huge number of VPLs, they are clustered into a smaller number of higher-order virtual lights using a k-d tree. In 2D, the planar segments are just split at the median position. In 3D, a bounding box of the VPLs is computed and split at the median position along the dimension with greatest extent of VPLs. An improvement to this technique is to not just force a median split, but look for large gaps

in the VPL distribution and split there. Luksch et al. [LTH+13] optimize the splitting according to gap size and index of the VPL, relative to the median. After clusters with approximately equal numbers of VPLs are obtained, a 2D convex hull for each cluster is computed. To simplify these convex hulls, the authors use polyline vertex reduction. The simplified versions are then expanded and clipped against the k-d tree. In 3D, clusters are approximated by sphere lights.

Light Accumulation

To calculate all lighting information, shadow mapping, from the primary lights and all virtual lights, is performed. With each light bounce a new VPL is created. The simulation calculates the illumination from each light source and all VPLs consecutively. These results are accumulated in the light map via additive blending.

3.3.3 Glossy Materials

To extend the so far only diffuse illumination information, stored in light maps, Luksch et al. [LTM+14] use mipmapped environment maps for glossy material properties and introduce a regular sampling scheme that produces better results than importance sampling for a low number of samples. Their material model is based on measured material data (BRDFs) and supports reflection lobes for glossy materials. One environment map is created for each glossy scene object, and mipmaps serve as pre-filtered versions of the glossy reflection and can be sampled according to the BRDF. Because the resulting image pyramid does not depend on the BRDF, one environment map is sufficient to render an object with arbitrary materials, reducing the number of necessary updates after material changes.

HILITE's Material Model

Scene complexity or interactions limit most approaches that try to deal with glossy materials. Physically based BRDF models like described by Cook and Torrance [CT82] have a high evaluation cost. There has been a lot of research around empirical models, like done by Phong [Pho75], Ward [War92] and Schlick [Sch94]. Kurt et al. [KSKK10] combine anisotropy and Fresnel effects of Ward [War92] and Schlick [Sch94] to achieve physically plausible results, and their method was therefore chosen as base for HILITE's material model, which is explained in more detail in Section 5.2.

CHAPTER 4

Overview

This chapter provides a brief overview of the algorithm presented in this thesis. The algorithm's line of action is summarized and a list of tasks shows what requirements need to be fulfilled to reach the defined goals.

The purpose of this project was to create a global illumination algorithm to simulate the light distribution of luminaires. The application should allow interactive changes and should be able to display noise-free previews until the final result is simulated. To achieve this, we designed an algorithm that renders at interactive frame rates, produces

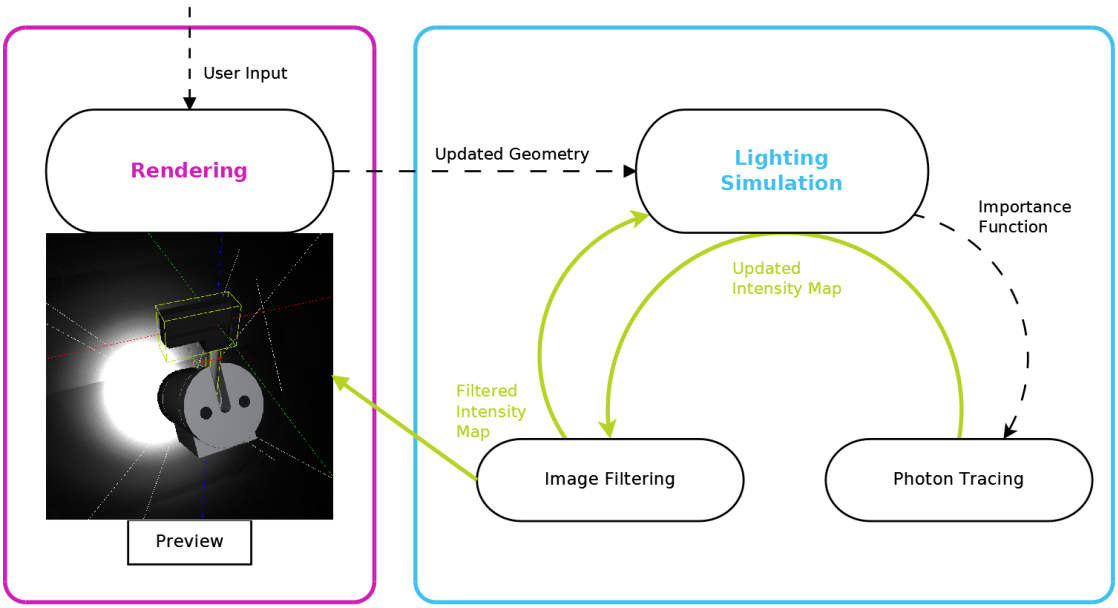


Figure 4.1: Overview of the main components of our application.

visually pleasing intermediate results, and converges to a physically accurate solution in the same time or faster than state-of-the-art global-illumination methods do.

The application, as shown in Figure 4.1, consists of two major parts: a rendering stage and a simulation stage, which are executed in parallel. During the simulation stage, the light distribution of a luminaire is simulated using photon tracing and filtered to achieve visually pleasing intermediate results. In the rendering stage, the results of the simulation stage are displayed and user modifications to the 3D model of a luminaire are handled.

4.1 Simulation Stage

The lighting simulation is an iterative process that calculates the light distribution of a light source. The simulation's main components are a photon-tracing process and a multi-resolution image-filtering method. Both are executed at each iteration. First, raytracing is performed to cast rays from the light-emitting part of the light source and reflect them on the light-source geometry. Photons are only stored at intersections with the surrounding cubemap, which results in a cube-map texture that represents the light distribution of the luminaire. More information on this photon-tracing method can be found in Chapter 5. The obtained cube map then functions as input for the image-filtering procedure. This method is described in detail in Chapter 6.

4.2 Rendering Stage

In the rendering stage, the light-source geometry is rendered as a 3D scene enclosed by a cub that shows the light distribution. Our algorithm works iteratively, and progressively refines the result. Between iterations, the user can modify the geometry and material properties of the luminaire by manipulating the 3D model. After each iteration of the simulation, a preview of the final result is rendered on the cube and shown in the render view.

To be able to display previews of the final result, our algorithm produces intermediate results between simulation iterations as described in the following section.

4.3 Production of Intermediate Results

The most important requirement for our raytracer is interactivity. Traditional workflows for luminaire design require the user to wait for the whole lighting simulation to finish (or at least wait for some time to get a smooth image) in order to see the results of manipulations of the 3D model representing the luminaire. To allow the user to manipulate the light-source geometry and see the effects at an early stage, fast previews are necessary, thus turning the luminaire design into an interactive process.

To produce intermediate results that serve as previews, our algorithm has to work iteratively. In each iteration, a number of random rays are cast into the scene to calculate

the light distribution. To get visually pleasing and fast previews, the following tasks have to be performed:

- **Guidance for Ray Casting Directions**

  To avoid artifacts, such as visible patterns caused by a pseudo random number generator, we integrate forced random sampling as described by Cornel [Cor14]. By reducing the number of rays that have to be cast, and concentrating them in specific regions, we try to speed up our photon-tracing process further. Homogeneous regions can be smoothed during our multi-resolution image-filtering procedure without losing information. Hence, we should not need a lot of photons in these regions. In regions that contain more information content, like shadow borders, we need more lighting information, and therefore more photons, to be able to preserve these details during image filtering. That's why the majority of photons should not be emitted into very homogeneous regions in terms of intensity. Thus, we explore integrating an importance function into our photon tracer to guide its ray casting scheme.

- **A Multi-Resolution Approach**

  Nichols and Wyman [NW09] introduced a novel approach to reduce the costs for reflective shadow map-based indirect illumination. They interactively splat indirect illumination using a multi-resolution buffer, where only splats near discontinuities are rendered in high resolution. The parts we can adapt for our method are the multi-resolution approach together with the interpolation technique the authors use to remove discretization artifacts when additively blending results.

  In order to reduce rendering time further, we need to decrease the number of necessary rays cast into the scene by adapting the upsampling technique described by Nichols and Wyman [NW09]. Instead of refining the buffer at some locations, multiple mipmap levels can be used to store each photon in each mipmap level. This results in different photon counts per texel on each level, as each texel in a mipmap level is the sum of the 4 corresponding texels of the level below.

  Our multi-resolution image-filtering approach is a scattered-data interpolation method based on the pull-push algorithm, like described by Kraus [Kra09]. During the pull phase, an image pyramid of a sparse texture is constructed. This image pyramid is used in the push phase to re-calculate texel values on all levels, from coarse to fine. Our approach uses the mentioned mipmaps as levels of the image pyramid (for our cube map that stores photon counts per texel) and re-calculates texels that contain less photons than a certain threshold and are therefore classified as "not stable". The texels are recalculated by upsampling values from the next higher mipmap level.

- **Upsampling using Intelligent and Fast Image Filters**

  As first step in the development of a suitable interpolation technique for upsampling, we explore bilinear interpolation. This is a standard interpolation technique that is

fast to calculate and easy to implement and to test, but also slightly blurs edges between regions of different intensity and can produce artifacts. To achieve a high visual quality, we have to preserve shadow borders as best as possible. We therefore introduced an edge-aware image filter that can be used to interpolate texels that do not satisfy our threshold from the next higher mipmap level. We chose bilateral filtering as base for the interpolation technique of our algorithm.

Our bilateral interpolation scheme uses a Gaussian distribution as described by Tomasi and Manduchi [TM98], which causes less artifacts than pure bilinear interpolation, while still consisting of fairly simple calculations and therefore performing very well in terms of rendering times. The bilateral filter is a combination of a domain filter, using a closeness function based on Euclidean distance, and a range filter, applying a photometric similarity function dependent on the intensity difference of texels.

CHAPTER 5

Interactive, Progressive Photon Tracing

The algorithm presented in this thesis mainly consist of two parts, a photon-tracing method and an image-filtering step. This chapter describes the first part of our developed algorithm, the interactive, progressive photon tracing as highlighted in Figure 5.1. The underlying concept of the lighting simulation is presented as well as the implemented material model. Furthermore, we explain in what respect our method is interactive

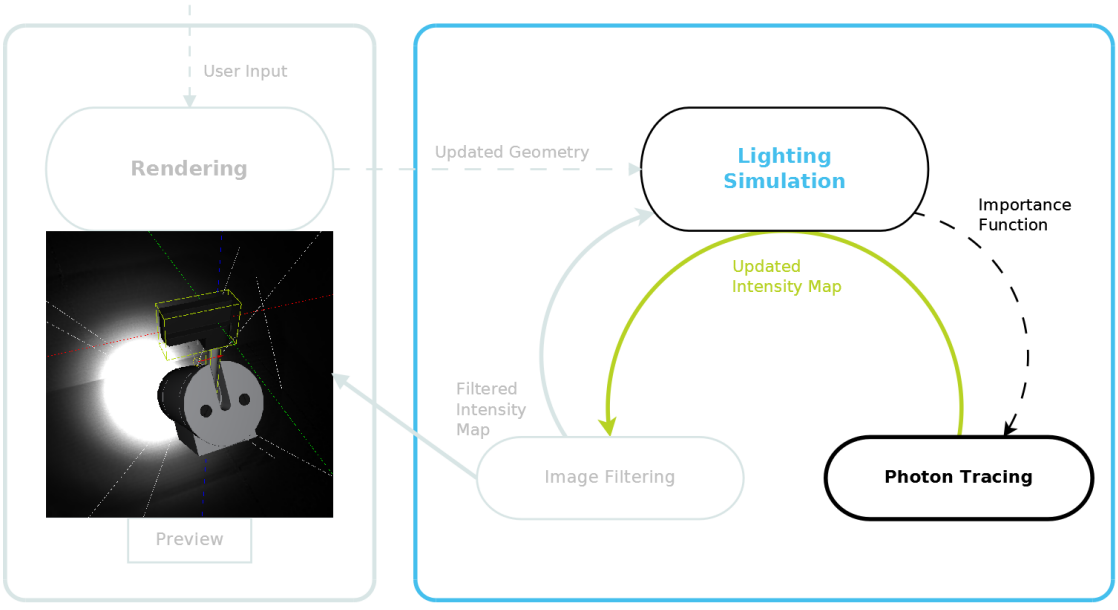


Figure 5.1: Overview of the main components of our application.

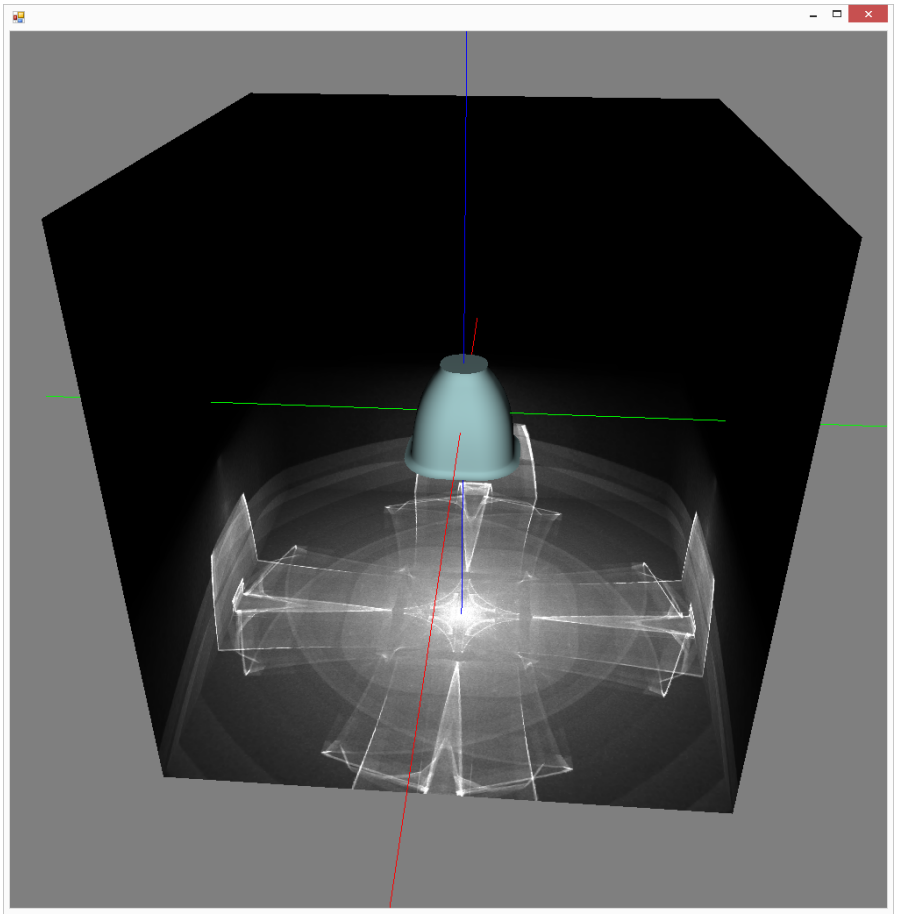


Figure 5.2: Light-source geometry enclosed by a cube.

and how to guide the photon tracer in order to achieve interactive frame rates. For the development of our interactive, progressive photon tracing approach, we built upon a photon tracer, developed at VRVis, that already provides functionalities to intersect rays with geometry, calculate intersection points and store photons in a specific data structure (which we call "photon store").

5.1 Photon Tracing

In our application, the 3D scene that is displayed in the render view consists of a 3D model of a luminaire with a light emitting part and a surrounding cube, as shown in Figure 5.2. Our algorithm is based on the concept of photon tracing. Rays are cast from the light-emitting part of the light-source geometry into the scene. Our algorithm uses an OpenCL buffer, called "photon storage", which stores all photons, that are produced during one iteration (with attributes like power, origin and direction), and a cube-map texture that only stores the total number of photons per texel (over all iterations). We

refer to this cube map, which surrounds the light-source geometry, as "intensity map". This intensity map is used throughout our whole algorithm. The photon storage is only necessary for incorporating adaptive photon-tracing methods like the importance sampling scheme, which we described in Section 5.4.1.

When a ray hits the cube, the photon count at the corresponding position in the intensity map is updated, and a photon is added to the photon storage. At each intersection with a surface, a reflection direction is calculated according to the surface material properties. For perfect mirrors, there is only one reflection direction. For other reflective or diffuse surfaces, we need to sample the reflection directions. For this sampling, we use an importance-sampled bidirectional reflectance distribution function (BRDF) as described in Section 5.2, based on Monte Carlo integration [Jen01]. Pure Monte Carlo ray tracing is unbiased, but very time consuming. The variance of Monte Carlo methods, depicted as noise in the renderings, is reduced by an increasing number of samples. Reducing the error to zero is not possible, because this would require casting an infinite number of rays, and following the reflection path of each ray (regardless of the number of reflections with the geometry) until it hits the surrounding cube. Therefore, we restrict our algorithm to a user defined amount of rays and maximum number of reflections per iteration. We then progressively refine the results by adding new photons to the photon storage (and increasing the photon count in the intensity map respectively) in each iteration.

In contrast to standard photon-tracing algorithms, we do not use a radiance density estimate or final-gathering step, since this yields only approximated solutions where fine details might get lost. Instead, our multi-resolution image-filtering technique, as described in Section 6, is used to smooth the image and preserve fine details. This way, our algorithm is able to show smooth previews in the render view while simulating in parallel, and converge to a physically plausible solution as will be shown in the Results and Discussion section.

Our photon tracer creates rays, processes them in groups, and intersects them with the scene. Upon intersection with the light-source geometry, new reflected rays are created. If a ray hits the surrounding cube, a photon is stored in the photon storage. Simultaneously, the intensity map, containing only the numbers of photons that hit a certain texel, is updated. This intensity map serves as representation of the light distribution of the luminaire for our multi-resolution image-filtering approach. We use a texture atlas to allow for efficient memory usage and access of the intensity map. A texture atlas is one large texture containing a number of sub-textures. (A detailed description of texture atlases can be found in NVIDIA's SDK White Paper [NVI04].) Our border handling scheme for the texture atlas, including textures with several mipmap levels, is described in section 7.3. Furthermore, we use shader programs built with "CoSMo" (Intent-based composition of shader modules), as presented by Haaser et al. [HSM+14], to add functionalities for processing material properties, and implemented single scattering, like described by Luksch et al. [LTM+14], for our photon tracer.

5.2 Material Model

In order to obtain new reflection directions for our photon tracer at each bounce, we implemented HILITE's material model [LTM+14], which is based on the anisotropic BRDF model developed by Kurt et al. [KSKK10]. This model allows to generate physically plausible results. To achieve this, the simulation has to fulfill energy conservation laws as well as the Helmholtz reciprocity, which basically means that incoming and outgoing directions can be exchanged.

For standard Monte Carlo methods, best results are achieved with sampling formulas that yield random directions that are distributed proportional to the cosine-weighted BRDFs. Kurt et al. [KSKK10] build upon the Beckmann distribution as a normalized microfacet distribution. Cook and Torrance [CT81] already developed an effective microfacet-based BRDF model. However, this model does not support anisotropic materials, nor importance sampling. Ward's [War92] BRDF model is not normalized and therefore not physically plausible. Kurt et al. [KSKK10] proposed a physically plausible BRDF model for anisotropic materials, which also works for isotropic materials and uses Ward's importance sampling formulas to incorporate importance sampling of the BRDF.

Kurt et al. [KSKK10] use the following physically plausible normalized microfacet distribution function:

$$D(h) = \frac{1}{\pi m_x m_y cos^4(\theta_h)} q(h)$$

$$q(h) = e^{-tan^2(\theta_h)\left(\frac{cos^2(\Phi_h)}{m_x^2} + \frac{sin^2(\Phi_h)}{m_y^2}\right)}$$

with roughness parameters m_x and m_y. For isotropic materials, these parameters are equal, which results in the standard Beckmann distribution function:

$$D(h) = \frac{1}{\pi m_x m_y cos^4(\theta_h)} e^{\frac{-tan^2(\theta_h)}{m^2}}$$

The implemented BRDF model consists of a diffuse and a specular term:

$$f(\omega_i, \omega_o) = \frac{k_d}{\pi} + \frac{k_s F(\omega_o \cdot h) D(h)}{4(\omega_o \cdot h)((\omega_i \cdot n)(\omega_o \cdot n))^\alpha}$$

The Lambertian term $\frac{k_d}{\pi}$ represents the diffuse illumination and the second term represents the specular reflection lobe with k_s as specular reflectivity and $F(\omega_o \cdot h)$ as Fresnel function, that can be approximated by Schlick's [Sch94] formula:

$$F(\omega_o \cdot h) = f_0 + (1 - f_0)(1 - (\omega_o \cdot h))^5$$

with f_0 being the reflection coefficient at normal incidence.

Kurt et al. [KSKK10] also support the use of multiple reflection lobes, enabling the simulation of very complex materials. Furthermore, the method matches measured BRDFs more accurately than other existing models and was therefore chosen for our implementation.

Scattering Photons According to the BRDF

For a given incident vector **i**, we need to calculate a corresponding outgoing reflection direction vector **o** that closely matches the BRDF.

We use two uniform random variables u and v and use the importance transformation presented by Walter [Wal05] to obtain an outgoing reflection direction for Kurt et al.'s BRDF . Since Ward [War92] originally omitted an arctangent in his calculations, we use the following formula presented by Walter:

$$\Theta_h = \arctan\left(\sqrt{\frac{-\log(u)}{\frac{cos^2(\Phi_h)}{m_x^2} + \frac{sin^2(\Phi_h)}{m_y^2}}}\right)$$

$$\Phi_h = \arctan\left(\frac{m_y}{m_x}\tan(2\pi v)\right)$$

and can use Θ_h and Φ_h to calculate a halfway vector **h**:

$$\mathbf{h} = [sin(\Theta_h)cos(\Phi_h), sin(\Theta_h)sin(\Phi_h), cos(\Theta_h)]$$

With this halfway vector we are now able to find a sampling direction **o** using the following reflection formula:

$$\mathbf{o} = 2(\mathbf{i} \cdot \mathbf{h})\mathbf{h} - \mathbf{i}$$

5.3 Interactive Changes

By designing our algorithm in an iterative manner, we enable the user to interactively make modifications to the scene, which take effect as soon as the next iteration of the light simulation starts. We integrated a set of editing functionality to translate, rotate or scale parts of the geometry as shown in Figure 5.3. Furthermore, we provide the user

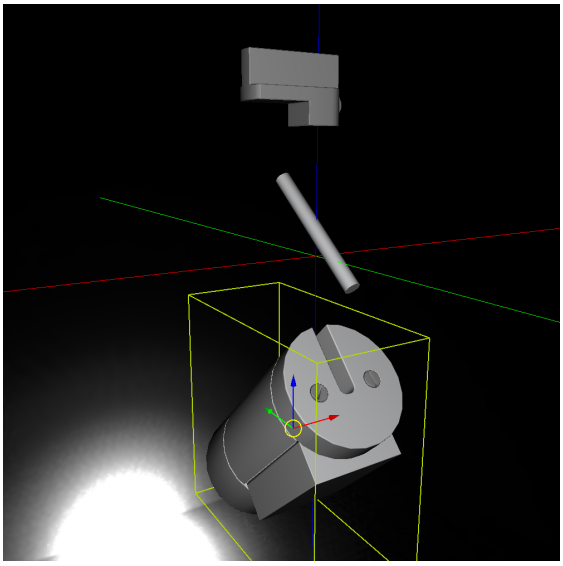


Figure 5.3: editing geometry

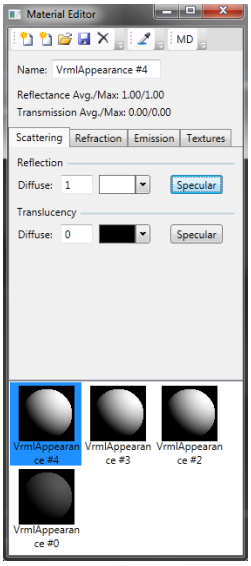


Figure 5.4: HILITE's Material Editor

with the possibility to directly specify which part of the 3D model is the light-emitting geometry, and assign pre-calculated or measured light-distribution curves to it.

Apart from geometrical alterations, it is also possible to assign materials and interactively change material properties like diffuse or specular reflection or translucency, refraction or perfect reflection, or provide parameters like the Fresnel term for our material model via HILITE's material editor. Materials with similar reflection properties as mirrors or glass, as well as diffuse materials, can be reproduced and are then processed by our photon tracer as described in Section 5.2.

The great benefit of allowing this editing functionality in parallel to the simulation is that it can speed up the light-source design workflow, because the light-source engineer does not have to wait for the final results of a lighting simulation anymore. Our application enables users to change parameters of the model after getting the first previews of the simulation. Furthermore, each modification of the geometry or the material properties instantly restarts the simulation to show the new results, which turns the light source design from an iterative into an interactive process.

5.4 Guided Photon Tracing

In order to increase illumination information around edges or fine details, we want to guide our photon tracer to cast photons into specifically interesting directions. Note that this affects only the emittance of photons and does not influence the BRDF, which is used to calculate reflectance directions upon surface intersections. First, we have to define in which directions to cast photons from the light emitter. The most accurate results can be achieved by loading and assigning a light-distribution curve to each illuminant (a

light-emitting device inside electric light fixtures, e.g., light bulb or LED). This light-distribution curve serves as emission profile to limit possible ray directions according to the light-distribution behavior of the illuminant. Pre-calculated distributions like an omni-directional light-distribution curve can be imported to approximate the light distribution of an illuminant. In the range of possible ray directions, defined by the emission profile, random directions have to be chosen. To avoid artifacts, such as visible patterns caused by a pseudo random number generator, we integrate forced random sampling as described by Cornel [Cor14] to generate our random ray directions for each iteration. By concentrating rays in specific regions, we should be able to reduce the number of rays that have to be cast, which would improve the performance of our application regarding rendering time. Homogeneous regions can be smoothed during our multi-resolution image-filtering procedure without losing information. Hence, we should not need a lot of photons in these regions. In regions that contain more information content, like shadow borders, we need more lighting information, and therefore more photons, to be able to preserve these details during image filtering. That's why the majority of photons should not be emitted into very homogeneous regions in terms of intensity. Thus, we enable our photon tracer to take an importance function as input, which can guide the photon tracer's ray-casting scheme.

5.4.1 Importance Function

The goal of using an importance function is to get more photons in regions that contain details like borders of intensity regions (homogeneous regions in terms of intensity). Photons that result from randomly choosing ray directions, as described in Section 5.4, are stored with similar energy (e.g.: with an energy of one). If we now want to cast rays not just according to the emission profile, but also guided by an importance function, we have to adapt the energy of the photons to this importance function. When increasing the amount of photons that are cast into a certain direction, the energy of each of these photons has to decrease.

As Jensen et al. [Jen01] describe, integrating a function $f(x)$ using Monte Carlo integration is done by multiplying the mean of the function (calculated as an average of uniformly distributed random numbers ξ_i in the interval) by the length of the interval, which results in an estimate I_m of the integral I:

$$I = \int_a^b f(x)dx$$

$$I_m = (b-a)\frac{1}{N}\sum_{i=1}^{N} f(\xi_i)$$

Importance sampling improves the quality of this estimate by concentrating samples in important parts of the function. For this purpose, the uniform random samples ψ_i are scaled by the probability density function (pdf) $p(x)$, and we obtain the following estimator for the integral:

$$I_{mi} = \frac{1}{N} \sum_{i=1}^{N} \frac{f(\psi_i)}{p(\psi_i)}$$

Hence, when casting a ray into a direction that has a certain importance, we set the energy of the resulting photon to a value inversely proportional to the importance of the chosen ray direction.

After the first iteration of casting rays and storing photons, the importance function is calculated. To direct more photons to regions where more lighting information is needed (e.g., borders of intensity regions), we need to identify ray directions, as seen from the light source, that resulted in photons stored near regions of interest, and assign a higher importance to these ray directions. Ray directions are calculated by constructing a vector from the light-source origin through a texel in the emission profile. Photons that are stored in the photon storage contain information about their photon energy and the position of the target texel in the emission profile that was used to calculate their ray-casting direction. This enables us to connect each photon in the photon storage to the ray direction that was used to cast it. That means even after a photon has been reflected multiple times, we know its corresponding original ray direction from the light source. So we can back-project photons from the photon storage to the emission profile.

In order to identify photons in regions of interest, we use the intensity map, containing the photon count of each texel, and the photon storage. To extract regions of interest, we can simply apply an edge-detection filter, like the discrete Laplacian operator, to the intensity-map image. The discrete Laplacian is an approximation of the sum of second derivatives. It is calculated by summing up the differences of the four nearest neighbors (N_1 to N_4) to a texel T

$$\sum_{i=1}^{4} (f(N_i) - f(T)) = f(N_1) + f(N_2) + f(N_3) + f(N_4) - 4f(T)$$

and can also be interpreted as filter kernel in 2D:

$$\mathbf{L} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Because we only cast a certain number of photons each iteration, not all texel values in the intensity map change in one iteration, and we do not need to calculate this sum of derivatives for texel values that did not change. Hence, we only take photons that have been added to the photon storage during the current iteration, look at their corresponding position in the intensity map and apply the Laplacian operator to these texels. This yields information about where regions of interest are located. However, we do not just want to cast photons in these directions, regardless of how many photons are already stored at these locations. If we have already stored a large amount of photons at the

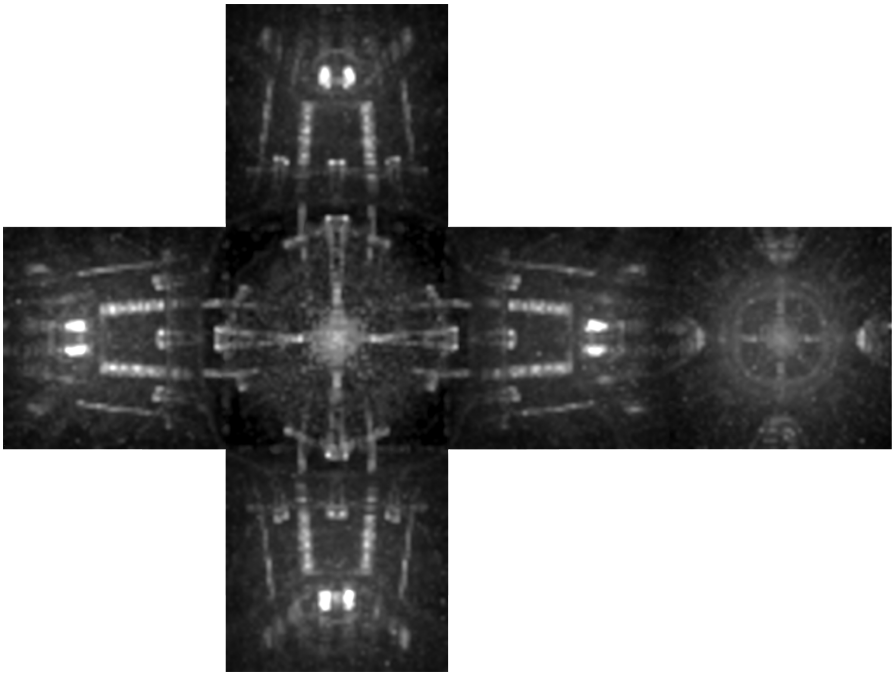


Figure 5.5: Example of an importance image corresponding to the 3D scene shown in Figure 5.2.

border of intensity region, so the areas in this region are smooth and the edges between the regions are well defined, we do not want to still direct a high amount of photons in this direction. That is why we calculate our importance $imp(T)$ as the value obtained from applying the Laplacian operator, relative to the value already stored at the exact same position in the intensity map:

$$imp(T) = \frac{|\mathbf{L}(f(T))|}{f(T)}$$

Finally, we add these calculated importance values to the emission profile at the positions belonging to the ray directions that resulted in the photons which we processed in this iteration. We update the importance, and thereby the emission profile, as described in this section, in each iteration. Furthermore, we calculate mipmaps of the updated emission profile and apply our multi-resolution image-filtering approach to it, in the same way as described for the intensity map in Section 6. This yields a smooth image, as shown in Figure 5.5, that can be used as importance function to guide ray-casting directions.

However, evaluations revealed that using the presented importance function increases variance in smooth image regions and is therefore not beneficial. Problems that arise from this importance sampling scheme will be elaborated in Section 8.

CHAPTER 6

Multi-Resolution Image Filtering

This chapter describes the second part of our algorithm, the multi-resolution image-filtering step indicated in Figure 6.1, and explains our design decisions for the algorithm. In Section 6.1, we give a high-level overview and Section 6.2 describes the workflow of our multi-resolution image filtering as shown in Figure 6.4. Furthermore, details of our upsampling procedure are discussed, and we investigate problems that arise from using bilinear interpolation during upsampling and how we improve results with bilateral interpolation.

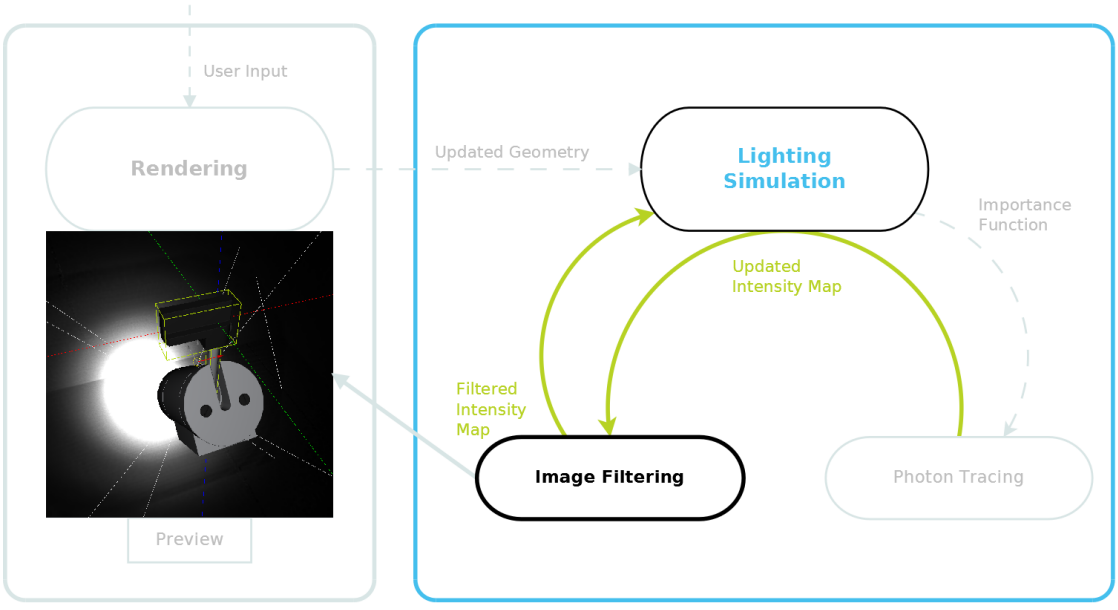


Figure 6.1: Overview of the main components of our application.

6.1 High-Level Overview

Our lighting simulation works iteratively and consists of two parts: a photon-tracing procedure and a multi-resolution image-filtering step. The simulation uses a 3D model of a luminaire and a cube surrounding the light-source geometry to calculate the light distribution of the luminaire. Our photon-tracing algorithm, presented in Section 5, casts rays from the light-emitting part of the geometry and reflects them on the geometry of the luminaire until the maximum number of allowed bounces per ray is reached or a ray hits the surrounding cube. Our algorithm uses a cube-map texture that stores the total number of photons per texel. We refer to this cube map as "intensity map". If a ray from the light emitter hits the surrounding cube, the photon count at the corresponding position in the intensity map is updated.The light distribution of a luminaire is represented by the photon count in the intensity map. A high photon count in an area means that this area has high illumination, whereas small photon counts indicate that a region is in shadow. During each iteration, additional photons are cast and photon counts in the intensity map are increased. The intensity map represents the light distribution of a luminaire. Each photons transports a percentage of the total emission power of the light source. In order to provide a visualization of the distribution represented by the intensity map, we apply a simple variant of linear tone mapping using the number of iterations as scaling factor. Since we only use uniform omni-directional emission profiles to approximate the light distribution of LEDs, all emitted photons have the same energy (unless we incorporate an importance-sampling scheme). When calculating reflection directions, upon surface intersection, according to the BRDF, Russian Roulette [Jen01]is used. This means that the photon tracer reflects a high amount of photons with the same energy into directions specified as important by the BRDF, instead of the same amount of photons (as in every other direction) with high energy. Therefore, we are only dealing with photons with equal power and are only interested in the number of photons per texel in the intensity map during our multi-resolution image-filtering approach.

Our goal is to obtain a high-resolution intensity map image that has not only sharp edges at borders of intensity regions, but also a low variance in homogeneous regions and therefore looks smooth. Regarding homogeneous regions, we can see that during the first few iterations of photon tracing, photons will only hit some texels, while other texels will be reached by none or fewer photons than their neighbors. It takes a certain overall amount of photons and rendering time to get an even photon distribution in image regions that are supposed to be smooth. To still get images that show previews of the light distribution rather than just random noise at an early stage, we want to smooth these regions. Texels that have a significantly lower photon count than their neighbors show up as noise in the image. To eliminate this noise, we need to replace these low photon counts by values that more accurately represent the illumination in this region. To distinguish between texels whose values should be replaced, and texels whose values should not be modified, we categorize them as "stable" or "not stable", depending on their photon count. Because the photon distribution depends on the light-source geometry and material properties, this categorization has to be done by manually choosing a threshold.

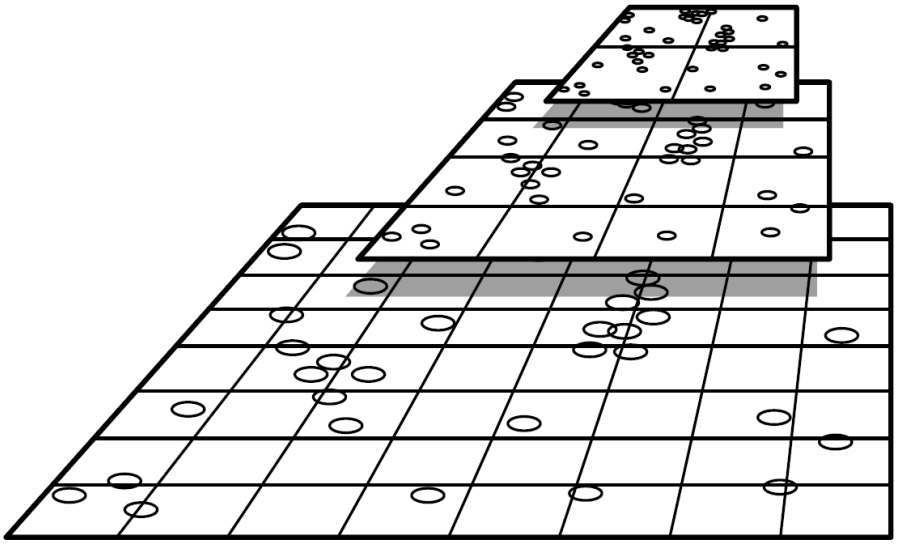


Figure 6.2: 2D pull-push. At higher resolutions the gaps, corresponding to regions with low values that need to be upsampled, are bigger. Reprinted from [GGSC96].

We implemented a pull-push algorithm for our multi-resolution image-filtering approach, similar to the pull-push algorithm developed by Gortler et al. [GGSC96] for their Lumigraph system. Different resolutions of the same texture are used, as shown in Figure 6.2. During the pull phase, values of the higher resolution textures are summed up to get approximations for lower resolutions. In the push phase, lower resolution values are used to upsample values on higher resolutions, and fill in regions at higher resolutions that do not have sufficiently high values.

One of the fundamental concepts of our multi-resolution image-filtering approach is an image pyramid. Our algorithm operates on different resolutions of the same image, so we need to calculate mipmaps to build up an image pyramid. The term “mip” originates from the Latin phrase “multum in parvo”, meaning “much in little”. A mipmap is an array of pre-calculated texture maps, subsequently reduced in resolution by the power of two. Each level of this image pyramid is constructed by blending four samples of the texture on the lower level together, creating an average, as shown in Figure 6.3. The largest texture is referred to as “level 0”, smaller textures are numbered sequentially [Len12].

In contrast to the standard mipmap creation process, in which texels are averaged, we just sum up texels to create multiple resolutions of the intensity map. We did this because of implementation details of the photon tracer we built upon. We compare the total sum of photons per texel to the user-defined global threshold and identify texels with low photon counts. Texels on high mipmap levels are more likely to have higher values than the threshold and will therefore not be altered.

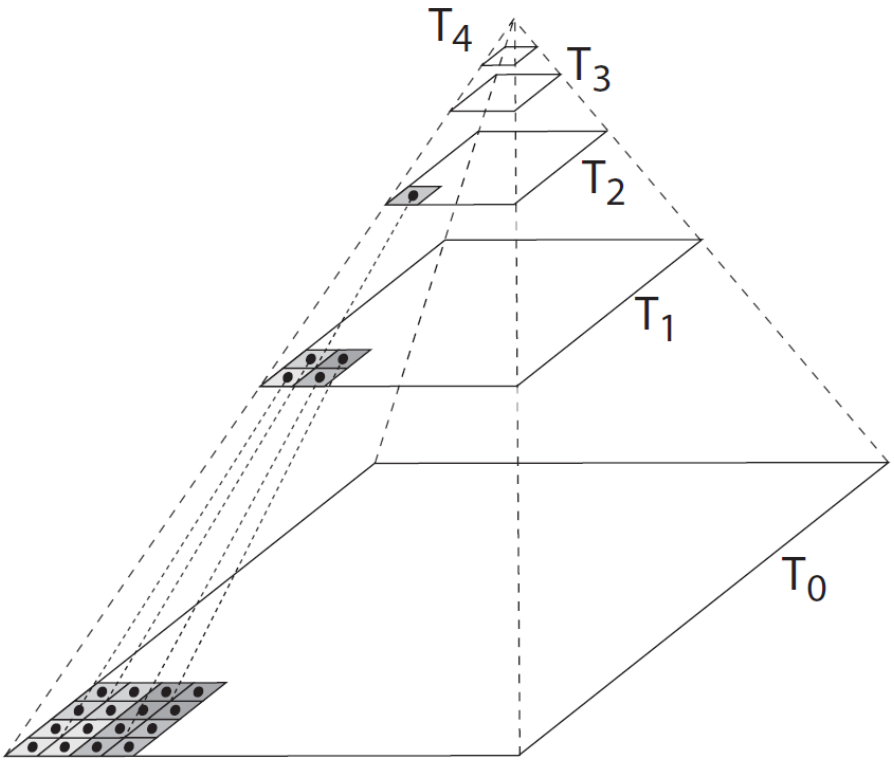


Figure 6.3: Mipmap hierarchy with texture T_0 being the original resolution. Reprinted from [Hec86].

On lower mipmap levels, more texels will have lower values than the threshold, and will hence be considered not stable. To replace the values of texels that are classified as not stable, we calculate a new value from the next higher mipmap level. We locate the position of the currently processed texel on the next higher mipmap level and look up the four texel values in proximity of this location. Then we interpolate between these four values to obtain a new value that we use to replace the photon count of the currently processed texel. Since each texel on one of our mipmap levels is the sum of four texels on a lower level, we have to divide this value by four before we can overwrite the original value of a texel by the interpolated value. This process of looking up texel values from a higher mipmap level and interpolating between them to obtain a replacement value is called "upsampling" and is explained in more detail in Section 6.4.

6.2 Workflow

Figure 6.4 shows the workflow of our multi-resolution image-filtering approach. Our algorithm starts its push phase by processing texels at a user-defined mipmap level and works its way downwards through the image pyramid until level zero (the highest resolution of the intensity map) is reached. For each texel, we look up the photon

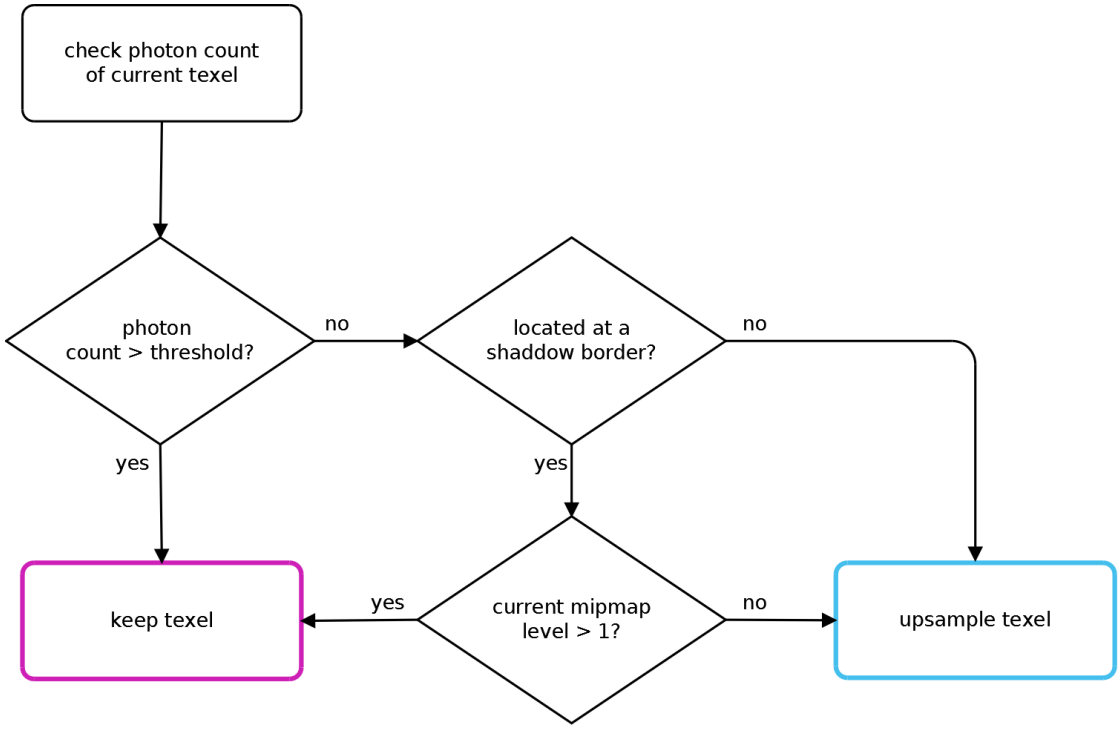


Figure 6.4: Multi-Resolution Image Filtering Workflow.

count in the intensity map. This value is compared to the threshold defined by the user. Depending on whether the photon count is higher or lower than the threshold, we either keep the texel or we investigate its neighborhood as explained in Section 6.4.2. If the photon count is lower than the threshold, and the texel is not directly located at a "shadow border" (a border of a high-intensity region), we upsample it as described in Section 6.4. In general, we want to upsample all texels that have a low photon count, but by excluding texels at shadow borders on higher mipmap levels from this upsampling step, we are able to reduce artifacts due to intensity leaks as described in Section 6.4.3. In the case that a texel is located at a shadow border, we do not upsample it if the the mipmap level currently processed is higher than level 1. Only for level 0 and level 1, we also upsample texels that have a lower photon count than the threshold and are located directly at a shadow border. We treat the first two mipmap levels differently to avoid pixelated shadow borders in the final image.

6.3 Basic Concept

The concept of our multi-resolution image-filtering approach is based on work by Nicolas and Wyman [NW09]. They introduce a novel approach to reduce the costs for reflective shadow map-based indirect illumination by interactively splatting indirect illumination using a multi-resolution buffer where only splats near discontinuities are rendered in

high resolution. For our application, the interesting part of this method is the multi-resolution approach together with the interpolation technique the authors use to avoid discretization artifacts. When splats are rendered into the multi-resolution buffer, a min-max mipmap is used to define areas of discontinuity. Instead of storing average values of texels of lower mipmap levels, min-max mipmaps store the minimum and maximum value of the corresponding texels. If the results of the min-max mipmap (e.g., differences of min and max values of the four corresponding texels on the next lower level) are greater than a certain threshold, a discontinuity is detected and the region is refined. To blend texel values coming from different levels without getting blocky artifacts or strange multi-resolution haloing Nicholas and Wyman use a unique upsampling scheme: They progress from highest to lowest level of the multi-resolution buffer, subsequently increasing the resolution. Texels that already contain some illumination information are bilinearly interpolated with their neighbors of the same resolution to obtain values for the corresponding texels on the next lower level. The output of this upsampling pass are texels of higher resolution on the next lower level of the multi-resolution buffer.

Our approach is slightly different from the original technique proposed by Nicholas and Wyman [NW09]. Instead of refining the buffer at some locations, we use multiple mipmap levels and store the number of photons that hit the area of a texel in each mipmap level. We therefore get different photon counts per texel on each level, since each texel in a mipmap level is the sum of the four corresponding texels of the level below. The basic idea of this approach is replace the value of each texel that does not have more photons than a certain threshold (by upsampling from the next higher mipmap level), and keep the ones that do. Instead of processing one level to output values for the next lower level, like described by Nicolas and Wyman [NW09], our algorithm processes a level and makes lookups in the next higher mipmap level (the lower resolution) if a texel needs to be upsampled. The user choses the number of mipmap level that should be taken into account and we start our multi-resolution image filtering at the second highest of these mipmap levels. If a texel does not contain enough photons, it is considered as “not stable” and upsampled by interpolating the values from its corresponding texels on the next higher mipmap level.

6.4 Upsampling

During multi-resolution image filtering, we identify texels that are not stable (due to their low photon count) on each mipmap level and execute our upsampling procedure for such texels. In that case we calculate its corresponding position on the next higher mipmap level, look up the texel values of the four texels in proximity of this position and interpolate between them.

6.4.1 Upsampling Texels with Low Photon Counts

Nicholas and Wyman suggest no upsampling of pixels with zero energy (in our case “shadow texels”) and excluding texels with zero energy from the interpolation. As we

usually don't have pure black shadows, but instead darker and lighter shadows, this total lack of upsampling preserves shadow borders, but produces noisy results in dark areas. We therefore also want to upsample texels in shadow regions and have to take texels in the same region on the next higher mipmap level into account for the interpolation.

However, simply upsampling a texel with a too low photon count, regardless of the intensity differences between texels in this area, yields problems when using a lot of mipmap levels together with a high threshold. This combination results in several mipmap levels where a large amount or all texels are upsampled, because none of them satisfies the threshold. As a consequence, sharp edges get blurred. For example, bilinearly interpolating over a shadow border not only blurs the border, but also creates a small intensity leak from a light pixel into the shadow area. This intensity leak increases with each upsampled mipmap level. The intuitive solution to this problem is to upsample all texels that have too few photons, except texels that are directly on a border to a region with higher intensity. In order to exclude shadow borders from the upsampling pass, we first need to detect these "shadow borders".

6.4.2 Shadow Borders

Following the approach of Nicolas and Wyman [NW09], we compare the number of photons a texel has stored to a user-defined global threshold. If the detected photon count is higher than the threshold, we consider it stable. Otherwise, we add it up with the photon count of one of its eight neighboring texels. If the sum of these photon counts is at least twice as much as the predefined threshold, the neighboring texel is considered stable. If this is given for three to six of the neighboring texels, we recognize the texel that has the lower photon count as belonging to a shadow border, and therefore do not interpolate it. Texels with more than six stable neighbors are not classified as belonging to a shadow border, but instead considered as isolated pixels that have not been reached by enough photons yet and have to be interpolated. Done iteratively on every mipmap level, this technique can preserve shadow borders of dark shadows, because we avoid energy bleeding from bright to dark areas. However, borders might look pixelated due to the fact that the texels next to shadow borders are not upsampled on any mipmap level. To avoid this pixelation, we only do these checks for higher mipmap levels and interpolate every texel that does not satisfy the threshold on level 1 and 0. There are still cases where intensity leaks can occur right next to a shadow border and grow with each mipmap level. Figures 6.5 to 6.7 illustrate the reason for the mentioned intensity leak that can occur while upsampling shadow texels, even if we do not upsample the shadow border texels.

6.4.3 Intensity Leaks Near Shadow Borders

The approach we discussed so far can produce artifacts in some regions, which we identified as "intensity leaks". We will now present an example to explain why these intensity leaks occur and discuss how we can reduce them by using bilateral interpolation.

0	0	0	200	200	200
0	0	0	200	200	200
0	0	0	200	200	200
0	0	0	200	200	200

Figure 6.5: Example of a shadow border with highlighted texel (in pink), which is going to be interpolated.

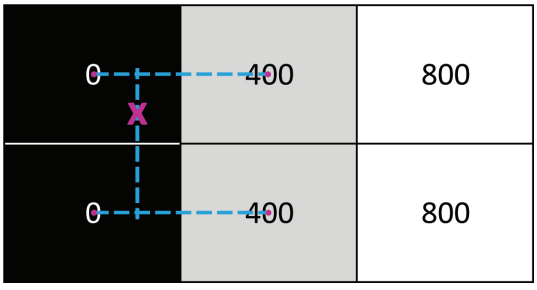
0	0	0	200	200	200
0	0	0	200	200	200
0	0	0	200	200	200
0	0	0	200	200	200

(a)

0	400	800
0	400	800

(b)

Figure 6.6: Four texels framed in blue (a) are summed up to one texel on the next higher mipmap level (b).



(a) bilinear interpolation

0	25	0	200	200	200
0	25	0	200	200	200
0	25	0	200	200	200
0	25	0	200	200	200

(b) intensity leak

Figure 6.7: A bilinear interpolation (a) near a shadow border can produce intensity leaks (b).

Assuming we are looking at a shadow border as displayed in Figure 6.5. The texels that are in shadow have a photon count of 0, the other illuminated texels next to the shadow border have a photon count of 200 each. In this example, we are currently processing the texel highlighted in pink. It does not satisfy our threshold and it is not directly located at a shadow border, so according to the method we developed so far, it should be upsampled. Figure 6.6b shows the next higher mipmap level at the exact same location. Each texel on the higher mipmap level is the sum of the corresponding four texels (framed in blue in Figure 6.6a) of the lower level. We now take the four texels of the higher mipmap level and do a bilinear interpolation as shown in Figure 6.7a and the following calculations.

Assuming that distances of texel centers are exactly at a value of 1, or normalized respectively, we calculate the horizontal distances α of our target point $P = (x, y)$ to the center of the left upper texel $H_1 = (x_1, y_1)$, as well as the vertical distance β to H_1. These values also correspond to the horizontal distance to the left lower texel center $H_2 = (x_2, y1)$ or the vertical distance to the center of the right upper texel $H_3 = (x1, y2)$ respectively.

$$\alpha = |y - y_1| \ , \ \beta = |x - x_1|$$

We can now use α as weight for a linear horizontal interpolation and obtain intermediate results

$$f(x1) = f(H_1)(1 - \alpha) + f(H_3)\alpha$$

and

$$f(x2) = f(H_2)(1 - \alpha) + f(H_4)\alpha.$$

Another linear interpolation between these intermediate results, using the vertical distance β as interpolation factor yields the following result:

$$f(x, y) = f(x_1)(1 - \beta) + f(x_2)\beta$$

This is standard bilinear interpolation. By inserting the values of the presented example into these equations, we get a value of 100. Because each texel on the higher mipmap level is a sum of four texels of the lower level, we then have to divide the calculated value by four. As final result we get a value of 25 for our currently processed texel. The resulting intensity leak for this whole region is shown in Figure 6.7b. The shadow texels next to the border are not upsampled in this example, because they are identified as a shadow border.

Even though shadows seem smooth, the example above shows that we might get artifacts at some shadow borders. In addition, at this point we do not take care of the bright

0	~0.1	0	200	200	200
0	~0.1	0	200	200	200
0	~0.1	0	200	200	200
0	~0.1	0	200	200	200

Figure 6.8: Result of bilateral interpolation.

texels near intensity discontinuities on a mipmap level where even some bright texels do not satisfy the threshold. These bright texels might still be darkened by upsampling them and bilinearly interpolating between bright texels and shadow texels of the next higher mipmap level. Our implemented bilateral interpolation, shown in Section 6.4.4, significantly reduces these problems. A bilaterally interpolated result of our presented example, yielding a much smaller intensity leak than bilinear interpolation, is shown in Figure 6.8.

6.4.4 Bilateral Interpolation

To preserve shadow borders as best as possible even if we upsample them, bilateral filtering using a Gaussian distribution as described by Tomasi and Manduchi [TM98] was integrated into our approach to interpolate texels. It consists of a combination of a domain filter, using a closeness function based on Euclidian distance in the spatial domain, and a range filter, applying a photometric similarity function dependent on the intensity difference of texels.

If we want to upsample texel $P = (x, y)$ and apply a bilateral interpolation, we first calculate the Euclidian distance of our target texel P to all 4 texel centers $H_1 = (x_1, y_1)$, $H_2 = (x_2, y1)$, $H_3 = (x1, y2)$ and $H_4 = (x2, y2)$ in the proximity of P's location on the next higher mipmap level. Then we use these distances d_i to construct a Gaussian filter in the spatial domain as *closeness function* c_i for each texel H_i:

$$c_i = e^{-\frac{1}{2}\left(\frac{d_i}{\sigma_d}\right)^2}$$

where e is the Euler Constant and σ_d the chosen standard deviation of the Gaussian filter. Analogous to c_i, we now want to calculate the *similarity functions* s_i, by using the intensity difference of P to a texel H_i on the higher mipmap level. This calculation requires knowledge of the intensity value stored at P. So this can be calculated when applying a bilateral filter to an image, but for our case the intensity $f(P)$ is what we want to interpolate and therefore unknown. To solve this issue, we use the bilinear interpolated intensity value $\tilde{f}(P)$ as approximate value for $f(P)$.

$$s_i = e^{-\frac{1}{2}\left(\frac{\delta(\tilde{f}(P), f(H_i))}{\sigma_r}\right)^2}$$

We can now combine the *closeness functions* and *similarity functions* for each texel H_i and use it as weighting factors for the final interpolation and normalize by the sum of these weights. We also divide by four to account for the lack of normalization in our mipmap pyramid.

$$f(P) = \frac{\sum_{i=1}^{4} f(H_i) c_i s_i}{4 \sum_{i=1}^{4} c_i s_i}$$

6.5 Problems and Considerations

The choice of a good threshold is crucial and difficult. The most significant effect of the chosen value for the threshold can be seen on the mipmap level on which shadow texels have less and all other texels have more photons than the chosen threshold. No filtering has to be performed for levels that contain only values above the threshold. However, on levels where all texels are below the threshold, the whole texture is re-computed. Hence, a too high threshold can lead to multiple levels that are re-computed by upsampling all texels from the next higher mipmap level. Even with an edge-aware image filter for the interpolation during the upsampling step, this would yield an over-blurred final result. Furthermore, it proved to be difficult to find a fitting value for the standard deviation σ of the domain filter and the range filter of the bilateral interpolation method. Whether or not the chosen values for σ_d and σ_r produce good results depends on the threshold and the intensity map image itself. It is possible to adjust the standard deviations to the threshold, but the distribution of photons (stored as photon counts in the intensity map) is what changes the quality of the results. In regions with a higher number of photons, but which is still below the threshold (resulting in upsampling the texels in this region), different values for σ produce good results, as compared to regions with a low photon count. During a series of tests, we chose $\sigma_d = 0.5$ and $\sigma_r = 0.2$, because this combination produced the best results for varying thresholds.

CHAPTER 7

Implementation Details

This chapter provides implementation-specific details of our prototype. Apart from programming languages and frameworks that we used, we also discuss the program architecture and the data flow of our application.

7.1 Languages and Frameworks

Our prototype is implemented in C#, using the Aardvark framework, developed at VRVis. Aardvark [fVRuVFGa] is a rendering framework for industrial 3D applications. Its library of software tools include mesh manipulation and rendering tools, visualization and interaction tools, parsers for 3D file-formats as well as caching and streaming methods. Shadow maps and light-map pre-computation are also integrated in the framework. Additionally, Aardvark's vast math library is tailored to computer graphics applications.

In order to calculate our simulation while at the same time rendering a 3D view of a luminaire, and allowing interactive modifications, we make use of C#'s task parallel library. For parallel calculations on the GPU, we chose the Open Computing Language (OpenCL) as framework for our shader programs. An OpenCL integration for C#, implemented in Aardvark, provided utility functions for tasks like setting kernel arguments, enqueuing, or calling an OpenCL kernel.

Furthermore, we implemented the material-model functionality for our application in CoSMo [HSM+14] (intent-based composition of shader modules), which allowed us to use C# syntax to write shader code, and combine multiple shader programs that are executed on the GPU at runtime.

7.2 Program Architecture

The most important components of our implemented prototype are listed below:

- Rendering Application (operates the 3D render view and processes user input)
- Photon Simulator (coordinates the whole simulation and the data flow between the photon tracer and the upsampler)
- Upsampler (executes the mulit-resolution image filtering)
- Photon Tracer (casts rays and processes photon hits)
- GUI (Graphic User Interface; allows user input to modify simulation parameters)

7.2.1 The Rendering Application

The rendering application is the core of our implemented prototype and responsible for creating the main render task, scene and light-source geometry, the GUI and the photon simulator. It initiates the rendering of the GUI and the scene graph in the render view, and starts the photon simulator's simulation in parallel, using C#'s task parallel library.

7.2.2 The Photon Simulator

The photon simulator first creates the OpenCL context and handles creation and disposal of OpenCl buffers as well as the usage of the underlying photon tracer. The initial amount of photons, that are cast from the light emitter, equals half the resolution of the targeted cube map, with a user defined maximum of bounces for each photon. However, it is not possible to predict the absolute number of rays, that will be produced in one iteration, due to the usage of a dither matrix for forced random sampling. Dithering and forced random sampling , which is explained in great detail by Cornel[Cor14], create randomly distributed as well as evenly spread rays. All rays are created by calculating ray directions from the light emitter through a pixel in the emission profile. These rays are cast and intersected with the light-source geometry. The resulting photons are stored in the photon store and the intensity map (an OpenCL buffer, that contains the photon counts per texel) is updated accordingly. The photon simulator then passes the intensity map to the upsampler, for further processing.

After applying multi-resolution image filtering on the intensity map, an (optional) importance function can be updated for the next iteration of creating and processing rays, storing photons and image-filtering. At the end of each iteration, the photon simulator provides the intermediate result to the rendering application, so the 3D render view can be updated with the new image.

7.2.3 The Upsampler

The upsampler handles the whole multi-resolution image-filtering procedure, which is described in detail in Section 6. It has a dedicated OpenCL shader program and handles kernel calls, as well as refreshing the shared texture between OpenCL and SlimDX, that is needed by the rendering application for immediate displaying of intermediate results. We implemented an iterative method, which means that the upsampler is activated after each time the photon tracer has cast a new wave of photons and updated the texels in the intensity map, that correspond to the hit positions.

The upsampler starts by compiling the necessary OpenCL kernels and calling the first kernel that performs the actual image-filtering. This kernel is called for each mipmap level individually. It takes the intensity map and a second OpenCL buffer, containing a copy of the intensity map (along with a few parameters) as input. The image-filtering is performed on the second buffer. With only read access to the original intensity map, we make sure that its values are not altered. This is important, because the intensity map is progressively refined by adding more photons in each iteration. If the intensity map would be filtered in each iteration, errors that are caused by a very coarse interpolation of sparse values would remain in the final image.

Texels are filtered in parallel, but as the mipmap levels are processed sequentially from the highest to the lowest level, we do not need another separate output buffer. The higher mipmap levels serve as input information and we can store the results of the current kernel directly in the OpenCL buffer at the location reserved for the currently processed mipmap level. Lookups of neighboring texel values (e.g. to identify if the currently processed texel is located at a shadow border) are done in the original intensity map. After all mipmap levels have undergone the image-filtering procedure another OpenCL kernel extracts the final cube map from the image-filtered buffer and stores it in highest resolution and without mipmaps in another buffer.

The third and final step for the upsampler is to copy the extracted result to the previously mentioned shared texture for the 3D render view. Depending on the user input, it will copy either the image-filtered result or the pure photon count (from the intensity map) to the shared texture for display. This allows the user to switch the view between pure photon counts and image-filtered intensity map. During this process it also adjusts the brightness of the result to account for the increasing number of photons. Note that our current implementation does not correct the result by applying tone mapping to the image.

7.2.4 The Ray Tracer

One of the most important elements of our application is an OpenCL photon tracer that provides functionality for creating and intersecting rays as well as processing hits and storing photons in mipmaps. The photon tracer is created once and updated if needed, i.e. if the geometry in the scene is changed. For each light emitter in the scene, the photon tracer is activated separately. It operates in 3 stages as shown in Figure 7.1:

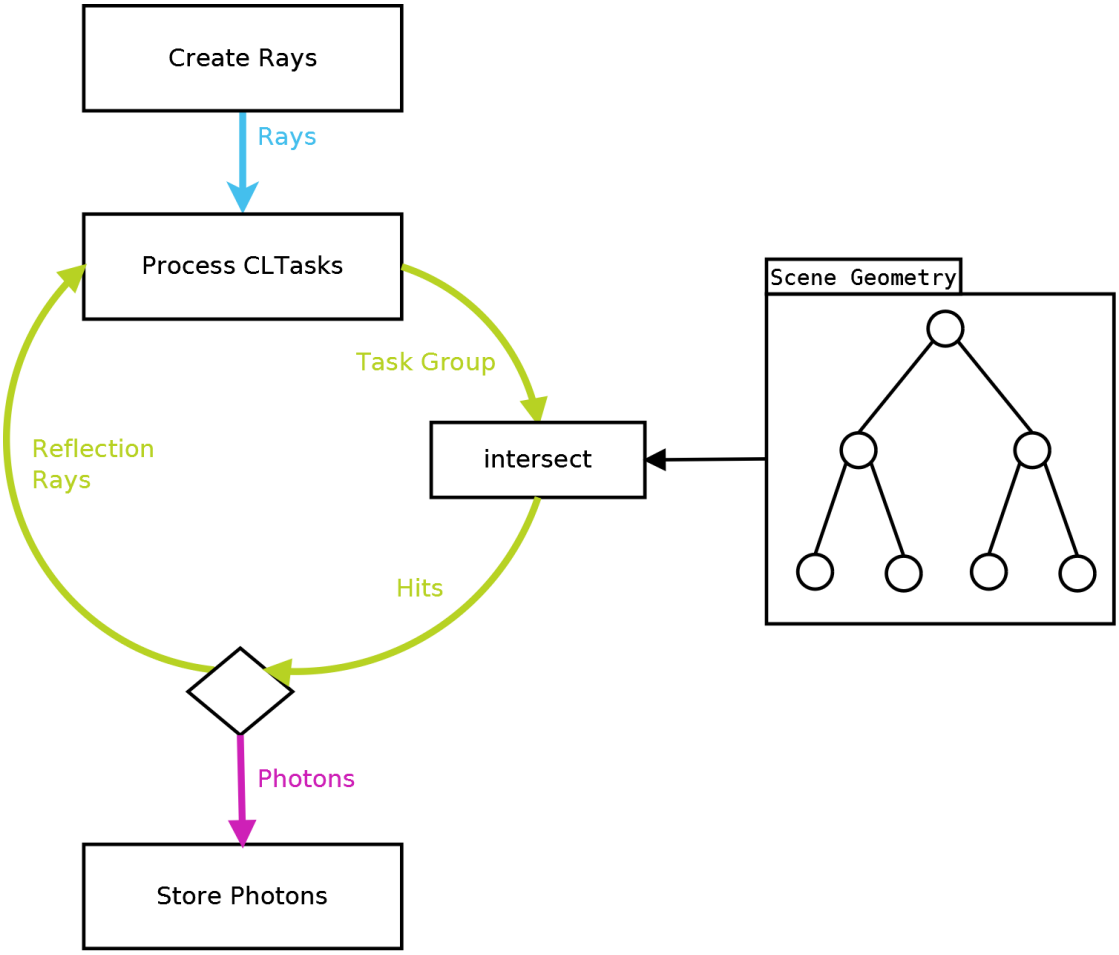


Figure 7.1: Photon Tracer Workflow

Create Rays

First of all, the photon tracer allocates memory for the number of rays that should be cast at each iteration. Because it is not possible to predict the absolute amount of rays that will be created by using forced random sampling with a dither matrix, one third more than necessary is allocated to make sure that the photon tracer will not run out of memory. A lookup in the dither matrix provides the "dither threshold" for each target texel as seen from the light source. This dither threshold in combination with the emission profile of the light emitter (which might not be uniform in each direction), and an optional importance function, determines which texels are valid targets for creating rays in these directions. The indices of these valid target texels are then used to create so called "ClTasks", that represent rays with origin, direction, and weight, and are stored in the "task buffer".

Process ClTasks

Rays are processed in small packages on the GPU, by taking groups of rays from a sub buffer of the task buffer. Rays are cast and intersected with a k-d tree, that holds the light-source geometry. The resulting hits are then again processed in groups by an OpenCL kernel. If a ray hits the cube, that surrounds the light-source geometry, a photon is stored in the photon store and the intensity map is updated. At each hit on the geometry, a reflection ray is calculated according to the material model (see Section 5.2), specified by the CoSMo [HSM+14] shader program. These reflection rays are the new CLTasks, which will be intersected with the geometry again. A maximum number of bounces determines how often a ray can be reflected. This processing of ClTasks is done recursively.

Store Photons

Finally, all new photons that have been produced during one iteration of ray tracing, are stored in a global photon store and the intensity map is updated. Mipmaps of the intensity map are created by simply adding up the corresponding photon counts of a texel without normalizing texel values.

7.2.5 The Graphic User Interface

The graphical user interface (GUI) allows the user to adjust input parameters and therefore influence the simulation and rendering of our application. Most of these parameters are OpenCL-kernel parameters. To avoid inconsistencies, the changes do not take effect immediately, but after the current simulation iteration has finished. The next iteration is then rendered with the new parameters. A progress bar shows the advance of the simulation. It is possible to pause the rendering (after the current iteration has finished), change parameters, and then resume or restart the simulation. Changing the cube-map face size prohibits resuming and forces the simulation to restart. This is due to the fact that new OpenCL buffers, with the now updated size, have to be created and passed to the shader program. Furthermore, we cannot take the current intensity map as input for the next iteration if it has a different resolution than the new output buffer.

7.3 Cube Maps and Border Handling

Our whole image-filtering method operates on 2D textures. However, we need to filter a cube map since that is the data structure we use for our intensity map, to represent the light distribution of a luminaire. Figure 7.2 shows the implemented cube-map layout. The faces are stored one after another in our OpenCL buffer as can be seen in Figure 7.3.

During the described image-filtering process, we need to check at each texel if it is at a shadow border or not. Therefore, we also need to access all 8 neighbor texels of the current texel. If our current texel is located at the border of a cub-map face, a border handling method is required to access the correct neighbors. Furthermore, if we need

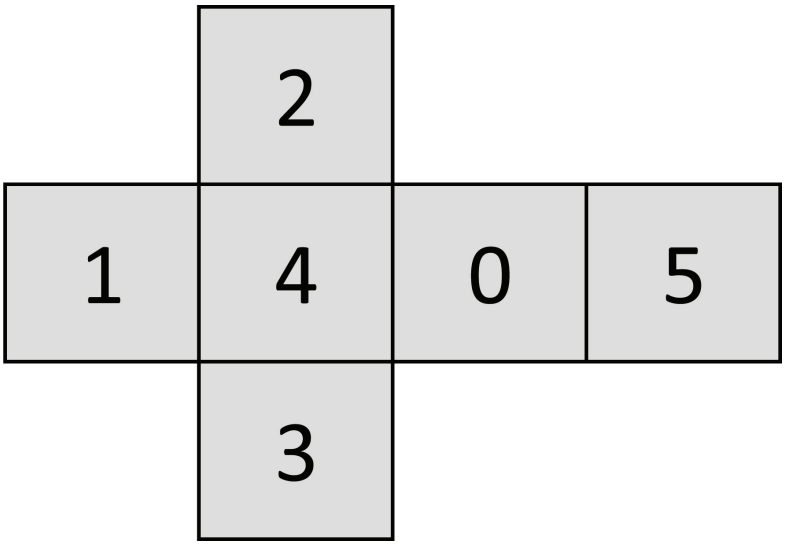


Figure 7.2: cube-map layout

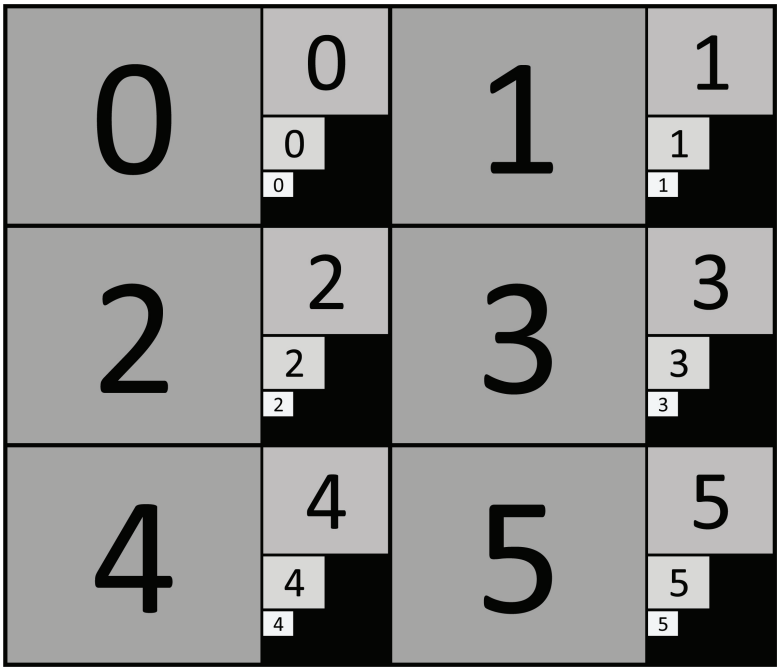


Figure 7.3: Cubemap faces stored in the texture atlas (in an OpenCL buffer).

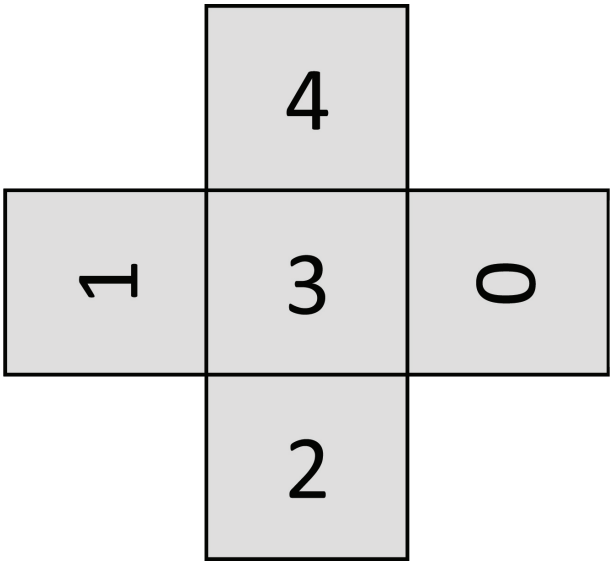


Figure 7.4: Neighbor relations of face number 3.

to upsample a texel next to a cube-map face border, we also need to access the correct neighbor texels on the next higher mipmap level. Only processing one cube map face at a time would lead to visual artifacts at cube map-face borders or darkened corners.

If we look at face number 3 for example, we can see in Figure 7.4 the orientation of the neighboring faces according to the cube-map layout. In this example, the above neighbor of the top left texel in face 3, corresponds to the bottom left texel in face 4. This means we have to do the texture lookup for the bottom left pixel in face 4 at the correct location in the buffer. Additionally, each face also contains all generated mipmap levels of that face, like shown in Figure 7.3. This leads to complex calculations in order to get the correct coordinates for neighbor texels at the borders, because we have to do the texture lookup in the correct face and on a specific mipmap level. So not only the offset and orientation of the face itself have to be considered, we also have to take into account the offset of the specific mipmap level, we are currently processing, relative to its face origin. To calculate the correct coordinates at borders, we use constant arrays (in the shader program), which define the neighbors for every face and specify the transformation that is necessary to get the right texel, once we have calculated the correct face, in which we have to do the lookup.

CHAPTER 8

Results and Discussion

In this chapter we will evaluate our new method for interactive luminaire simulation. The intensity maps that result from our simulation are evaluated in terms of image quality and performance (the time it takes to generate them). First, our test scene is described. Then we explain the calculation of our reference image. Different error metrics are used to compare our results to this reference image. We discuss the performance and limitations of our algorithm and our results with respect to state-of-the-art global-illumination and image-filtering algorithms.

8.1 Evaluation

As quality measures for the results, produced with our approach, we calculate the Peak Signal-to-Noise Ratio (PSNR), Signal-to-Noise Ratio (SNR) and the Root Mean Squared Error (RMSE) between our results and the reference image.

The intensity maps, that are compared to each other, in our evaluation, store the total amount of photons per texel instead of the integral over all photons. So we have to make the following adjustment to be able to compare them. Because each iteration of simulation increases the total number of photons in the image, we need to scale two different output images, so their values are in the same range, before we can calculate the PSNR, SNR or RMSE between them. Due to the randomly chosen ray directions, multiple photons can hit the same spot, and the brightest pixels in the images might be outliers. Therefore, maximum values are not suitable as scaling factors. Instead, we use the mean of an image to scale it before we calculate any error values.

8.1.1 Test Scene

We chose a bell-shaped luminaire, shown in Figure 8.1, and modified its material properties to make it reflective and translucent at the same time (these are material properties that

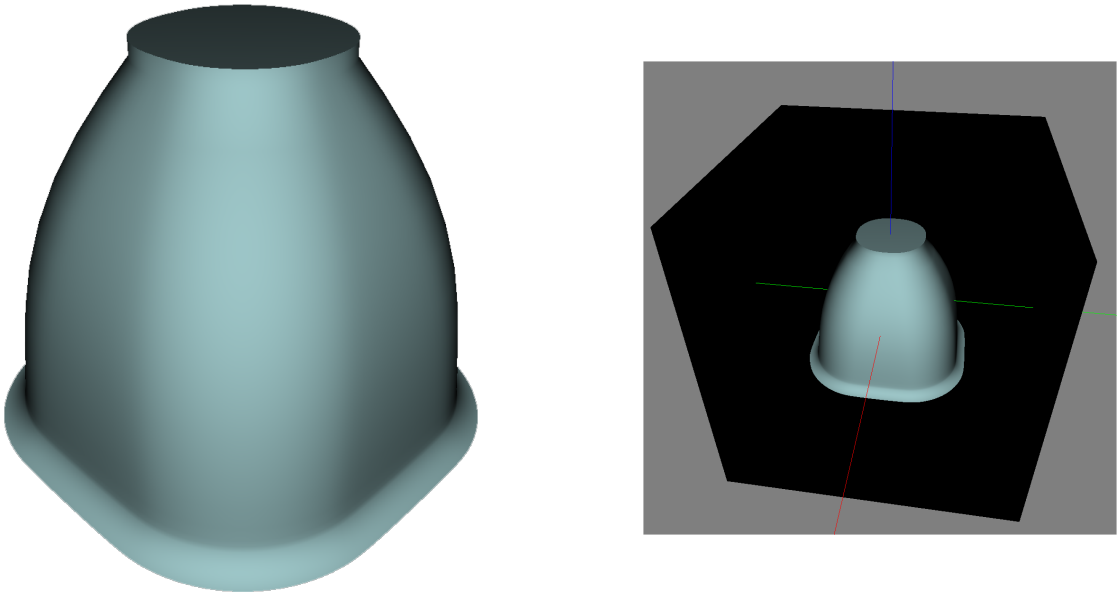


Figure 8.1: 3D model of the luminaire in our test scene.

you might expect from opal glass or translucent glass). That way, photons are widely spread along the surrounding cube, thus creating a worst case for any photon-tracing algorithm. We assigned an omni-directional light-distribution curve to the luminant (the light-emitting part of the luminaire) to get a uniform light emission profile. This means our photon tracer casts rays from the luminant outwards into every direction with the same probability. According to our chosen material properties, 60 percent of photons that hit the light-source geometry are transmitted, another 30 percent are diffusely reflected and the last 10 percent are perfectly reflected. Our luminaire is not perfectly round inside, and its inner structure shows as reflections on the bottom face of the cube in our test scene.

8.1.2 Reference Image

To evaluate our results, we first generate a reference image by pure random photon tracing. Rays are cast from the light-emitting part of the light geometry and reflected on the geometry. The resulting photons are stored in the photon storage, and photon counts in the intensity map are increased at positions corresponding to the photon hits on the the cube, that surrounds the 3D model of the luminaire. We only use the light-distribution curve assigned to the light emitter as emission profile without any importance function. No image filtering is performed. Because of limited resources on the GPU, a certain number of photons have to be cast for several iterations to reach a sufficient total number of photons that yields a smooth image. By pure random photon tracing, we can store about 13 172 300 photons in our photon storage (and increase photons counts in the intensity map respectively) in each iteration.

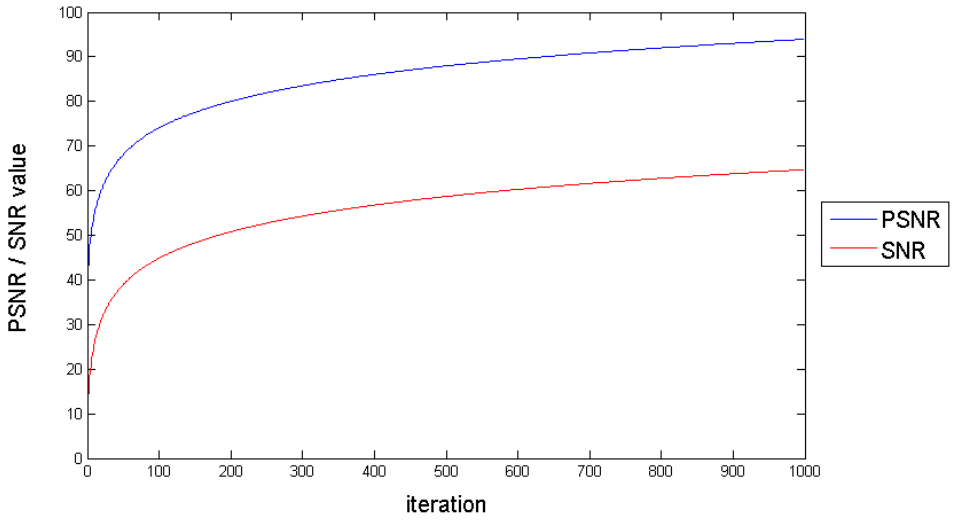


Figure 8.2: This plot shows the PSNR and SNR values of two consecutive iterations of pure random photon tracing in order to find a suitable image as reference image for further evaluations.

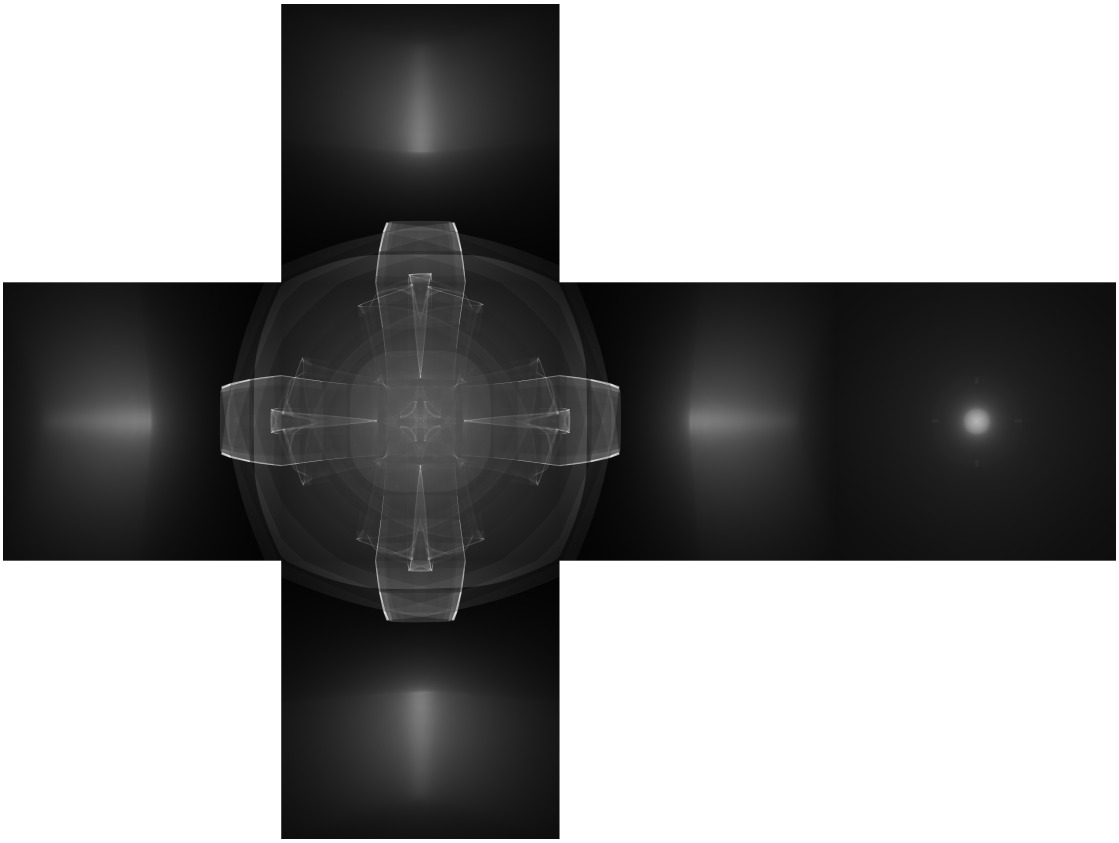


Figure 8.3: Reference image with cube-map layout.

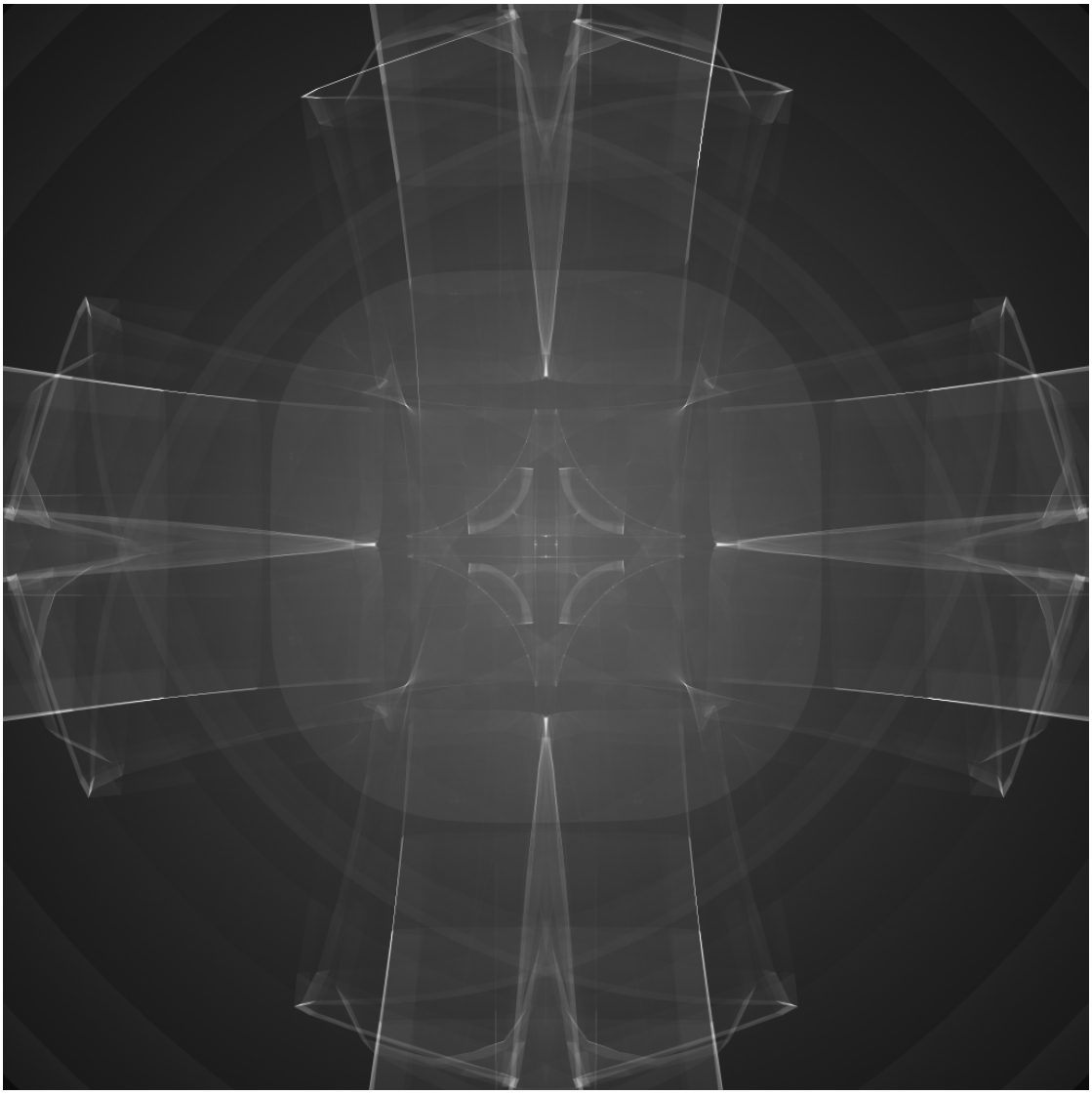


Figure 8.4: Each face of the reference image has a resolution of 1024^2 pixels. This figure shows face number 5.

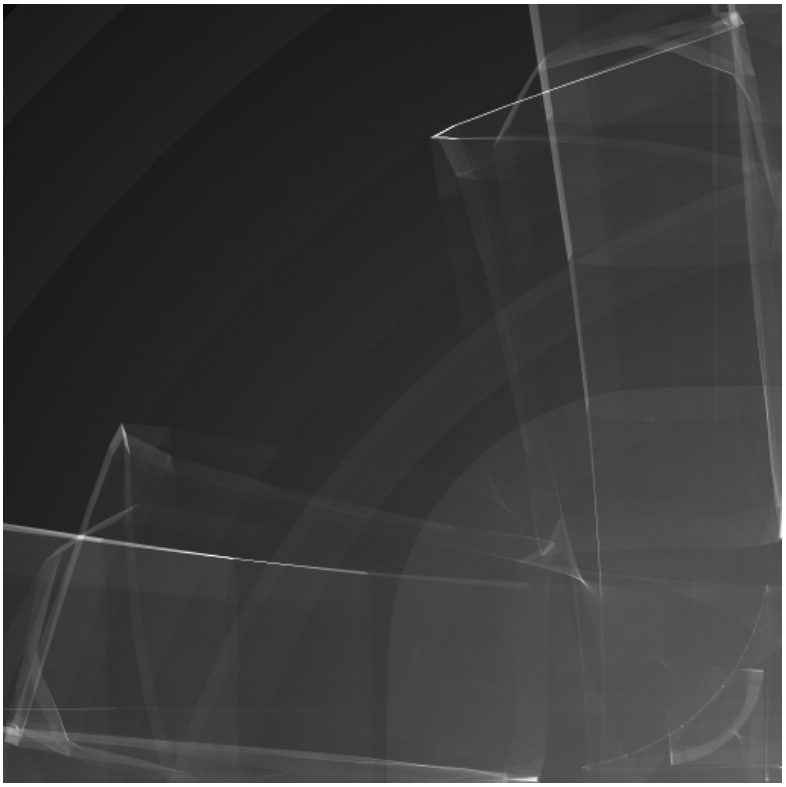


Figure 8.5: closeup of face 5

Figure 8.2 shows the PSNR and the SNR between two consecutive iterations for iteration 0 to 1000. The PSNR is increasing for each iteration, even after any increase in visual quality can be detected by the human eye. As reference image regarding image quality, we choose iteration result number 5000, with almost 66 billion (65 897 624 872) photons, obtained after about 12.5 hours of rendering time. The whole image has a resolution of 2048×3072, with 1024^2 pixels for each cube map face. The reference image with the implemented cube-map layout can be seen in Figure 8.3, and a closeup of face number 5 is displayed in Figure 8.5.

8.1.3 Error Evaluation of Intermediate Results

First we calculate the PSNR, SNR and RMSE between the reference image and our 32-bit-data values, stored in the intensity maps. These error metrics are calculated for intensity maps of subsequent iterations (with increasing photon count). The results in Figure 8.6 show a RMSE that vastly decreases during the first few iterations, until a certain amount of photons is reached, and does not change significantly afterwards. Our calculations also yield a RMSE of 0.0138 between our reference image and a second reference image calculated in the same time and with the approximate same amount of photons (except for slight differences due to fluctuations of the amount of photons

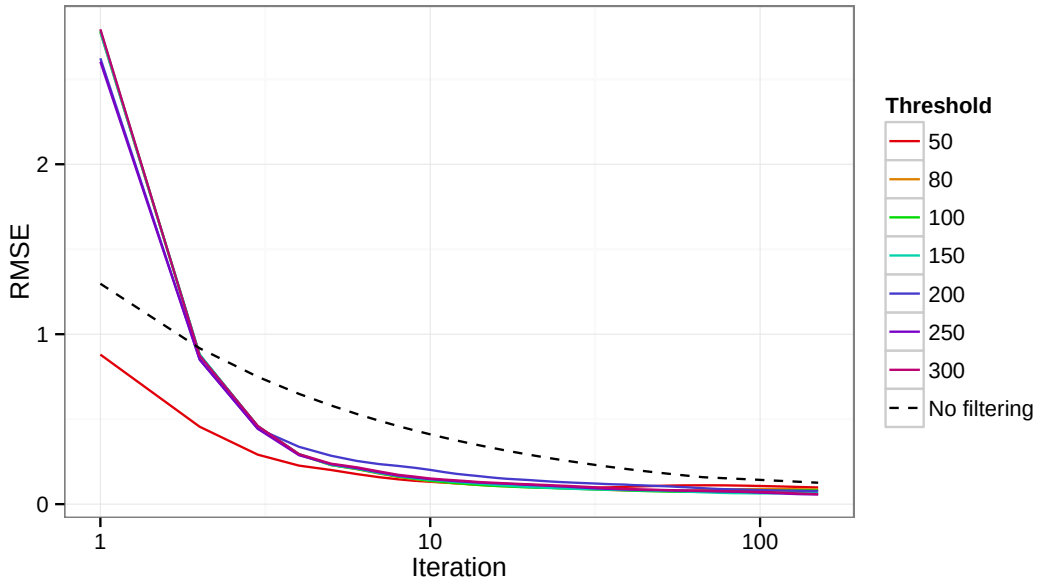


Figure 8.6: This plot shows RMSE values with respect to the reference image. The RMSE between the reference image and our image-filtered intensity maps, using thresholds from 50 to 300 is compared to the RMSE between the reference image and photon images without multi-resolution image filtering. Rendering time: ∼3.6 seconds per iteration for the image-filtered results and ∼3.3 seconds for images without filtering.

per iteration, caused by the random number generator, that is used to calculate ray directions for photons).

Figure 8.7 shows PSNR values between our image-filtered results and the reference image. Depending on the chosen threshold, results differ over time. During the first few iterations, a low threshold (e.g., 50) is more beneficial, while higher thresholds (e.g. 150-250) yield better results after more iterations. It can also be seen that our image-filtered results have significantly better PSNR values than a pure photon image without applying an image-filtering method to it.

Figure 8.8 and 8.6 show that our approach also achieves better SNR values and a lower RMSE than images without filtering. We obtained each image-filtered result in approximately 3.5 to 3.7 seconds, of which about 0.3 seconds were consumed by the multi-resolution image filtering and the remaining time was used by our photon-tracing method. Since our approach is integrated into an interactive tool, the first few iterations are the most interesting. Depending on user preferences, we can get even faster results at the cost of image quality by lowering the number of rays cast from the light emitter into the scene per iteration.

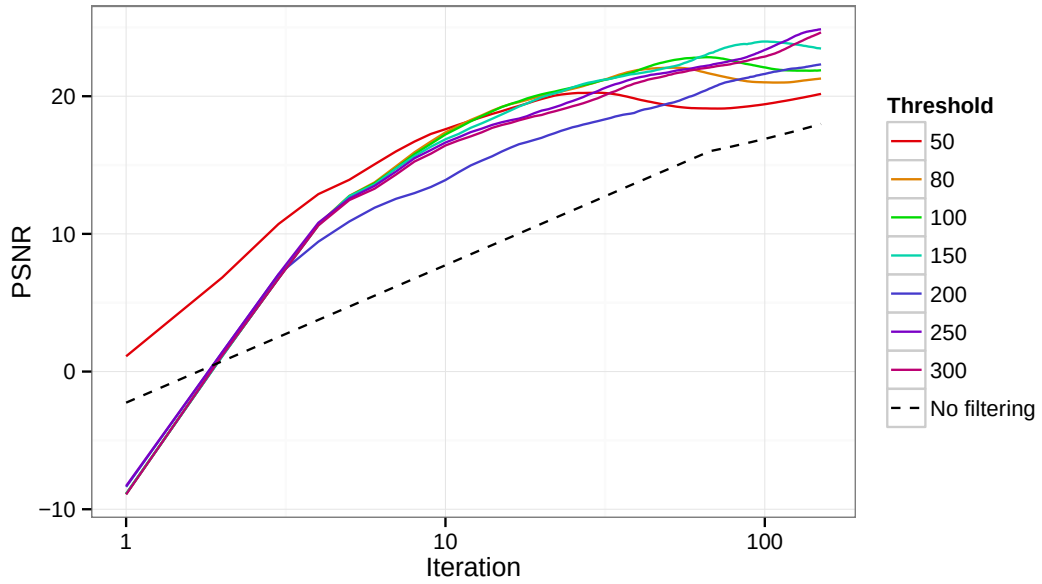


Figure 8.7: This plot shows PSNR values with respect to the reference image. The PSNR between the reference image and our image with filtered intensity maps, using thresholds from 50 to 300, is compared to the PSNR between the reference image and photon images without multi-resolution image filtering. Rendering time: ∼3.6 seconds per iteration for the image-filtered results and ∼3.3 seconds for images without filtering.

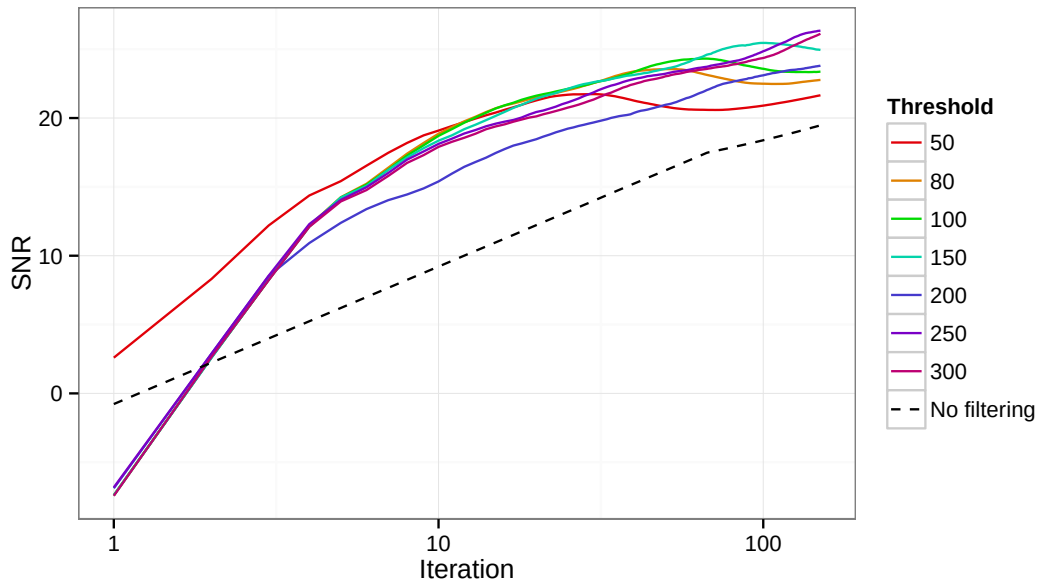


Figure 8.8: This plot shows SNR values with respect to the reference image. The SNR between the reference image and our image with filtered intensity maps, using thresholds from 50 to 300, is compared to the SNR between the reference image and photon images without multi-resolution image filtering. Rendering time: ∼3.6 seconds per iteration for the image-filtered results and ∼3.3 seconds for images without filtering.

Rendering at Interactive Frame Rates

Figures 8.9 to 8.16 show iterations 0, 5, 10, 15, 20, 25, 30 and 100 of our simulation for the test scene (Figure 8.1) presented in Section 8.1.1. The left parts of the figures show the unfiltered intensity maps, as displayed in the render view, and the right parts show the same intensity maps after applying our multi-resolution image-filtering approach. We configured our application to cast 500 000 rays per iteration to fill an intensity texture of size 2048 × 3072 texels (1024^2 per cube map face). A global threshold of 100 was used for the multi-resolution image filtering. During this simulation, each iteration took about one second to render in the 3D view. Finally, Figure 8.17 shows our reference image in the 3D view. The side-by-side views show that apart from the evaluation with objective error metrics, our multi-resolution image-filtering approach significantly enhances the visual quality of our intermediate results. We also evaluated only visually perceivable errors in Section 8.1.4 by calculating PSNR, SNR and RMSE values.

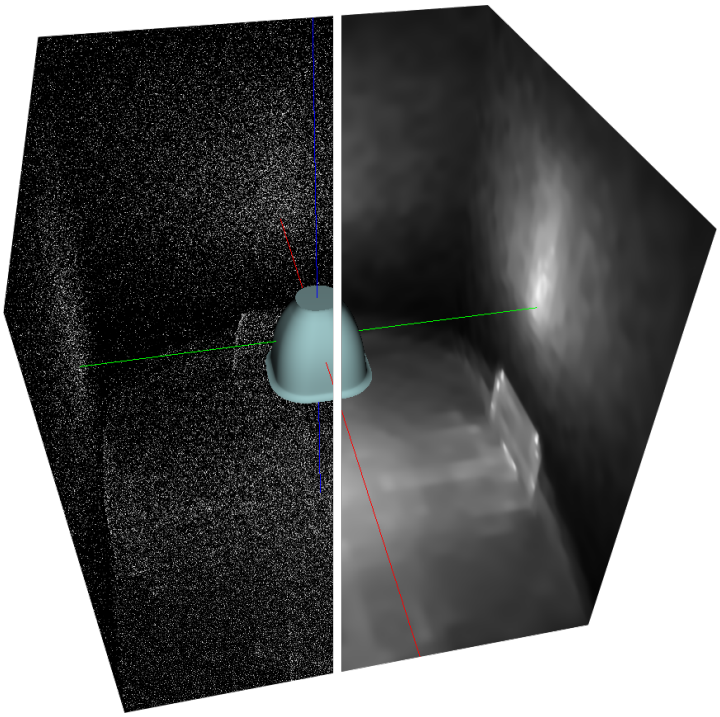


Figure 8.9: Iteration 0. Left: unfiltered intensity map. Right: filtered preview. Rendering time: ∼1 second.

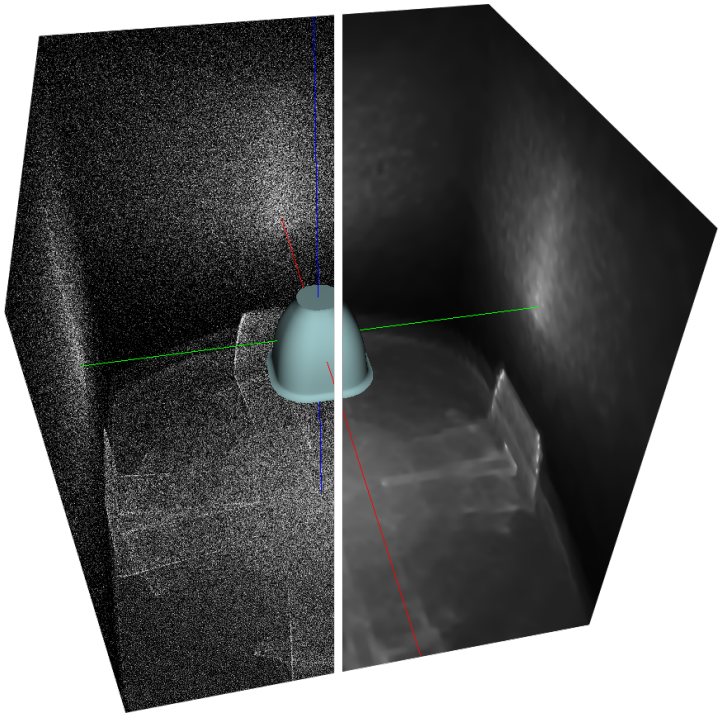


Figure 8.10: Iteration 5. Left: unfiltered intensity map. Right: filtered preview. Rendering time: ∼6 seconds (∼1 second per iteration).

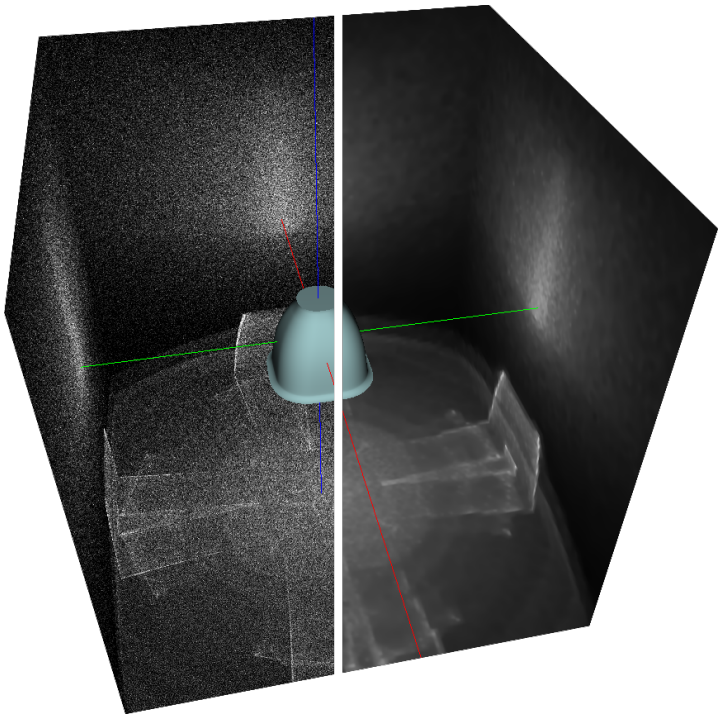


Figure 8.11: Iteration 10. Left: unfiltered intensity map. Right: filtered preview. Rendering time: ∼11 seconds (∼1 second per iteration).

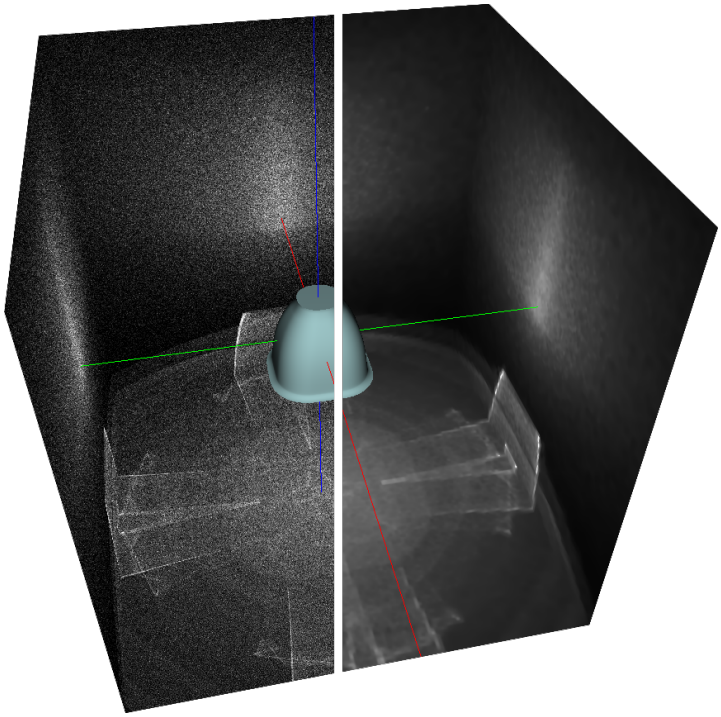


Figure 8.12: Iteration 15. Left: unfiltered intensity map. Right: filtered preview. Rendering time: ∼16 seconds (∼1 second per iteration)

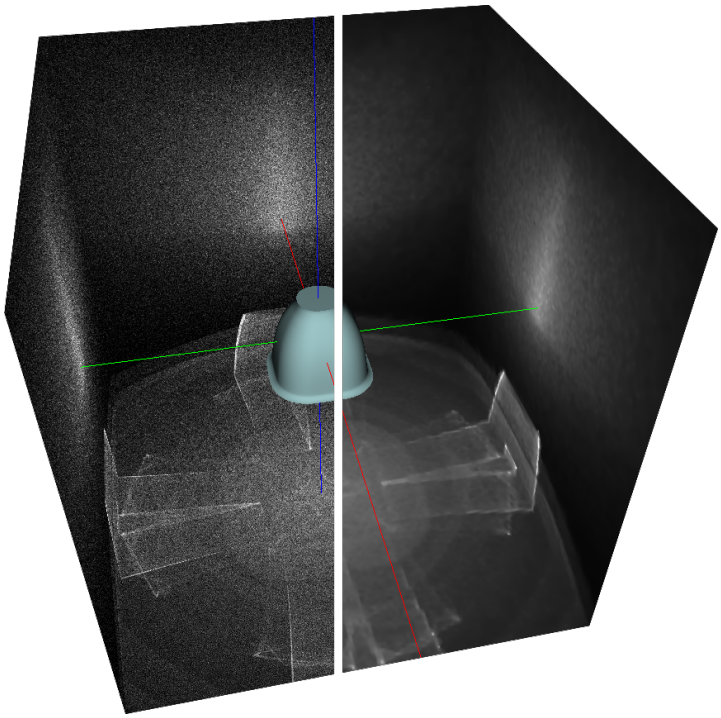


Figure 8.13: Iteration 20. Left: unfiltered intensity map. Right: filtered preview. Rendering time: ∼21 seconds (∼1 second per iteration).

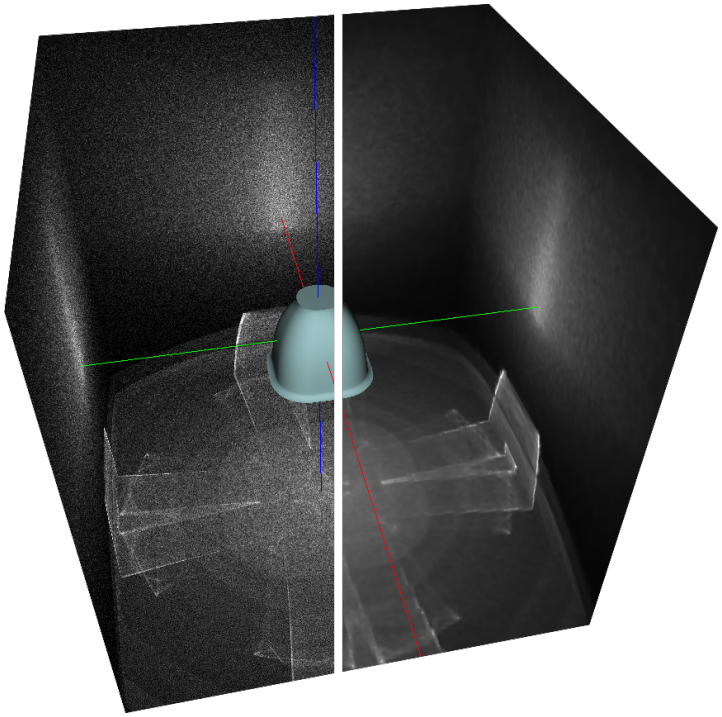


Figure 8.14: Iteration 25. Left: unfiltered intensity map. Right: filtered preview. Rendering time: ∼26 seconds (∼1 second per iteration).

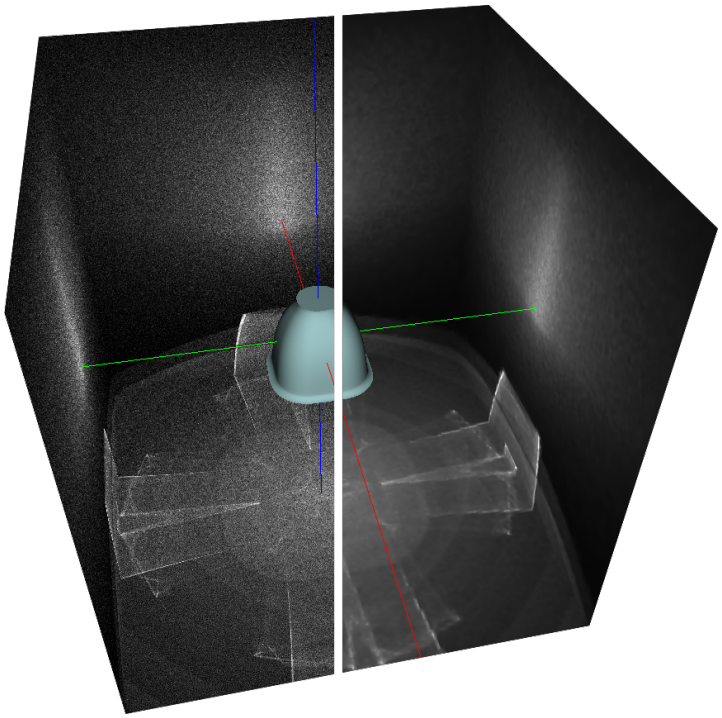


Figure 8.15: Iteration 30. Left: unfiltered intensity map. Right: filtered preview. Rendering time: ∼31 seconds (∼1 second per iteration).

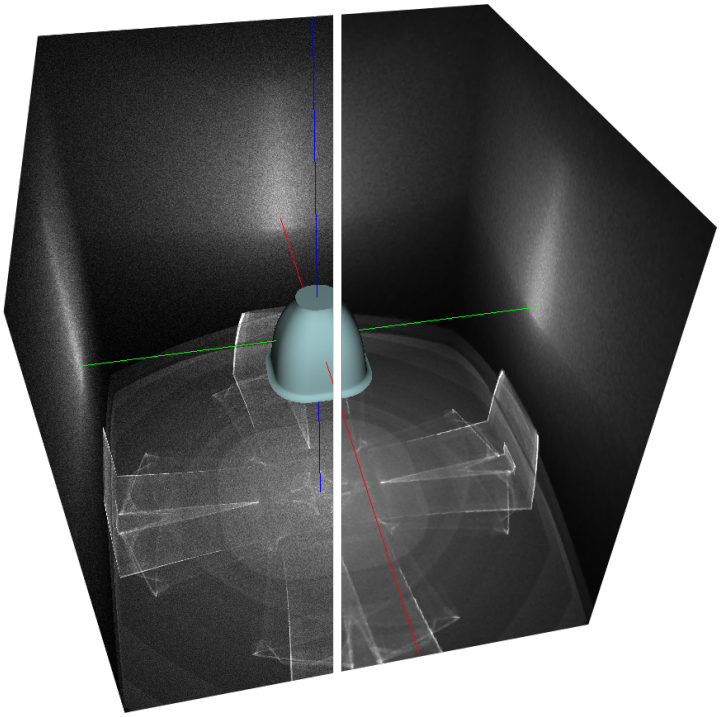


Figure 8.16: Iteration 100. Left: unfiltered intensity map. Right: filtered preview. Rendering time: ∼101 seconds (∼1 second per iteration).

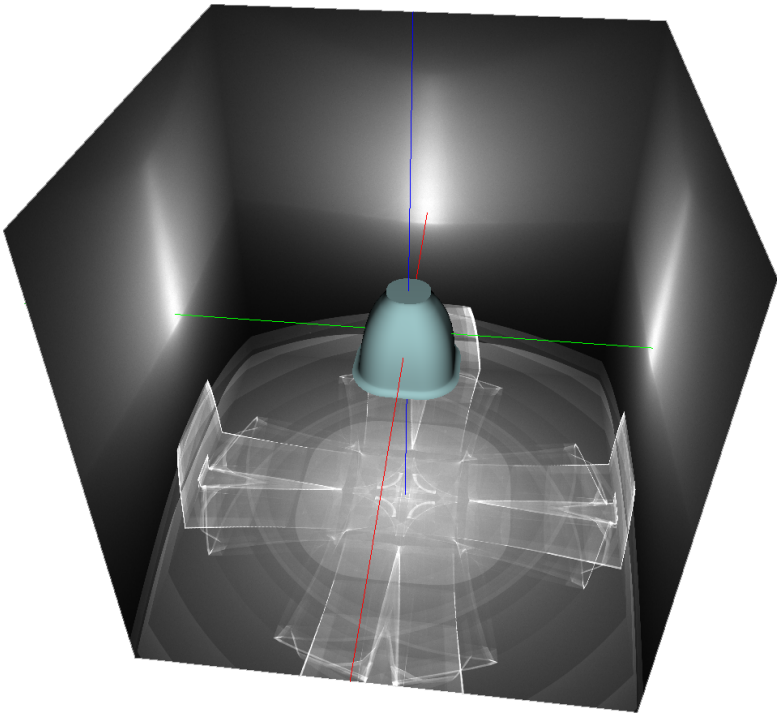


Figure 8.17: Reference image in 3D view. (unfiltered) Rendering time: ∼12.5 hours.

8.1.4 Visually Perceivable Errors

For most computers, the luminance intensity resolution is 256, meaning 256 different luminance intensities can be displayed on the screen. To that end, users can at most recognize differences in luminance of pixels on the screen in the range of 0 to 255. Each difference that we can perceive between our test images and the reference image is considered an error. To measure the visually perceivable error, we transform the image values to integer values in the range [0 255], clamping some of the brightest pixels, which are considered as outliers. After this transformation, we calculate the PSNR, SNR and RMSE between our test images and the reference image. The results, shown in Figures 8.18, 8.19 and 8.20, are similar to the evaluation results of the original data values as presented in Section 8.1.3. Future work would be to do this evaluations on tone-mapped images in order to account for the logarithmic luminance perception of the human eye.

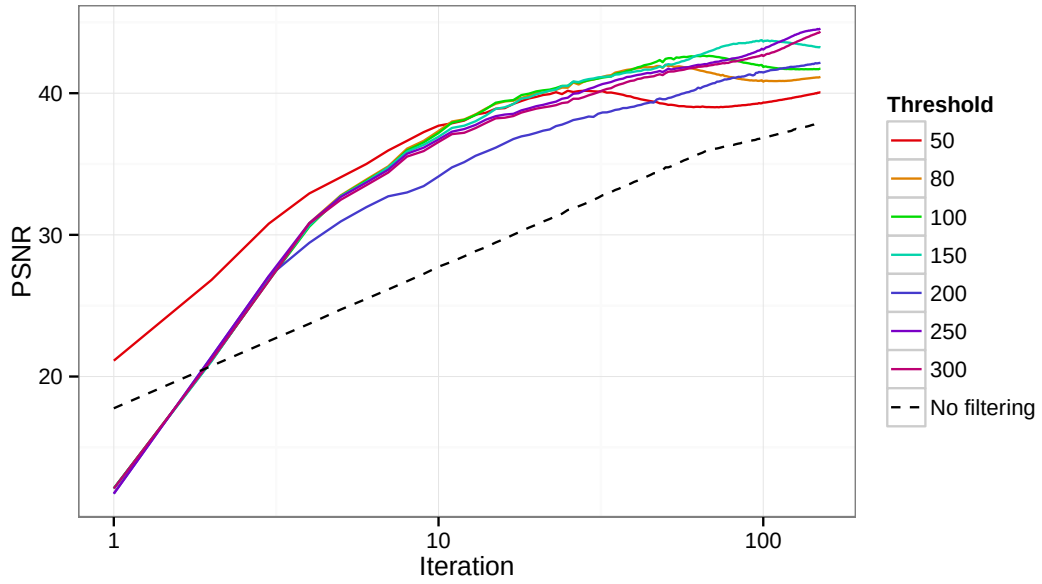


Figure 8.18: This plot shows PSNR values with respect to the reference image. The PSNR between the reference image and our image with filtered intensity maps, using thresholds from 50 to 300, is compared to the PSNR between the reference image and photon images without multi-resolution image filtering. Rendering time: ∼3.6 seconds per iteration for the image-filtered results and ∼3.3 seconds for images without filtering. Values are in the range [0 255].

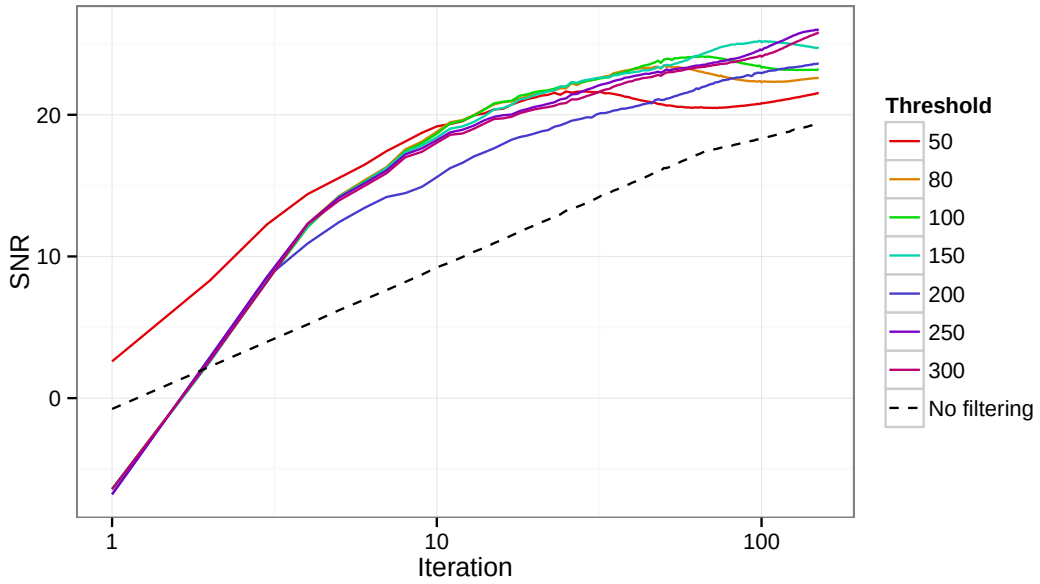


Figure 8.19: This plot shows SNR values with respect to the reference image. The SNR between the reference image and our image with filtered intensity maps, using thresholds from 50 to 300, is compared to the SNR of photon images without our multi-resolution upsampling. Rendering time: ∼3.6 seconds per iteration for the image-filtered results and ∼3.3 seconds for images without filtering. Values are in the range [0 255].

8.2 Performance

Image quality as well as performance, in terms of rendering time, of our algorithm are highly dependent on the simulation parameters adjusted by the user. A very high threshold can increase computation time for the multi-resolution image-filtering, since this results in a lot of texels that have a photon count below the threshold and need to be upsampled. The maximum number of reflections allowed for one ray can also have an impact on performance, especially in very complex scenes, where rays are likely to be reflected on the luminaire geometry for multiple times. However, the biggest influence on the performance by far, is the number of rays that are cast, to emit photons into the scene. Reducing the resolution of the intensity-map faces also reduces the number of necessary photons to obtain visually pleasing results. Depending on the simulation parameters, rendering times for one iteration can range from 0.1 seconds to 10 seconds. Figure 8.21 shows a result obtained after 100 iterations of casting 100 000 rays to fill an intensity map with a cube-map face size of 512^2. Each iteration was rendered in about 0.3 seconds with an overall simulation time of 31.329 seconds (including exporting the final intensity map). Another example is shown in Figure 8.22. This result was obtained after 100 iterations with a total rendering time of just 12.109 seconds.

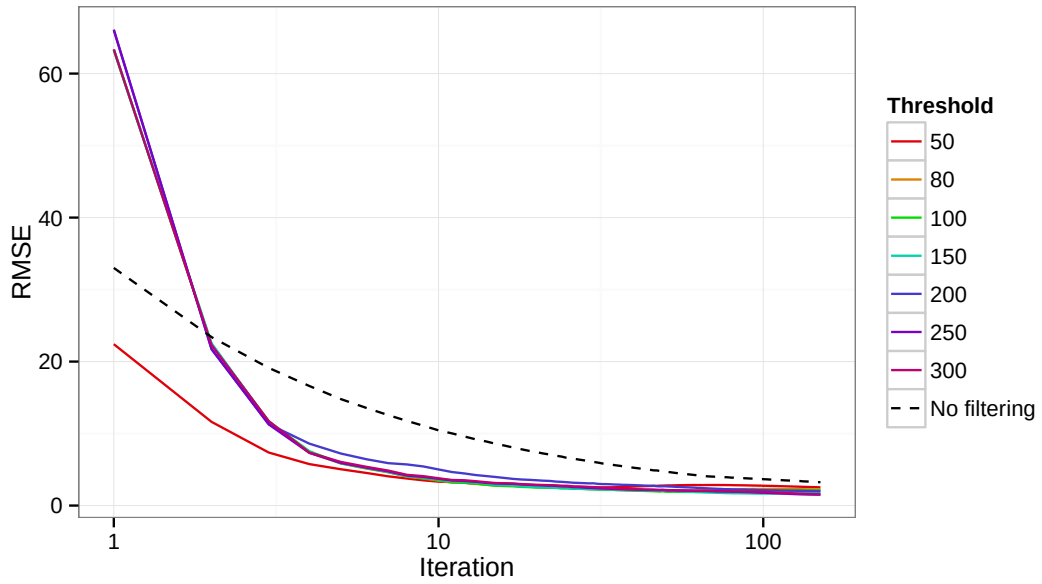


Figure 8.20: This plot shows RMSE values with respect to the reference image. The RMSE between the reference image and our image with filtered intensity maps, using thresholds from 50 to 300 is compared to the RMSE of photon images without our multi-resolution upsampling. Rendering time: ∼3.6 seconds per iteration for the image-filtered results and ∼3.3 seconds for images without filtering. Values are in the range [0 255].

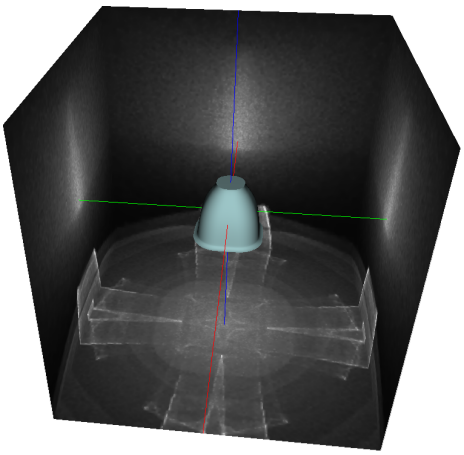


Figure 8.21: Result after 100 iterations, with 100 000 rays cast during each iteration and a threshold of 80 for a resolution of 512^2 per cube map face. Rendering time: 31.329 seconds

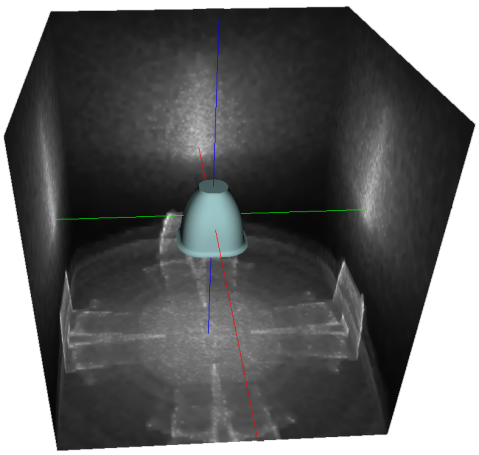


Figure 8.22: Result after 100 iterations, with 100 000 rays cast during each iteration and a threshold of 70 for a resolution of 128^2 per cube map face. Rendering time: 12.109 seconds.

8.3 Limitations

Our method excels in scenes with complex light-source geometry and complex material properties, which result in a large distribution of photons in all directions. Luminaires that concentrate their light distribution in a very distinguished, small area (like spotlights do) can benefit very little from our algorithm. Pure random photon tracing is already sufficient to get high-quality results for cases like this in just a few iterations.

The approach presented in this thesis is restricted by a trade-off between image quality and speed, both regarding the interactive previews. It is possible to decrease the number of rays cast per iteration and therefore also decrease the simulation time of the photon tracer. As a consequence, the intensity maps contain less information and our multi-resolution image-filtering approach will produce inferior results when processing them.

Because the main parts of our algorithm (the photon simulation and the image-filtering) run on the GPU and we use huge buffers for our photon storage and intensity map, the performance of the whole algorithm is highly dependent on the graphics card and its memory capacity. At the moment the implementation of our prototype is tailored to the NVIDIA GeForce GTX 970.

8.4 Discussion

State-of-the-art global-illumination algorithms that render visually pleasing images in real time are often view dependent or image space techniques, or use either some kind of averaging or approximations to calculate the illumination in a scene. They do not achieve physically correct results. Offline algorithms (like most photon mapping algorithms) on the other hand, which converge to a physically correct solution, are very time consuming and therefore not feasible for interactive applications.

Recent research ([KD13, GKDS12]) in the field of photon mapping showed that most improvements can be achieved using progressive photon mapping and parallel photon mapping implemented on the GPU. A key part in progressive photon mapping is radiance estimation, which we dismissed for our approach due to its trade-off between variance and bias. Furthermore, by calculating a radiance estimation on sparse textures, such as the lowest mipmap level of our intensity map (during the first few iterations of our simulation), high-frequency details might get lost. Future work could be to combine radiance estimation with our multi-resolution image-filtering approach.

Our algorithm combines advantages of both real-time global-illumination algorithms and offline rendering. We are not able to render in real time, but we achieve interactive frame rates, which allow the usage of our algorithm in an interactive editing tool. Even though the intermediate results of our algorithm are image-filtered and not physically correct, it converges to a physically plausible solution with a small overhead in computation time (caused by the multi-resolution image filtering). This additional time depends on the desired resolution of the resulting image and the chosen parameters for the image filtering, like the threshold or the number of mipmap levels.

Since our multi-resolution image-filtering approach has to produce smooth images of, sometimes, very sparse textures during the first few iterations, most image-filtering techniques that are not based on a multi-resolution technique, cannot produce smooth results. Other state-of-the-art image-filtering or noise-reduction techniques, like the edge-preserving smoothing model by Kniefacz and Kropatsch [KK15], or the structure-preserving image decomposition operator based on the bilateral filter, presented by Cho et al. [CLKL14], have to be adapted to serve as an interpolation technique. Both approaches need more textures, more computations and more texture lookups than our implemented bilateral interpolation (as explained in Section 6.4.4). Therefore, such techniques would increase the overall computation time of our algorithm. Note that our algorithm allows to easily exchange the implemented interpolation technique to increase image quality. However, the evaluation on how much computation time increases and image quality can be improved, by integrating such advanced techniques is future work.

The implementation of our bilateral interpolation minimizes artifacts caused by intensity leaks. However, our multi-resolution image-filtering method is still not able to produce perfect intermediate results as compared to the reference image. When constructing the mipmap image pyramid, we sum up four texels to get one value for the next higher mipmap level. This calculation causes a loss of information about the exact location of the values that are added together, which introduces an error to our upsampling procedure. To reduce this error it is necessary to implement more sophisticated methods of constructing an image pyramid.

Our evaluation shows that different thresholds produce different image qualities depending on the photon distribution in the intensity map. Because this photon distribution is hard to predict, the threshold has to be adjusted by the user in our current implementation.

An open issue of our algorithm is importance sampling. The goal of using an importance function is to increase the photon count in regions that contain details such as borders of intensity regions. Photons that result from randomly choosing ray directions, as described in Section 5.4, are stored with an energy of one. In order to to cast rays not just according to the emission profile, but also guided by an importance function, it is necessary to adapt the energy of the photons to this importance function. When casting a ray into a direction that has a certain importance, we set the energy of the resulting photon to a value inversely proportional to the importance of the chosen ray direction. As a consequence we get many photons with low energy in regions of interest, which is the desired result for these regions. Ray directions that cast photons into areas that are already quite smooth have a low importance. Hence, only very few photons are emitted in these directions, but these few photons have a very high energy, which accounts for the low importance of these ray directions. High-energy photons that are added in smooth areas therefore increase variance in these areas and are visible as noise. On the one hand, we get more photons in regions of interest, but on the other hand we introduce noise to already smooth regions. Since our goal is to produce smooth results, we have to conclude that the usage of an importance function, like presented in Section 5.4.1, is not beneficial, and was therefore deactivated in our final prototype.

CHAPTER 9

Conclusion and Future Work

We presented an interactive global-illumination algorithm that simulates the light distribution of a light source and produces visually pleasing intermediate results at interactive frame rates before it converges to a physically plausible solution, resulting in a visualization of the light distribution of a luminaire.

We combined an interactive, progressive photon tracing algorithm, using forced random sampling as described by Cornel [Cor14], with a multi-resolution image-filtering approach, based on the upsampling technique described by Nicholas and Wyman [NW09]. Our algorithm significantly enhances the output when compared to non-filtered textures. This allows us to display a visually pleasing preview after each iteration.

We compared the intermediate results, generated at each iteration with different thresholds, to a physically plausible reference image (obtained after several hours of rendering time), using objective quality metrics. The evaluation showed that our image-filtering approach is able to increase image quality (as compared to non-filtered results), especially during the first few iterations, which are most important for an interactive tool.

A luminaire editor, as shown in Figure 9.1 was developed at VRVis, as extension to the HILITE software. By providing fast previews, it allows the user to edit the geometry or material of the luminaire in real time. The editor provides a multitude of parameters (for transformation of the geometry, modification of material properties, adaptation of rendering speed and image quality,..) and shows the progress of the simulation. By developing a tool, that provides visually pleasing previews at interactive frame rates, we turned the very time-consuming iterative luminaire design into an interactive process. Furthermore, since our editor can be used during lighting design to directly edit the light distribution of luminaires, we provided a new way to combine two former totally decoupled workflows for luminaire design and lighting design.

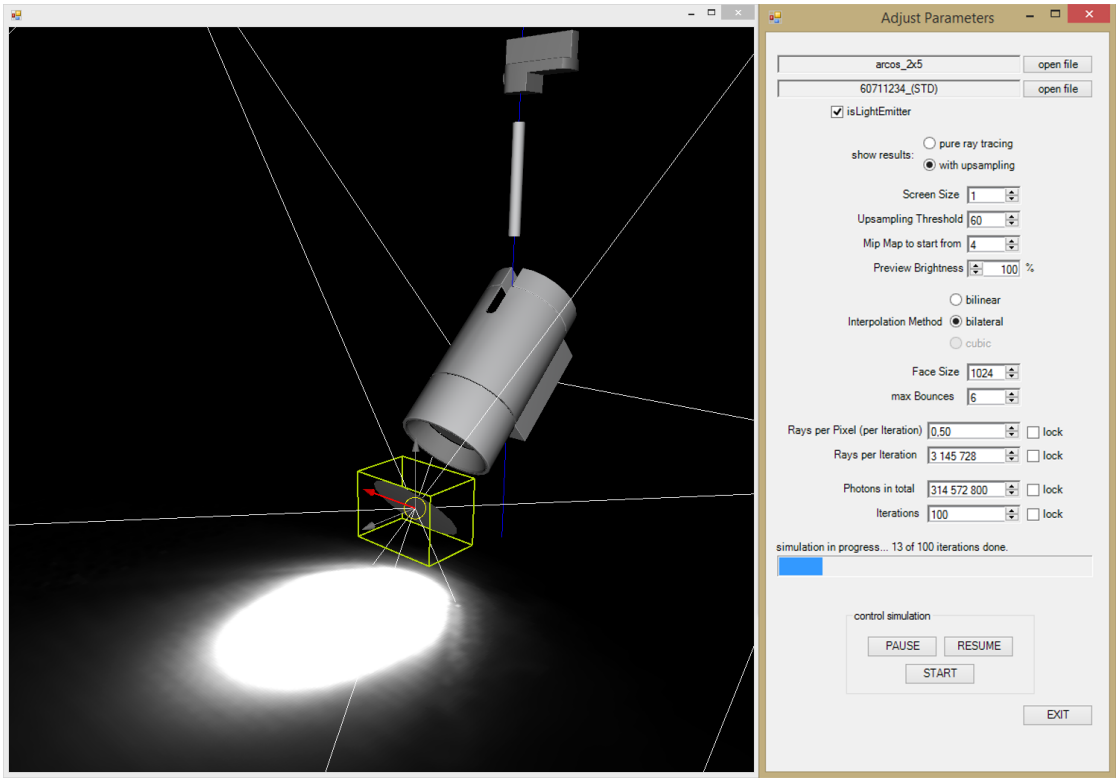


Figure 9.1: Luminaire editor.

9.1 Future Work

Adaptive Threshold

During our multi-resolution image filtering, we use a user-defined global threshold to specify what number of photons is sufficient to categorize a texel as stable. Texels that have a lower photon count are upsampled using bilateral interpolation. Texels on high mipmap levels are more likely to have higher values than the threshold and will therefore not be altered. On lower mipmap levels, more texels will have lower values than the threshold, and will hence be considered not stable. Finding a good threshold that will result in homogeneous regions being smoothed and high intensity details being kept unaltered is difficult. As our evaluation showed, different thresholds yield a different quality of results, depending on the amount of photons in a region. In our current implementation, the threshold has to be defined and adapted by the user. So future work could be calculating an optimal threshold for the whole image or separate thresholds for different image regions, as well as automatically adjusting the threshold in each iteration according to the new amount of photons that is added per iteration.

Importance Sampling

The importance-sampling scheme we presented introduced additional noise in previously homogeneous regions and was therefore neglected for the final prototype. In future work, other importance-sampling schemes, which direct more photons into regions of interest without casting a few high-intensity photons into directions of smooth regions, could be explored.

Advanced Image Filtering

Since most of the rendering time of our algorithm is consumed by photon tracing, future work could be to introduce an an image filter that can produce smooth images from very sparse textures, so we can reduce the number of photons that is necessary to produce visually pleasing results. Note that our bilateral interpolation can be exchanged by any fast edge-aware image interpolation method or image filter, which can be adapted for interpolating texel values. Most advanced image filtering techniques need more texture lookups and computation time than a bilateral interpolation. In order to include advance interpolation techniques (like the bilateral texture-filtering method presented by Cho et al. [CLKL14]), it is necessary to investigate if the gain in image quality outweighs the performance loss, or if the improved image quality allows the photon tracer to reduce the amount of necessary photons and simulation time, to account for the increased time of the image-filtering approach. A combination of radiance-estimation techniques with our multi-resolution image-filtering approach might also be beneficial.

List of Figures

Bibliography

[AMHH08] Tomas Akenine-Möller, Eric Haines, and Naty Hoffman. *Real-time rendering*. CRC Press, 2008.

[CLKL14] Hojin Cho, Hyunjoon Lee, Henry Kang, and Seungyong Lee. Bilateral texture filtering. *ACM Transactions on Graphics (TOG)*, 33(4):128, 2014.

[Cor14] Daniel Cornel. Analysis of forced random sampling. Master's thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria, October 2014.

[CRMT91] Shenchang Eric Chen, Holly E Rushmeier, Gavin Miller, and Douglass Turner. A progressive multi-pass method for global illumination. In *ACM SIGGRAPH Computer Graphics*, volume 25, pages 165–174. ACM, 1991.

[CT81] Robert L Cook and Kenneth E Torrance. A reflectance model for computer graphics. In *ACM Siggraph Computer Graphics*, volume 15, pages 307–316. ACM, 1981.

[CT82] Robert L Cook and Kenneth E. Torrance. A reflectance model for computer graphics. *ACM Transactions on Graphics (TOG)*, 1(1):7–24, 1982.

[fVRuVFGa] VRVis Zentrum für Virtual Reality und Visualisierung Forschungs-GmbH. Aardvark. `https://www.vrvis.at/projects/aardvark`. Accessed: 2016-03-20.

[fVRuVFGb] VRVis Zentrum für Virtual Reality und Visualisierung Forschungs-GmbH. Hilite. `https://www.vrvis.at/projects/hilite`. Accessed: 2016-03-11.

[GGSC96] Steven J Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F Cohen. The lumigraph. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 43–54. ACM, 1996.

[GHUPS14] Ruben Jesus Garcia Hernandez, Carlos Urena, Jordi Poch, and Mateu Sbert. Overestimation and underestimation biases in photon mapping

with non-constant kernels. *Visualization and Computer Graphics, IEEE Transactions on*, 20(10):1441–1450, 2014.

[GKDS12] Iliyan Georgiev, Jaroslav Krivánek, Tomas Davidovic, and Philipp Slusallek. Light transport simulation with vertex connection and merging. *ACM Trans. Graph.*, 31(6):192, 2012.

[Hec86] Paul S Heckbert. Survey of texture mapping. *Computer Graphics and Applications, IEEE*, 6(11):56–67, 1986.

[HOJ08] Toshiya Hachisuka, Shinji Ogaki, and Henrik Wann Jensen. Progressive photon mapping. *ACM Transactions on Graphics (TOG)*, 27(5):130, 2008.

[HSA91] Pat Hanrahan, David Salzman, and Larry Aupperle. A rapid hierarchical radiosity algorithm. In *ACM SIGGRAPH Computer Graphics*, volume 25, pages 197–206. ACM, 1991.

[HSM+14] Georg Haaser, Harald Steinlechner, Michael May, Michael Schwärzler, Stefan Maierhofer, and Robert Tobler. CoSMo: Intent-based composition of shader modules. In *Computer Graphics Theory and Applications (GRAPP), 2014 International Conference on*, pages 1–11. IEEE, 2014.

[HST13] Kaiming He, Jian Sun, and Xiaoou Tang. Guided image filtering. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(6):1397–1409, 2013.

[JC95] Henrik Wann Jensen and Niels Jørgen Christensen. Photon maps in bidirectional monte carlo ray tracing of complex objects. *Computers & Graphics*, 19(2):215–224, 1995.

[JC98] Henrik Wann Jensen and Per H. Christensen. Efficient simulation of light transport in scences with participating media using photon maps. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 311–320. ACM, 1998.

[Jen96] Henrik Wann Jensen. Global illumination using photon maps. In *Rendering Techniques' 96*, pages 21–30. Springer, 1996.

[Jen01] Henrik Wann Jensen. *Realistic image synthesis using photon mapping*. AK Peters, Ltd., 2001.

[Kaj86] James T Kajiya. The rendering equation. In *ACM Siggraph Computer Graphics*, volume 20, pages 143–150. ACM, 1986.

[KD13] Anton S Kaplanyan and Carsten Dachsbacher. Adaptive progressive photon mapping. *ACM Transactions on Graphics (TOG)*, 32(2):16, 2013.

[Kel97] Alexander Keller. Instant radiosity. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 49–56. ACM Press/Addison-Wesley Publishing Co., 1997.

[KK15] Philipp Kniefacz and Walter Kropatsch. Smooth and iteratively restore: A simple and fast edge-preserving smoothing model. *arXiv preprint arXiv:1505.06702*, 2015.

[Kra09] Martin Kraus. The pull-push algorithm revisited. *Proceedings GRAPP*, 2009, 2009.

[KSKK10] Murat Kurt, László Szirmay-Kalos, and Jaroslav Křivánek. An anisotropic brdf model for fitting and monte carlo rendering. *ACM SIGGRAPH Computer Graphics*, 44(1):3, 2010.

[KW00] Alexander Keller and Ingo Wald. *Efficient importance sampling techniques for the photon map*. Universität Kaiserslautern. Fachbereich Informatik, 2000.

[KWXM16] Chun-meng Kang, Lu Wang, Yan-ning Xu, and Xiang-xu Meng. A survey of photon mapping state-of-the-art research and future challenges. *Frontiers of Information Technology & Electronic Engineering*, 17:185–199, 2016.

[Len12] Eric Lengyel. Mathematics for 3d game programming and computer graphics. 2012.

[LTH+13] Christian Luksch, Robert F Tobler, Ralf Habel, Michael Schwärzler, and Michael Wimmer. Fast light-map computation with virtual polygon lights. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 87–94. ACM, 2013.

[LTM+14] Christian Luksch, Robert F. Tobler, Thomas Mühlbacher, Michael Schwärzler, and Michael Wimmer. Real-time rendering of glossy materials with regular sampling. *The Visual Computer*, 30(6-8):717–727, June 2014.

[LW93] Eric P Lafortune and Yves D Willems. Bi-directional path tracing. In *Proceedings of CompuGraphics*, volume 93, pages 145–153, 1993.

[LZ14] Xiao-Dan Liu and Chang-Wen Zheng. Adaptive importance photon shooting technique. *Computers & Graphics*, 38:158–166, 2014.

[MLM13] Michael Mara, David Luebke, and Morgan McGuire. Toward practical real-time photon mapping: Efficient gpu density estimation. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 71–78. ACM, 2013.

[NVI04] NVIDIA. Improve batching using texture atlases. SDK white paper, NVIDIA Corporation, Santa Clara, CA 95050, Jul 2004.

[NW09] Greg Nichols and Chris Wyman. Multiresolution splatting for indirect illumination. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, pages 83–90. ACM, 2009.

[Pho75] Bui Tuong Phong. Illumination for computer generated pictures. *Communications of the ACM*, 18(6):311–317, 1975.

[PSA+04] Georg Petschnigg, Richard Szeliski, Maneesh Agrawala, Michael Cohen, Hugues Hoppe, and Kentaro Toyama. Digital photography with flash and no-flash image pairs. In *ACM transactions on graphics (TOG)*, volume 23, pages 664–672. ACM, 2004.

[RDGK12] Tobias Ritschel, Carsten Dachsbacher, Thorsten Grosch, and Jan Kautz. The state of the art in interactive global illumination. In *Computer Graphics Forum*, volume 31, pages 160–188. Wiley Online Library, 2012.

[RPJV92] Holly E Rushmeier, Charles W Patterson Jr, and Aravindan Veerasamy. Geometric simplification for indirect illumination calculation. 1992.

[SAS92] Brian E Smits, James R Arvo, and David H Salesin. An importance-driven radiosity algorithm. In *ACM SIGGRAPH Computer Graphics*, volume 26, pages 273–282. ACM, 1992.

[Sch94] Christophe Schlick. An inexpensive brdf model for physically-based rendering. In *Computer graphics forum*, volume 13, pages 233–246, 1994.

[Shi90] Peter Shirley. A ray tracing method for illumination calculation in di use-specular scenes. In *Proceedings of Graphics Interface*, volume 90, pages 205–212. Citeseer, 1990.

[SJ13a] Ben Spencer and Mark W Jones. Photon parameterisation for robust relaxation constraints. In *Computer Graphics Forum*, volume 32, pages 83–92. Wiley Online Library, 2013.

[SJ13b] Ben Spencer and Mark W Jones. Progressive photon relaxation. *ACM Transactions on Graphics (TOG)*, 32(1):7, 2013.

[TM98] Carlo Tomasi and Roberto Manduchi. Bilateral filtering for gray and color images. In *Computer Vision, 1998. Sixth International Conference on*, pages 839–846. IEEE, 1998.

[Wal05] Bruce Walter. Notes on the ward brdf. *Proceedings of Technical Report PCG-05–06, Cornell Program of Computer Graphics*, 2005.

[War92] Gregory J Ward. Measuring and modeling anisotropic reflection. *ACM SIGGRAPH Computer Graphics*, 26(2):265–272, 1992.

[ZSXJ14] Qi Zhang, Xiaoyong Shen, Li Xu, and Jiaya Jia. Rolling guidance filter. In *Computer Vision–ECCV 2014*, pages 815–830. Springer, 2014.