Efficient From-Point Visibility for Global Illumination in Virtual Scenes with Participating Media

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Thomas Engelhardt

aus Nürnberg

Tag der mündlichen Prüfung:	07. Februar 2013
Erster Gutachter:	Prof. Dr.-Ing. Carsten Dachsbacher
Zweiter Gutachter:	Prof. Dr.-Ing. Jan Kautz

Abstract

Visibility determination is one of the fundamental building blocks of photorealistic image synthesis. Light reflected at a shading point depends on light reflected from the visible scene surfaces and on the visible light sources in the environment. However, computing visibility is computational highly demanding and global illumination algorithms, such as final gathering and bias compensation for instant radiosity spend most of their rendering time thereon. In this thesis we strive to improve the rendering performance of the aforementioned algorithms and propose methods that on the one hand compute and store visibility efficiently and on the other hand use feasible visibility approximations without introducing noticeable artefacts.

Therefore we investigate a (hemi-)spherical parametrization based on the octahedron for efficient storage of visibility at arbitrary shading points. Building on this foundation we then formulate a point-based rendering technique that efficiently computes (hemi-)spherical visibility for final gathering.

The principles of epipolar geometry allow us to exploit visibility coherence and we show how to derive an optimised sampling strategy for single scattering from point light sources in homogeneous media. Combining single scattering from many (virtual) point lights make it possible to approximate global illumination with multiple scattering by instant radiosity complemented by bias compensation. The latter corrects illumination errors from Instant Radiosity and is computational highly demanding due to a high number of visibility evaluations. For better efficiency we propose the usage of various visibility approximations in participating media and investigate several integration strategies for optimal results. Finally, we demonstrate that compensation on surfaces can be accurately approximated by a simple image-space post-process that does not require any visibility evaluations at all.

Kurzzusammenfassung

Sichbarkeitsbestimmung ist einer der fundamentalen Bausteine fotorealistischer Bildsynthese. Licht, das einem Punkt reflektiert wird, ist von den sichtbaren, reflektierende Obeflächen und Lichtquellen, d.h. der sichtbaren Umgebung an diesem Punkt, abhängig. Da Sichtbarkeitsberechnungen allerdings sehr berechnungsaufwändig sind, beanspruchen globale Beleuchtungsalgorithmen wie Final Gathering oder Bias Compensation für Instant Radiosity die meiste Zeit nur zu diesem Zweck. In dieser Arbeit streben wir danach, die Darstellungsperformanz der eben genannten Algorithmen zu verbessern und stellen Methoden vor, die einerseits Sichtbarkeit effizient berechnen, andererseits aber auch sinnvolle und effiziente Approximationen ausnutzen ohne wahrnehmbare Artefakte zu erzeugen.

Hierzu untersuchen wir eine (hemi-)sphärische Parametrisierung, die auf dem Oktaeder basiert und Sichtbarkeit effizient an beliebigen Punkten speichern kann. Aufbauend auf dieser Grundlage formulieren wir eine punktbasierte Darstellungstechnik die hemisphärische Sichtbarkeit für Final Gathering berechnet.

Weitergehend zeigen wir, dass sich basierend auf den Grundlagen der epipolaren Geometrie Sichtbarkeitskohärenzen ableiten lassen, die wir ausnutzen, um eine optimale Bildraumabtaststrategie für volumetrische Effekte in homogenen, einfachstreuenden Medien zu entwickeln. Akkumuliert man Einfachstreuung von vielen (virtuellen) Punktlichtquellen, lässt sich globale Beleuchtung mit Mehrfachstreuung auf Basis von Instant Radiosity und dazugehöriger Bias Compensation realisieren. Letztere korrigiert Beleuchtungsfehler von Instant Radiosity und ist wegen Sichtbarkeitsberechnungen sehr teuer. Zur Effizienzverbesserung des Korrekturverfahrens für Volumenbeleuchtung schlagen wir deshalb diverse Sichtbarkeitsapproximationen in heterogenen Medien vor und untersuchen effiziente Integrationsstrategien. Schließlich zeigen wir, dass sich die Korrektur auf Oberflächen durch eine einfache Nachbearbeitung im Bildraum mit hoher Genauigkeit approximieren lässt und dabei ganz ohne Sichtbarkeitstests auskommt.

Contents

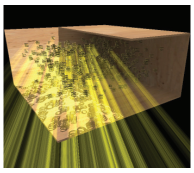
List of Figures

List of Tables

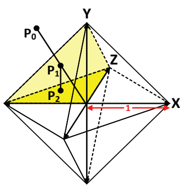
CHAPTER I

Zusammenfassung

Der Zweck der fotorealistischen Bildsynthese ist es, aus einer virtuellen Szenenbeschreibung ein darstellbares Bild zu berechnen, das der Realität zum Verwechseln ähnlich sieht.

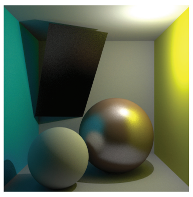


Dazu werden zuerst die von einem Betrachtungspunkt der Kamera aus sichtbaren Oberflächen berechnet. Ein Problem auf Grafikkarten ist dabei, dass für die Bestimmung der sichtbaren Oberflächen alle Objekte gezeichnet werden müssen, da verdeckte Geometrie nicht immer von vornherein vom Zeichen ausgeschlossen werden kann. Das betrifft insbesondere instanziierte Objekte. Um dies dennoch zu ermöglichen werden in Kapitel 6 dieser Arbeit zwei Algorithmen vorgestellt, die Sichtbarkeit vom Betrachtungspunkt der Szene aus auf Grafikhardware feingranular aufschlüsseln. Dadurch wird es ermöglicht, die Sichtbarkeit direkt in den programmierbaren Ausführungseinheiten der Grafikhardware abzufragen, so dass dadurch beispielsweise einzelne Dreiecke, größere Objektteile oder komplette Instanzen vom Zeichen ausgeschlossen werden können.



Nachdem der Bilderzeugungsprozess die vom Betrachter aus sichtbaren Oberflächen ermittelt hat, können diese beleuchtet werden. Dazu wird das aus der Umgebung auf einen Punkt einfallende Licht bestimmt. Unter der Annahme, dass sich die Umgebung in unendlicher Ferne zum beleuchtenden Punkt befindet, ist an jedem Punkt der Szene dieselbe Umgebung sichtbar. Folglich kann das einfallende Licht für alle Punkte der Szene vorberechnet und unter einer geeigneten Parametrisierung, in einem sogenannten Umgebungsbild (engl. environment map) gespeichert werden.

In Kapitel 7 wird der Oktaeder als eine solche Möglichkeit zur Parametrisierung der Umgebung untersucht. Dabei werden zwei Projektionsschemata betrachtet, die beide die Umgebung in ein einziges, quadratisches Bild projizieren, ohne dabei Speicherplatz ungenutzt zu lassen, wie es bei einigen vergleichbaren Ansätzen der Fall ist. Mit den beiden vorgestellten Projektionsschemata ist der Oktaeder, neben der Würfelparametrisierung, die einzige Möglichkeit ein Umgebungsbild ohne fehlerhafte Verzerrung direkt auf der Grafikkarte mittels Rasterisierung zu erzeugen.



In Kapitel 8 wird ein Verfahren vorgestellt, das Umgebungsbilder nutzt, um indirekte Beleuchtung zu berechnen. Hierzu muss lediglich ein Bild der (direkt) beleuchtenden Umgebung an jeden sichtbaren Oberflächenpunkt berechnet werden, wozu Rasterisierung jedoch zu ineffizient ist. Stattdessen wird ein punktbasiertes Darstellungsverfahren vorgeschlagen, das Dank des Einsatzes einer Hierarchie die Komplexität der zu zeichnenden Geometrie während des Zeichenvorgangs automatisch anpasst. Des Weiteren wurden speziell verzerrende Projektionen genutzt, die größere Bildbereiche solchen Bereichen der Umgebung zuordnen, die mehr zum reflektierten Licht beitragen als andere. Davon profitieren insbesondere glänzende Oberflächen. Zudem ist das Verfahren auch leicht auf Grafikhardware zu parallelisieren, wodurch interaktive Darstellungsraten erreicht werden können.

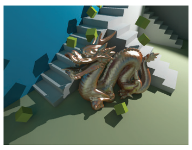
In Kapitel 9 wird Sichtbarkeit im Zusammenhang mit der Entstehung von Strahlenbüscheln in partizipierenden Medien (Volumen) untersucht und ausgenutzt. Für die fotorealistische Darstellung von Volumenbeleuchtung werden Streueffekte entlang von Sichtstrahlen akkumuliert. Dabei entsteht ein Effekt, der als Strahlenbüschel bezeichnet wird und für den hauptsächlich die Sichtbarkeit der direkten Lichtquelle an jedem Punkt entlang des Sichtstrahls verantwortlich ist. Einstreuung entlang eines Strahls zu akkumulieren ist jedoch sehr kostspielig, insbesondere wenn die Integration für jeden Abtastpunkt im Bildraum (Sichtstrahl) berechnet wird. Aus den Konzepten der epipolaren Geometrie lässt sich ableiten, dass sich Strahlenbüschel an epipolaren Linien im Bildraum ausrichten. Durch Platzieren von Abtastpunkten entlang dieser Linien kann die Anzahl der Abtastpunkte (und damit Sichtbarkeitsauswertungen) im Vergleich zu regulären Abtastmustern stark reduziert werden. Zwar muss das Streubild dann durch Interpolation rekonstruiert werden, was aber dank des Abtastschemas ohne starken Qualitätsverlust möglich ist. Die Ergebnisse bestätigen, dass sich dadurch die Effizienz stark verbessert und hohe Darstellungsraten auf Grafikhardware erreicht werden.

In vielen partizipierenden Medien wird der Grad des Fotorealismus durch

Mehrfachstreuung erhöht. Diese lässt sich auf einfache Weise durch Beleuchtung des Mediums mit einer Menge virtueller Punktlichtquellen (VPLs) approximieren.

Jedoch haben VPLs zu Punkten in ihrer direkten Umgebung einen so hohen Beitrag, dass sich in der Darstellung Artefakte in Form von hellen Flecken bilden. Diese umgeht man, indem man deren Einfluss auf nahe Punkte künstlich einschränkt, was aber über *kurze Strecken* transportierte Lichtenergie vernichtet. Mittels Kompensationsverfahren kann diese Energie wieder zurückgewonnen werden, jedoch sind diese im Gegensatz zur Beleuchtung mit VPLs nicht für die Beschleunigung auf Grafikhardware geeignet.

In Kapitel 10 wird ein approximatives, für Grafikhardware geeignetes, Kompensationsverfahren für Volumenbeleuchtung vorgestellt. Es korrigiert den bei der Auswertung von VPLs entstanden Fehler durch Abtastung des Lichttransports in der Umgebung eines Einstreupunktes. Allerdings hängt die Abtastung von einer nicht-binären Sichtbarkeitsfunktion (sichtbar/nicht-sichtbar) ab, die in heterogenen Medien teuer zu berechnen ist. Aus diesem Grund wurden in dieser Arbeit unter anderem Methoden entwickelt, die diese nicht-binäre Sichtbarkeit nur approximativ, dafür weitaus effizienter berechnen, ohne dass sich sichtbare Fehler im Bild bemerkbar machen.

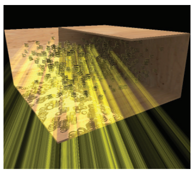


In Kapitel 11 wird ein effizientes Kompensationsverfahren für Oberflächenbeleuchtung vorgestellt. Diese Verfahren ermitteln die sichtbaren Oberflächen in der näheren Umgebung eines Punktes und gleichen den Energieverlust durch deren reflektiertes Licht aus. Wie auch im Volumen ist das Verfahren nicht für Grafikhardware geeignet, da Sichhtbarkeitsberechnungen auf Strahlverfolgung basiert. In dieser Arbeit wurde ein neuer Ansatz entwickelt, der diese Oberflächen effizient im Bildraum rekonstruiert und deren reflektiertes Licht für die Kompensation akkumuliert. Letzteres wurde zusätzlich durch ein hierarchisches Integrationsverfahren beschleunigt, wodurch das Verfahren auf Grafikhardware interaktive Darstellungsraten erreicht, das Ergebnis dennoch kaum von der Referenz zu unterscheiden ist.

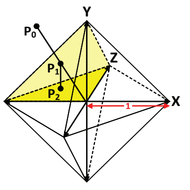
CHAPTER II

Summary

The purpose of photorealistic image synthesis is to compute an image resembling real photographs from the description of a virtual scene.

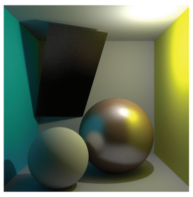


For this purpose, the surfaces visible in the image are computed first. A problem that arises on graphics hardware is that all objects have to be drawn in order to determine the visible ones. This is because hidden surfaces cannot always be determined a priori. For example instanced objects are affected by this problem and in Chapter 6 of this thesis we introduce two algorithms that break visibility down onto a fine granular level and make it possible to query visibility information directly in any execution unit of the GPU. Thereby our algorithm enables culling of individual triangles, groups of triangles or even entire instances.



After detecting the visible surfaces in the image, light reflected by them towards the observer can be computed. This in turn depends on light incident from the environment. Under the assumption that the environment is at infinite distance, it is identical at each shading point and incident illumination can be stored in a 2D image using a suitable parametrization, called an environment map. In Chapter 7 we examine the octahedron as the foundation for environment mapping and present two schemes, which make it possible to project the entire sphere into a single quadratic image. Compared to other schemes, this approach is the only one that uses a single image without introducing distortions due to the projection and rasterization on graphics hardware.

In Chapter 8 we present a method that uses environment maps to compute indirect illumination. Therefore an image of the (direct) illuminated



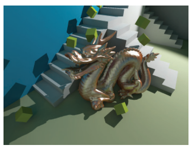
environment must be computed at each visible surface point. But because rasterization is too inefficient for this purpose, we propose a point based method which builds on hierarchical data structures and automatically adapts the complexity of the geometry during the drawing process. Beyond that, we also employ warped projections which map large regions of the environment images to directions from which high contributions to the reflected light can be expected. This improves quality especially for glossy surfaces. Finally, this approach is easily parallelized on graphics hardware which makes interactive frame rates possible.

In Chapter 9 we investigate the appearance of crepuscular rays in single scattering participating media. Photorealistic rendering of such effects accumulates scattered light along eye rays. Because visibility with respect to the light source(s) varies along eye rays, crepuscular rays emerge. Beyond that, the principles of epipolar geometry give a reason why the crepuscular rays even align with epipolar lines in image space. By placing sample points along these epipolar lines to compute the scattering effects can reduce the total number of evaluations otherwise required by regular sample patterns. Although this epipolar sampling then requires interpolation to reconstruct the scattering image, it is possible without significant quality loss. Our results confirm that this improves efficiency and enables high frame rates on graphics hardware.

Single scattering is a valid assumption as long as absorption effects outweigh scattering, otherwise multiple scattering becomes important for realism. Multiple scattering can be approximated accumulating single scattering effects from a distribution of virtual point light sources (VPLs). Unfortunatly, these VPLs create artefacts in the image due to a singularity in the illumination term. These artefacts become visible as bright splotches and can be avoided by artificially clamping or bounding the influence of VPLs on shading points. Besides removing artefacts, this also removes energy transfer over short distances, reducing overall brightness. Compensation algorithms recover this energy, but in contrast to illumination with VPLs, cannot be efficiently realised on graphics hardware.

In Chapter 10 we introduce an approximate compensation method for multiple scattering which is suitable for graphics hardware. The method corrects the error due to bounding, sampling the light transport in the close environment of the point where bounding occurred.

This however depends on a non-binary visibility function (visible/not visible) which is costly to compute in heterogeneous media. To alleviate rendering costs we develop an approximate approach which is far more efficient to compute, but nonetheless sufficiently accurate to avoid noticeable artefacts in the final image.



In Chapter 11 we present an efficient compensation method for surface instant radiosity. Bias compensation detects the visible surfaces in the close environment of a shading point and compensates the energy from light reflected by these surfaces. As in the volume, this approach is not suitable for graphics hardware, because of the abundant visibility queries. To remedy this situation we propose a new approach which efficiently reconstructs these surfaces and accumulates their reflected light for computing the compensation term. The latter has been realised using a hierarchical image-space integration scheme which accelerates this process. We show that our method achieves interactive frame rates on graphics hardware computing images that are hardly distinguishable from the reference solutions.

CHAPTER 1

Introduction

SINCE Ivan Sutherland invented *SketchPad* [Sutherland, 1963], the first computer program that relied on graphics for user interaction, the synthesis of digital images evolved into a ground-breaking research area, and eventually became an integral part of everyday life. For example, more and more electronic devices, such as desktop computers, mobile phones or TV sets, are controlled by the means of graphical user interfaces. But besides such mundane applications, computer graphics are also heavily used in various scientific disciplines to visualize and interpret numerical data for gaining insight and drawing new conclusions. Medical visualization, for instance, allows scientists and physicians to diagnose diseases without overly burdening patients with dangerous and lengthy procedures.

A particularly interesting field in computer graphics is the creation of photorealistic images. Simply speaking, photorealistic rendering computes an image of the appearance of virtual objects under realistic illumination. A prominent field where realistic image generation is employed is the entertainment industry. By means of digital visual effects rendered on a computer, real film footage can be augmented or even entirely replaced. Even creators of animated films that do not strive for photorealism, but set out to produce a particular artistic style, rely on principles of realistic image synthesis to accurately light virtual scenes. Realistic rendering methods for the film industry are heavily geared towards high image quality, for which rendering times up to several minutes per frame can be afforded. The same goal has been set by the games industry, however, within a time budget of several milliseconds per frame. With increasing computational power and improved real-time algorithms they bring realistically looking digital worlds to life, in which players can immerse themselves. Whereas the entertainment industry uses realistic rendering methods in an artistic context, allowing them to trade off accuracy for computational efficiency as long as the results look plausible, other applications demand high fidelity.

First and foremost lighting engineers benefit from realistic rendering methods. Accurate light transport simulation makes it possible to validate the effectiveness of newly-designed light fixtures. Beyond that, preview images of lighting arrangements for direct and ambient illumination in rooms and buildings can be generated and approved before anything is built. This is also closely related to architectural design. Architects can create preview images of buildings or their interiors for any given artificial or natural illumination condition created by lamps or daylight at any season or time of the day. In the same manner, product development utilizes realistic rendering algorithms. For instance, the automotive industry can evaluate newly-developed car finishes on any car by applying a computational model of the finish to a digital model of the car. Last but not least, the generation of realistic digital images supports training of robotic systems that rely on visual information, such as sorting-machines. Instead of building faulty products and filming them, digital images can be fed into the recognition software for training.

1.1 The Realistic Image Synthesis Framework

As we have seen, realistic image synthesis has many fields of application. But so far, we have not mentioned how such images are generated. As previously pointed out, realistic image synthesis computes the appearance of virtual objects under realistic illumination. This process can be described within the three orthogonal areas of *local reflection*, *light transport*, and *visual display* [Greenberg et al., 1997].

Local reflection: Local reflection describes the light scattering or reflection properties at a surface point given the incident light field, i.e. the incident light from all directions. For instance, when looking at a wall coated with matte paint, the wall appears the same under all viewing angles, whereas one notices highlights and fuzzy reflections of the environment when looking at car paint depending on the viewing angle. An extreme example is a mirror sphere which appears completely different under each viewing angle.

Light transport: Light transport is the integral component in realistic image synthesis that computes physically correct illumination. It can be imagined as small light particles (photons) that leave the light sources in straight paths and carry a discrete amount of energy. After being emitted, particles collide with objects, where they are either absorbed or scattered into different directions, depending on the local reflection model. The reflected particles impinge on other objects, and this process continues until an equilibrium of absorbed and newly emitted particles is reached. This equilibrium state of light or radiative energy can be expressed by means of a simple mathematical equation, the so called rendering equation [Kajiya, 1986], which will be formally introduced in Chapter 3.2. By means of a virtual camera sensor, a response to the equilibrium distribution of light can be computed. Formally this is expressed by the *measurement equation* and ultimately, the response is converted into a displayable image.

Visual display: On the one hand light transport is a physical simulation and computes spectral quantities that are measured by a virtual camera sensor. A display device, on the other hand, cannot process this information because it operates in colour spaces. The task of a visual display algorithm is to convert the measured (wavelength dependent) energy into a value that can be processed by a display device. This is challenging, as the range of physical quantities significantly exceeds the displayable range of colours, and dynamic range compression must be applied. This is tackled by *tone mapping* operators that employ various heuristics based on the behaviour of the *human visual system (HVS)* under natural lighting conditions. For instance, the HVS reacts more sensible to changes in the illumination in poorly lit rooms so we can still recognize our surroundings, whereas the sensitivity to changes in illumination is greatly reduced in brightly lit environments. This behaviour of the HVS is exploited by tone mapping operators to boost or compress the dynamic range, and to obtain a natural looking image within the limited colour space of the display device.

1.2 Motivation

Although all areas of the aforementioned framework for realistic image synthesis are well-understood and researched, computing a physically accurate image is still a time consuming process. The reason for that solely lies in the light transport simulation. Preparing an image for display is only a post-process executed on the final image. But even when employing the simplest local reflection model, a constant factor, a mathematically unbiased solution of light transport [Kajiya, 1986; Lafortune, 1996; Veach, 1997] easily takes

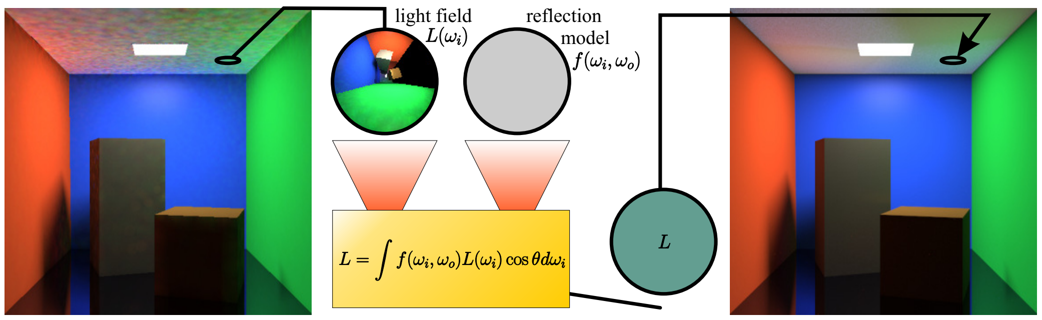


Figure 1.1: *Final Gathering improves the quality of approximate light transport solutions, such as photon mapping (left). It samples incident radiance from the visible surfaces in the environment (light field) at each shading point (middle) and computes reflected radiance (right). Final gathering for this example required 30x more time than directly visualizing the photon map (left).*

several hours to compute.

One of the reasons for this are from-point visibility computations, among others. For example, final gathering samples reflected illumination from the *visible* surfaces in the environment of a point using ray tracing (Figure 1.1). This is useful to improve the solutions of approximate light transport algorithms, such as photon mapping [Jensen, 1996], but the gathering process easily requires several hours due to visibility computations, and thus does not lend itself to interactive applications.

The same is true for instant radiosity (IR) [Keller, 1997], which first computes an approximate and biased solution of light transport, and corrects the error afterwards using *bias compensation* [Kollig and Keller, 2006], a technique that forbids interactive rendering of global illumination [Dachsbacher and Stamminger, 2005; Ritschel et al., 2008]. The reason for this is that it is similar to final gathering and requires costly visibility computations using ray tracing. IR and bias compensation are also possible in the presence of participating media [Raab et al., 2008], but visibility becomes even more intricate, as it is no longer binary (visible or invisible), due to extinction of light in the medium.

A typical approach for interactive global illumination applications is to reduce the amount of necessary computations employing a regular sub-sampling scheme in image space. The effects are rendered only for a low-resolution image, thereby sub-sampling visibility from the camera, and subsequently up-scaled to the full resolution. For indirect illumination, this approach is feasible because it varies smoothly over surfaces [Ward et al., 1988]. For crepuscular rays created by volumetric scattering in participating media it is non-optimal because crisp features created by this effect appear blurred.

Last but not least, visibility is not only an integral part of global illumination effects, but knowledge about visibility can also be used to accelerate the image generation process. For example, rasterization pipelines suffer from the overdrawing problem, i.e. invisible surfaces are drawn into the final image, but eventually replaced by visible surfaces during the process. Hardware-assisted culling techniques can alleviate these effects, but they operate only on entire objects. This prevents fine-grained culling on graphics hardware, i.e. triangles (or sets thereof) cannot be culled individually for improving rendering performance for costly shaders.

1.3 Outline

In this work we address the aforementioned visibility problems and present several different approaches for improvement. A detailed list of our contributions can be found below, where we additionally point out the particular visibility problem addressed. In Chapters 2 to 5 we first introduce preliminaries and fundamental concepts used in this thesis, whereas the following chapters present our contribution.

Preliminaries and Previous Work

In Chapter 2 we first give a brief introduction into image generation with ray tracing and rasterization. In particular, we discuss the areas of application and algorithms that accelerate visibility computations.

As mentioned earlier, photorealistic image synthesis is based on radiometric concepts, the reflection integral and the (volumetric) rendering equation. These concepts are introduced in Chapter 3 and algorithms that build on these fundamentals are briefly reviewed in Chapter 4.

Finally, we summarize the possibilities for storing directional data in 2D images in Chapter 5, including quality of representation and generation thereof.

Contributions

In the following chapters, we present our contributions:

- In Chapter 6 we present *granular visibility queries* that enable visibility determination of individual triangles of an object and efficient culling thereof to accelerate the execution of costly shaders in rasterization pipelines. This chapter is based on our work [Engelhardt and Dachsbacher, 2009] presented at the *Symposium on Interactive 3D Graphics and Games 2009 (I3D'09)*.

- In Chapter 7 we introduce *octahedron environment maps* as efficient means for compact storage of omni-directional (visibility) data on GPUs. We discuss interactive creation and usage thereof, discuss quality of representation, and mention possible applications. This chapter is based on our paper [Engelhardt and Dachsbacher, 2008] presented at the *Vision, Modeling and Visualization Workshop 2008*.

- In Chapter 8 we describe *micro-rendering*, an interactive technique for high quality final gathering on GPUs. This chapter is based on our collaborative work [Ritschel et al., 2009a] presented at *SIGGRAPH Asia 2009*.

- In Chapter 9 we present *epipolar sampling* an efficient sub-sampling technique for interactive rendering of crepuscular rays. We exploit concepts of epipolar geometry to reduce the number of (visible) scattering computations in screen space, while preserving the crisp features of crepuscular rays. This chapter is based on our publication [Engelhardt and Dachsbacher, 2010] presented at the *Symposium on Interactive 3D Graphics and Games 2010 (I3D'10)*.

- In Chapter 10 we develop an efficient *approximate bias compensation* technique for rendering multiple scattering with virtual point light methods sub-sampling incident illumination from VPLs (binary visibility) and using approximate non-binary visibility for short distance light transport. This chapter is based on our publication [Engelhardt et al., 2012] presented at *Pacific Graphics 2012*.

- In Chapter 11 we develop *screen-space bias compensation*, a reformulation of bias compensation that allows us to extract visibility information directly from screen space. This makes it possible to accelerate bias compensation on the GPU achieving interactive frame rates. This chapter is based on our work [Novák et al., 2011] presented at the *Symposium on Interactive 3D Graphics and Games 2011 (I3D'11)*.

CHAPTER 2

From Point Visibility for Image Generation

IMAGE synthesis is a complex process that involves several tasks to turn a description of a virtual scene into a displayable image. The scene is viewed from the position of a virtual camera that defines an image plane. From this perspective, an algorithm computes the *visible* surface through each image sample and records the corresponding colour value returned from a shading algorithm. For the sake of simplicity, we assume objects are represented by triangle meshes.

In this chapter we review *ray casting*, or *ray tracing* in its recursive form, and *rasterization*, the two fundamentally different algorithms for computing the visible surfaces from an arbitrary point of view (from-point visibility). Integral to both approaches is the number of (triangle) primitives that must be processed, and in the following we discuss this matter of efficiency mentioning techniques that quickly cull hidden primitives from image generation.

2.1 Ray Tracing

Simple ray tracing, as described by Appel [1968], is an image generation technique that is based on intersections between rays and objects or the primitives they are made of, respectively. The principle is simple: For each sample in the image plane, a *primary ray* in 3D world space is created. The position of the virtual camera is taken as ray origin, and the direction is constructed so it pierces the image sample (Figure 2.1 left). In the simplest form, ray tracing iterates over a list of all primitives and tests the ray for an intersection, thereby finding all potentially visible surface points. By keeping track of the intersection distance, the algorithm determines the visible surface point by selecting that which is closest to the camera.

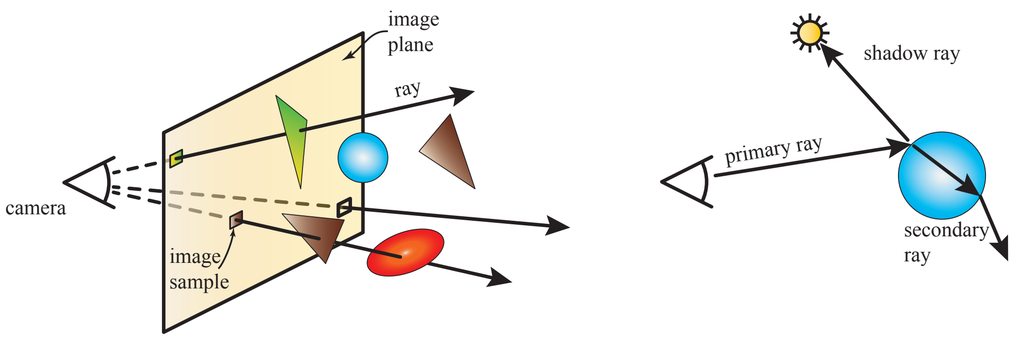


Figure 2.1: ***Left:*** *Ray casting shoots rays with the camera's position as origin through each image sample to intersect the scene objects. The closest intersection is recorded in the image plane.* ***Right:*** *Recursive ray tracing spawns secondary rays for refractions and reflections.*

2.1.1 Applications

After the visible surface has been determined by ray casting, the shading operation is executed at the intersection point. This for instance may include evaluating local illumination models such as the Phong model [Phong, 1975]. Detecting the visible surfaces from the camera is not the only purpose ray casting can be used for. Since rays can be arbitrarily defined, *shadow rays* can be used to test if any light reaches the surface point, testing its visibility from the light's point of view. Even shading operations can spawn new *secondary rays* recursively in order to handle transparency, refractions, reflections [Whitted, 1980] (Figure 2.1 right), or even diffuse inter-reflections [Cook and Torrance, 1982]. This recursive evaluation of visibility is then referred to as *ray tracing*, and because of the generality we will use this term henceforth in our discussions. Not only shading operations benefit from the flexibility of rays. Camera models can easily be extended to include depth of field [Cook and Torrance, 1982] or sophisticated projection schemes for non-planar viewports [Blinn and Newell, 1976; Heidrich and Seidel, 1998]. Beyond that, rays are an invaluable tool for light transport simulation because entire paths that describe the propagation of light in virtual scenes can be constructed [Pharr and Humphreys, 2010].

2.1.2 Efficiency

Unfortunately, ray tracing is computationally highly demanding because one must compute an intersection with each primitive for each ray, even with ones that are not intersected. To remedy this situation, researchers

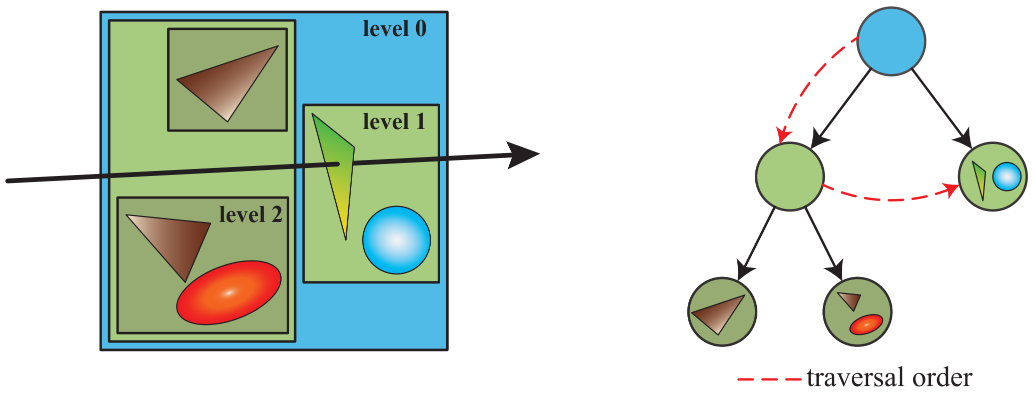


Figure 2.2: ***Left:*** *A bounding volume hierarchy intersected by a ray.* ***Right:*** *Sub-trees are skipped if the ray does not intersect the sub-trees' bounding boxes.*

have developed acceleration structures that quickly cull large sets of hidden primitives from ever being intersected. All these approaches are based on subdivision schemes.

Grids: Uniform grids [Fujimoto et al., 1986] divide the space containing the scene into rectangular cells, so called voxels, which store links to the contained geometry. Using a 3D digital differential analyzer (an incremental line drawing algorithm in 3D space), traversal quickly enumerates all voxels intersected by the ray and the geometry within. Later this approach was further extended, using hierarchical grids [Jevans and Wyvill, 1989] or grids specialised for rays with specific origins and directions [Hunt and Mark, 2008].

Hierarchical space subdivision: Hierarchical space subdivision schemes use splitting planes for recursively dividing the available space into smaller subregions. During traversal, the subregions a ray intersects are recursively refined until a subregion contains only primitives, which are then intersected. For this purpose, Glassner [1984] proposed octrees based on a rigid space subdivision scheme, but kd-trees [Kaplan, 1985] have proved to be more efficient due to adaptive placement of splitting planes. The latter has been optimized using *surface area heuristics* (SAH) [MacDonald and Booth, 1990]. The optimal position of a splitting plane is derived so that the probability of traversing the created subregions is minimized. Further tuning [Havran and Bittner, 2002], and improved [Wald and Havran, 2006] and parallel [Shevtsov et al., 2007] construction algorithms have contributed to the fact that the SAH has become the dominant construction algorithm for good ray tracing hierarchies.

Hierarchical object subdivision: Object subdivision schemes, such as *bounding volume hierarchies (BVH)* [Rubin and Whitted, 1980; Kajiya, 1986], pursue a slightly different strategy. BVHs use tight bounding volumes for the scene's primitives. Bounding volumes are recursively split finding tighter bounds for the remaining primitives in each step (Figure 2.2). As for kd-trees, elaborate construction schemes exist. Goldsmith and Salmon [1987] compute best-fitting bounding boxes based on the probability ever being hit by a ray, similar to the SAH algorithm, which has also been applied to BVH construction [Wald, 2007].

Interactive ray tracing: Ray tracing has long been limited to offline rendering applications, but ever increasing computational power and the availability of highly efficient acceleration structures spawned the field of interactive and real-time ray tracing. Much pioneering work was done by Wald [2004]. An important aspect of interactive applications are dynamic scenes, but rebuilding acceleration structures in each frame is prohibitive due to the costly SAH. A logical choice therefore was using the simpler grids [Wald et al., 2006], but BVHs have also been extended to deformable scenes [Wald et al., 2007a,b].

The first steps in this field were still done on the CPU, but ray tracing gradually moved onto the GPU thanks to its highly parallel hardware architecture and programmability. First results were shown by Purcell et al. [2002] who developed a GPU-based photon mapper. The lack of traversal stacks that are required for traversing acceleration structures made ray tracing on the GPU difficult [Horn et al., 2007] and stackless alternatives were developed [Popov et al., 2007]. In early days the possibilities of programming GPUs was limited, hindering the advances in GPU-based ray tracing, until the advent of GPGPU architectures [NVIDIA Corporation, 2011]. Since then impressive results have been achieved that show hundreds of million of rays per second [Aila and Laine, 2009] for solutions tailored to specific problems. Later, entire ray tracing solutions [Parker et al., 2010] emerged that can be used for interactive rendering, but also to accelerate offline image generation.

Not only ray tracing benefits from the GPU's computational power. The costly construction of good acceleration structures can also be accelerated with GPUs. Lauterbach et al. [2009] showed that bounding volume hierarchies for fully dynamic scenes with moving and deforming objects can be

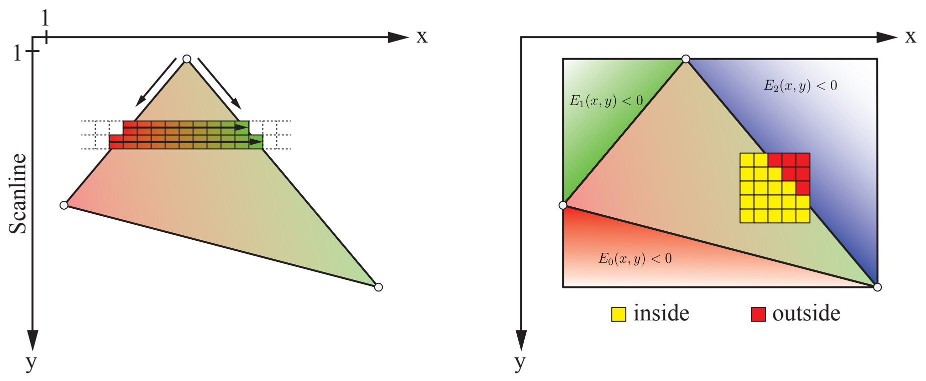


Figure 2.3: ***Left:*** *Scanline rasterization iteratively processes the image samples on each scanline segment enclosed by the triangle's edges.* ***Right:*** *For all image samples within the triangle's bounding box the signs of all three edge equations determine if a sample is covered by the triangle.*

built on the GPU for interactive rendering. Similarly, Zhou et al. [2008] demonstrate interactive kd-reconstruction for deformable scenes on GPUs.

2.2 Rasterization

In contrast to ray tracing, rasterization in its classical sense completely operates in the 2D image plane. This implies that a (triangle) primitive must first be projected onto the image plane before all image samples that are covered by the projection are determined. To avoid artefacts from projection, triangles must be clipped in projective space [Foley et al., 1990] beforehand. Early *scanline rasterization* algorithms [Foley et al., 1990] walked along triangle edges interpolating triangle barycentric coordinates over the sections of the scanlines falling between a triangle's boundaries (Figure 2.3 left). Because the algorithm operates on projected triangles, perspective-correct interpolation [Heckbert, 1989] must be used to recover barycentric coordinates of the corresponding triangle in 3D space. Pineda [1988] developed an alternative approach and derived edge equations of the form

$$E_i(x,y) = A_i x + B_i x + C_i, i \in \{0,1,2\}$$

for each triangle edge. An image sample is covered by the (projected) triangle when all edge equations have positive sign (Figure 2.3 right). This is trivial to parallelize, because each image sample can be processed independently. However, it is prohibitively costly when all image samples are tested

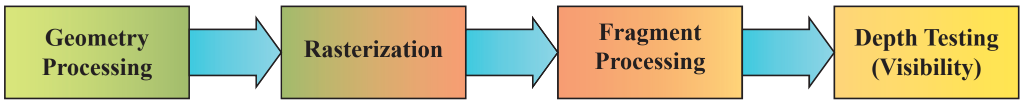


Figure 2.4: *A simplified rendering pipeline: (1) Before triangles are rasterized, geometry processing transforms, clips and projects them onto the image plane. (2) Rasterization computes sample coverage and interpolates barycentric coordinates. (3) Fragment processing executes shading operations and (4) depth testing decides whether the fragment will be visible.*

and Pineda [1988] proposed to test only those within the 2D bounding box of the triangle (also called *binning*). Later, edge equations were extended to homogeneous space [Olano and Greer, 1997]. Then rasterization operates in post-projective space, which avoids clipping entirely. Davidovič et al. [2012] developed 3D rasterization, avoiding projection entirely, making rasterization onto non-linear viewports possible.

However, rasterization only enumerates the pixels covered by a triangle. To resolve visibility, a z-buffer must be used. Because rasterization cannot stand on its own for image generation, it is part of a *rendering pipeline*, as implemented by modern DirectX [Microsoft Corporation, 2010] and OpenGL [The Khronos Consortium, 2011]. The rendering pipeline takes care of all tasks required before primitives can be rasterized (projection, clipping) and performs shading as well as visibility testing on fragments produced by the rasterizer. A fragment is a structure containing the address of an image sample and interpolated attributes, including a depth value.

Ultimately, the depth value is used for depth testing [Catmull, 1974]. A z-buffer stores a depth value for each image sample in the output, which is only replaced if the produced fragment passes the depth test. Typically this is the case if the depth value is smaller than the contents of the depth or z-buffer.

2.2.1 Applications

Rasterization is the main image generation algorithm for interactive rendering. The reason is twofold: First, rasterization is simple because it does not require complicated data structures, thus easily supports fully dynamic scenes with animation, moving objects, and deformations; an aspect ray tracing struggles with. Also, the order in which primitives are processed is clearly defined (Figure 2.4), which is ideal for hardware-based feed-forward pipelines. Second, rasterization, fragment generation, shading, and depth testing are completely independent of each other, i.e. rasterization can easily be parallelized. Modern GPUs have dedicated hardware rasterizers which achieve triangle throughputs of hundreds of millions per second.

2.2.2 Efficiency

Two aspects are important to maintain high efficiency in traditional rasterization pipelines (Figure 2.4): First, costly shading operations for hidden surfaces (or fragments) must be avoided. Second, geometry processing of hidden surfaces must also be avoided, culling them prior to submitting them to the pipeline.

Fragment culling: Visibility is resolved after shading because shaders may discard fragments or change their depth values. This implies that shading is also executed for hidden surfaces, and should be avoided for expensive shaders (e.g. lighting computation). In many cases rendering pipelines on the GPU can cull large blocks of fragments autonomously using early z-testing. Essentially, blocks of fragments are tested against a hierarchical depth-buffer [Greene et al., 1993; Greene, 1999] before they enter the shading stage. However, this is only possible if the shader does not discard fragments or change their depth value. Finally, the rendering order is also important, i.e. primitives should be submitted in order of increasing distance to the camera's position. This is because culling depends on the current contents of the depth buffer. Drawing the nearest primitives first increases the likelihood that hidden surfaces are culled before shading.

A more reliable approach to avoid shading for hidden surfaces is the geometry buffer [Saito and Takahashi, 1990]. First, images of shading parameters (position, normal, etc.) are rendered (potentially exploiting early-z culling), and second, shading is executed for each image sample using the previously rendered parameters as input. This is similar to ray tracing which also executes shading only for visible surface points.

Geometry culling: To maintain high rendering efficiency, it is not only crucial to avoid shading hidden surfaces, but also to avoid submitting geometry to the rendering pipeline that is not visible in the final image. This is the ultimate goal for a huge class of culling algorithms, which have been a vital part of computer graphics since the early 1970s, and a huge body of research exists. A classification of visibility problems and comprehensive overviews are given by [Cohen-Or et al., 2003; Bittner and Wonka, 2003].

Frustum culling [Akenine-Möller et al., 2008] is a simple method to avoid geometry processing, only submitting geometry to the rendering pipeline that lies within the camera's viewing frustum. Detecting hidden surfaces within the viewing frustum, however, is much more sophisticated and output-sensitive. On modern GPUs, this is possible with hardware occlusion queries (HOQ). They count the number of visible image samples by rasterizing geometry, such as bounding boxes, against the z-buffer. After reading back the query result, a decision is made whether to render the object or not. This is a fundamental building block for many algorithms. For example, many implementations of cell and portal algorithms [Luebke and Georges, 1995] determine the visibility of portals with HOQs. Reading back the query result implicitly synchronizes CPU and GPU execution which is detrimental to performance. This thereby induced read-back latency can be avoided querying the result not earlier than three frames after issuing the HOQ [Microsoft Corporation, 2010]. Yoon et al. [2003], Bittner et al. [2004], and Mattausch et al. [2008] demonstrate how to use HOQs for hiding the read-back latency efficiently, and how to exploit spatial and temporal coherence of visibility. Guthe et al. [2006] apply a statistical model to describe occlusion probability of queries in order to reduce the number of wasted queries. HOQs are not only viable for on-line visibility determination, but have also been successfully used to speed up from-region visibility pre-computation [Leyvand et al., 2003] and complexity reduction of large meshes [Ernst et al., 2004].

Conditional rendering (OpenGL) [The Khronos Consortium, 2011] or occlusion predicates (DirectX) [Microsoft Corporation, 2010] avoid explicit synchronization. First, the (bounding) geometry is rasterized against an existing depth buffer to generate the predicates, and later predicated draw calls are issued: rendering is automatically omitted, if the query result is available and full occlusion has been detected, i.e., predicates are non-stalling and are therefore not guaranteed. This is perfectly suited for per-object occlusion culling.

Apart from hardware occlusion queries (HOQs) various other methods exist which determine visibility in image space in an output-sensitive way: Hierarchical representations of depth maps (similar to [Greene et al., 1993]) have been presented by [Décoret, 2005], which can be queried in GPU programs, but can also be used to cull geometry prior to draw submission. Hierarchical occlusion maps [Zhang et al., 1997] store opacity and not depth information which allows approximate visibility culling by using an opacity threshold. For this, occluders and occludees are distinguished, and the visibility test is decoupled into an overlap test (do objects overlap in screen space), and a depth test (are occluders closer to the viewer than occludees).

Occupancy maps [Staneker et al., 2003] aim at reducing the number of visibility queries: A low-resolution version of the frame buffer is stored as a bitmask to determine efficiently if an object is visible or possibly occluded (which is then verified with a standard HOQ).

CHAPTER 3

Radiometry

Light transport simulations follow physical principles to measure the distribution of radiative energy in space. The central quantity of radiometry studies is *radiance,* a density of radiative power over solid angle and area. This density remains constant along straight paths or rays (assuming vacuum) and thus lends itself ideally to *ray* or *geometric* optics. As radiometry is a study of energy distribution, it cannot encompass any phenomena only explainable within *wave optics* or even *quantum optics*, for instance diffraction and interference.

By the means of radiometry, it is possible to formulate the relation between incident and reflected energy at a surface from which *local reflection* properties and eventually the rendering equation [Kajiya and Von Herzen, 1984] are derived.

Not only can radiometry be used to study the reflective behaviour of surfaces, but it can also be used in studies of gaseous phenomena. To that end Chandrasekhar [1960] established *radiative transfer,* the theory of radiative energy distributions in participating media. Nowadays the theory has been adopted by computer graphics and is heavily applied in volume rendering [Hege et al., 1993] and light transport simulations for simulating participating media [Dutré et al., 2006; Pharr and Humphreys, 2010].

3.1 Radiometric Quantities

Radiation can be conceptualized by streams of photons travelling through space [Dutré et al., 2006]. Each photon carries an energy of

$$e_\nu = h\nu,$$

which solely depends on the photon frequency ν and Planck's constant h. A stream of n photons of frequency ν carries the total amount of radiant energy $Q_\nu = n_\nu e_\nu$. Taking only visible frequencies from $\nu_0 = 400THz$ to $\nu_1 = 790THz$ into account, the total amount of *radiant energy* can be computed.

$$Q = \int_{\nu_0}^{\nu_1} n_\nu h\nu d\nu.$$

Based on radiant energy, we can define *radiant flux* Φ as the total amount of energy that is registered within the unit time interval, i.e.

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

As radiation flows through a surface, we can measure the flux per unit area. This quantity is called *irradiance* if the flux is incident on the surface:

$$E = \frac{d\Phi_{in}}{dA}\left[\frac{W}{m^2}\right].$$

Similarly, radiation can also leave the surface, due to reflection or emission. This exitant flux is called *radiosity*

$$B = \frac{d\Phi_{out}}{dA}\left[\frac{W}{m^2}\right],$$

and has the same physical units as irradiance.

Just as flux can be defined over surface, it can also be defined over *solid angle*. The solid angle is conceptually the same as the radian measure in 2D, but extended into 3D space. It is measured in *steradians* (sr) and corresponds to the total area of the projection of an object onto the unit sphere. Therefore the maximum solid angle is 4π and the differential solid angle $d\omega$ is essentially a unit direction. The relation between solid angle and surface are is given by

$$d\omega = \frac{\cos\theta}{r^2}dA. \tag{3.1}$$

Here dA is the differential surface at a position $\mathbf{y}$ which is projected onto the unit sphere at a point $\mathbf{x}$ where the solid angle is measured (Figure 3.1). The distance between $\mathbf{x}$ and $\mathbf{y}$ is denoted r, and θ is the angle between the connection between $\mathbf{x}$ and $\mathbf{y}$ and the normal at $\mathbf{y}$.

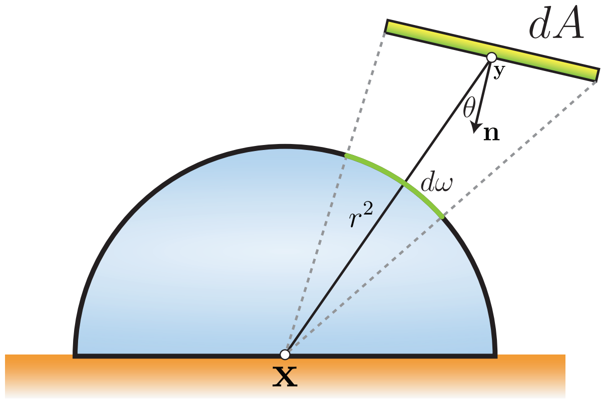


Figure 3.1: *The solid angle of a differential surface dA is its projection onto the unit sphere.*

Intensity is defined as the total amount of power that arrives at a point over solid angle:

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

The final and probably most important radiometric quantity for light transport simulations is *radiance*, defined as flux over projected area over solid angle

$$L = \frac{d^2\Phi}{cos\theta dAd\omega}\left[\frac{W}{m^2 \cdot sr}\right]$$

where θ is the angle between the surface's normal vector and the direction ω (Figure 3.2). From the definition of radiance, the relations to intensity, irradiance/radiosity and flux become immediately clear.

$$\begin{aligned} dE(\mathbf{x},\omega) &= L(\mathbf{x},\omega)\cos\theta d\omega \\ dI(\mathbf{x},\omega) &= L(\mathbf{x},\omega)\cos\theta dA \\ d\Phi(\mathbf{x},\omega) &= L(\mathbf{x},\omega)\cos\theta dAd\omega \end{aligned} \tag{3.2}$$

3.2 The Rendering Equation

The rendering equation is the governing equation of the radiative equilibrium distribution of spectral energy. It was first introduced by Kajiya [1986] to the field of computer graphics. We briefly summarize the results, starting with local reflection and extending the concept to light transport and global illumination.

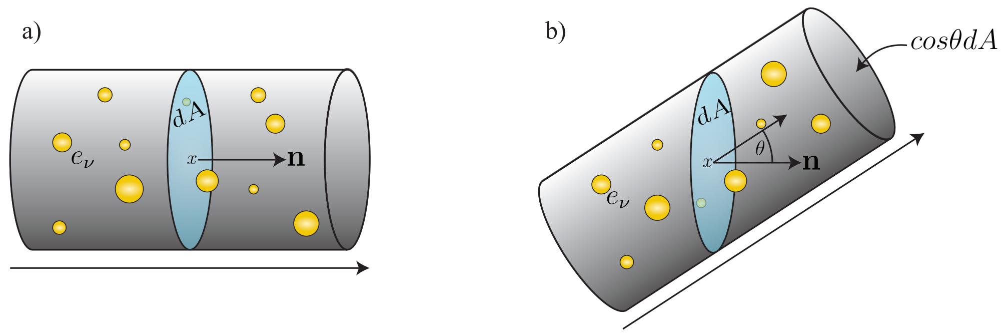


Figure 3.2: *Radiance is the energy density within a beam confined to a surface element dA with normal* **n**. *(a) At travel directions ω parallel to* **n**, *the beam cross-section is equivalent to dA. (b) At other angles, the beam cross-section is reduced by a factor of cos θ.*

3.2.1 Local reflection

The appearance of objects depends on two criteria, the incident illumination from all directions (also called light field), and the objects' material properties. Under the same incident illumination, different objects appear differently because of their reflective properties. This reflection behaviour under pre-set lighting conditions is the study of *local reflection*. Local reflection considers the behaviour of reflection on a macroscopic level. Instead of taking the underlying physical processes into account which leads to absorption and emission or scattering (microscopic level), the reflection model describes the amount of radiation reflected into a given direction given an incident radiance field.

In the most general case, light enters a surface from a direction ω_i at a surface point p_i, is scattered within the material and is outgoing into direction ω_o at a surface point p_o. This relation of incoming and outgoing radiance is described by the *bidirectional subsurface scattering reflectance distribution function (BSSRDF)* [Nicodemus et al., 1977]. For a huge class of materials, however, the simplifying assumption can be made that light is outgoing at the point of incidence, i.e. $p_i = p_o$. Then the function governing the ratio between incident and exitant radiation is called *bidirectional reflectance distribution function (BRDF)* and defined as the ratio of outgoing radiance and irradiance [Dutré et al., 2006]:

$$f_r(\mathbf{x}, \omega_i, \omega_o) = \frac{dL(\mathbf{x}, \omega_o)}{dE(\mathbf{x}, \omega_i)} = \frac{dL(\mathbf{x}, \omega_o)}{L(\mathbf{x}, \omega_i) \cos\theta \, d\omega_i}$$

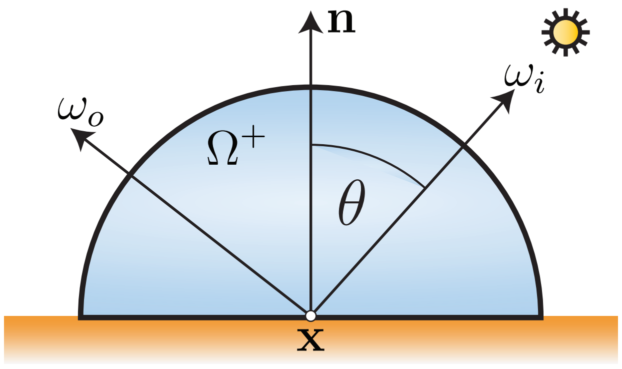


Figure 3.3: *The reflection integral (Equation 3.3) accumulates incident light arriving from direction ω_i in the upper hemisphere Ω^+ at the shading point* **x** *and computes radiance reflected into direction ω_o.*

Rearranging terms yields

$$dL(\mathbf{x}, \omega_o) = f_r(\mathbf{x}, \omega_i, \omega_o)L(\mathbf{x}\omega_i)\cos\theta\, d\omega_i$$

Physically-plausible BRDFs must obey three important properties [Dutré et al., 2006]: (1) Reciprocity, i.e. interchanging the incident and outgoing directions does not change the value of the BRDF. (2) the BRDF must be energy conserving, and (3) it must be linear with respect to all incident directions.

The latter property allows us to compute the total reflected radiance solving the *reflection integral* (Figure 3.3).

$$L(\mathbf{x}, \omega_o) = \int_{\Omega^+} f_r(\mathbf{x}, \omega_i, \omega_o)L(\mathbf{x}, \omega_i)\cos\theta\, d\omega_i, \tag{3.3}$$

where Ω^+ is the set of all directions in the upper hemisphere aligned with the surface normal at **x**, and $\cos\theta$ is the positive cosine of the angle between the normal and the direction ω_i.

3.2.2 The Rendering Equation

The rendering equation [Kajiya, 1986] states that light exitant from any surface point is the sum of its emitted and reflected radiance. Equation 3.3 already defines the reflected radiance and thus the rendering equation is

$$L(\mathbf{x}, \omega_o) = L_e(\mathbf{x}, \omega_o) + \underbrace{\int_{\Omega^+} f_r(\mathbf{x}, \omega_i, \omega_o)L(\mathbf{x}, \omega_i)\cos\theta\, d\omega_i}_{\text{local reflection}} \tag{3.4}$$

Integration over area: The rendering equation, or more importantly, the shading integral are defined as integration over solid angle. This formulation is not always convenient, for instance when computing direct illumination from area light sources, or in general, when gathering illumination from surface elements. In those cases it serves better to reformulate the shading integral as integration over surface area. Key to that is substituting the differential solid angle using Equation 3.1 (Figure 3.1).

$$d\omega = \frac{\cos^+\theta_{\mathbf{y}\to\mathbf{x}}}{||\mathbf{x}-\mathbf{y}||^2} d\mathbf{y}. \tag{3.5}$$

Here the $\cos^+\theta_{\mathbf{y}\to\mathbf{x}}$ is the positive cosine of the angle between the normal at **y** and the direction from **y** to **x**, i.e. it is non-zero, only if the normal at **y** faces **x**. Substituting then yields:

$$L(\mathbf{x}\leftarrow\mathbf{y}) = L_e(\mathbf{x}\leftarrow\mathbf{y}) + \int_A f_r(\mathbf{x}\leftarrow\mathbf{y}\leftarrow\mathbf{z}) G_v(\mathbf{y}\leftrightarrow\mathbf{z}) L(\mathbf{y}\leftarrow\mathbf{z}) d\mathbf{z} \tag{3.6}$$

Here the three point notation [Veach, 1997] has been used for brevity and to unmistakeably state the directional dependencies in this reformulation. The extended geometry term

$$G_v(\mathbf{y}\longleftrightarrow\mathbf{z}) = V(\mathbf{y}\longleftrightarrow\mathbf{z}) \underbrace{\frac{\cos^+\theta_{\mathbf{z}\to\mathbf{y}} \cos^+\theta_{\mathbf{y}\to\mathbf{z}}}{||\mathbf{y}-\mathbf{z}||^2}}_{=G(\mathbf{y}\longleftrightarrow\mathbf{z})}$$

arises from the substitution of the integration over solid angle, where visibility has to be taken into account explicitly. The visibility function $V(\mathbf{y}\longleftrightarrow\mathbf{z})$ evaluates to zero if the direct connection between the arguments is obstructed by a surface. Otherwise it is one.

Operator notation: As can be seen, the rendering equation is recursive in nature. This expansive notation often becomes cumbersome and Kajiya [1986] noted that is more convenient defining an integral *transport operator*

$$\langle \mathbf{T}L\rangle(\mathbf{x}, \omega_o) = \int_{\Omega^+} f_r(\mathbf{x}, \omega_o, \omega_i) L(\mathbf{x}, \omega_i) \cos\theta_i d\omega_i.$$

Then the rendering equation can compactly be written as: $L = L_e + \mathbf{T}L$. Recursively expanding the right-hand side substituting L with its definition, yields the well-known Neumann series

$$L = \sum_{i=0}^{\infty} \mathbf{T}^i L_e.$$

This equation states, that light $L(\mathbf{x}, \omega_o)$ is the sum of light emitted at $\mathbf{x}_0$, light that has been reflected once (direct illumination), twice (one bounce indirect illumination), and so on (i = 0,1,2,...). This formulation has been adopted to derive many rendering algorithms, such as path tracing [Kajiya, 1986], bidirectional path tracing [Lafortune, 1996], and instant radiosity Keller [1997]. The derivation of our screen space bias compensation (cf. Chapter 11) is also based on this formulation.

3.3 The Volumetric Rendering Equation

The rendering equation only considers light transport between surfaces in vacuum. In real scenarios, light (photons) also interacts with particles in the air. These, for instance, are small dust particles, water droplets, soot, or materials with high scattering behaviour (e.g. human skin). In general, this is called *participating medium*.

When photons travel through a medium, there are three possibilities how they interact with it [Chandrasekhar, 1960].

Absorption occurs when particles of the medium absorb all radiant energy of a colliding photon. Energy is typically transformed into other forms, such as heat, or re-emitted into the medium (with different wavelength).

Scattering occurs when a photon is deflected from its travelling direction ω after colliding with a particle of the medium (out-scattering), and travelling along a new direction ω' (in-scattering).

Emission is the result of thermal processes within the medium, for instance in fire. Photons are emitted into the medium.

All those effects are brought together in the general *equation of transport*, which states [Cerezo et al., 2005]:

$$\frac{dL(\mathbf{x},\omega)}{ds} = \underbrace{-\sigma_t(\mathbf{x})L(\mathbf{x},\omega)}_{\text{extinction}} + \underbrace{\sigma_a(\mathbf{x})L_e(\mathbf{x},\omega)}_{\text{emission}} + \underbrace{\sigma_s(\mathbf{x}) \int_\Omega f_p(\mathbf{x},\omega,\omega')L(\mathbf{x},\omega')d\omega'}_{\text{in-scattering}}.$$

Light travelling in direction ω is removed from the straight path due to absorption (σ_a) and out-scattering (σ_s). Both effects account for extinction and are expressed in the extinction coefficient $\sigma_s(\mathbf{x}) = \sigma_a(\mathbf{x}) + \sigma_s(\mathbf{x})$.

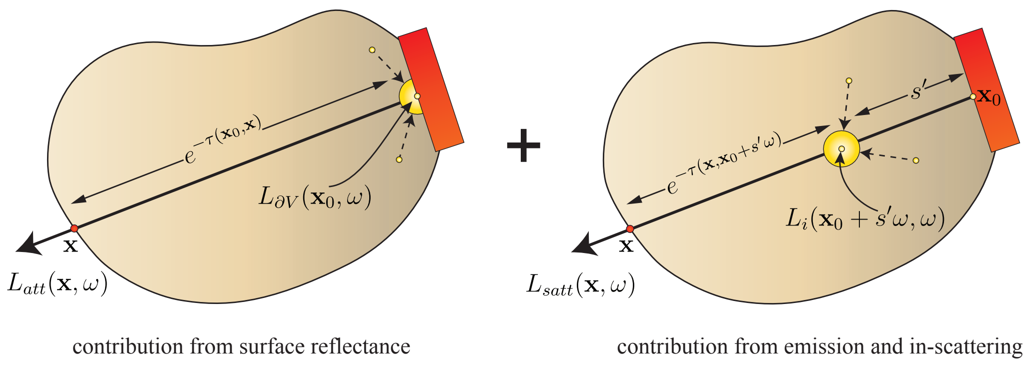

$e^{-\tau(\mathbf{x}_0,\mathbf{x})}$
$L_{\partial V}(\mathbf{x}_0,\omega)$
$\mathbf{x}$
$L_{att}(\mathbf{x},\omega)$
contribution from surface reflectance
+
s'
$\mathbf{x}_0$
$e^{-\tau(\mathbf{x},\mathbf{x}_0+s'\omega)}$
$L_i(\mathbf{x}_0+s'\omega,\omega)$
$\mathbf{x}$
$L_{satt}(\mathbf{x},\omega)$
contribution from emission and in-scattering

Figure 3.4: *The volumetric rendering equation is the sum of attenuated surface emission and reflectance and the integral over light emitted and scattered within the volume.*

Furthermore, radiative energy is added through emission and in-scattering. This is combined in the so called *source term*

$$L_i(\mathbf{x},\omega) = \sigma_a(\mathbf{x})L_e(\mathbf{x},\omega) + \sigma_s(\mathbf{x})\int_\Omega f_p(\mathbf{x},\omega,\omega')L(\mathbf{x},\omega')d\omega'. \tag{3.7}$$

The term $f_p(\mathbf{x},\omega,\omega')$ is called phase function and is the probability of a photon being scattered at $\mathbf{x}$ from direction ω into direction ω' [Pharr and Humphreys, 2010]. The phase function is a probability distribution, i.e. it is normalized:

$$\int_\Omega f_p(\mathbf{x},\omega,\omega')d\omega' = 1.$$

Integral formulation: The equation of transfer is an integro-differential equation and has an integral form counterpart, which makes it suitable for rendering algorithms.

In the integral form of the equation of transport, the new quantity *optical depth* is introduced. The optical depth is defined between two points $\mathbf{x},\mathbf{y}$, which lie on a straight path with direction ω, i.e. $\mathbf{y} = \mathbf{x} + s\omega$.

$$\delta(\mathbf{x},\mathbf{y}) = \int_0^s \sigma_t(\mathbf{x} + s'\omega)ds'.$$

Based on optical depth, the transmittance

$$\tau(\mathbf{x},\mathbf{y}) = \exp\big(-\delta(\mathbf{x},\mathbf{y})\big)$$

is defined as the fraction of light that is not attenuated travelling from **x** to **y**. Then the equation of transfer can be reformulated, and radiance $L(\mathbf{x},\omega)$ leaving a point **x** into direction ω becomes (Figure 3.4):

$$L(\mathbf{x},\omega) = \underbrace{L_{\partial V}(\mathbf{x}_0,\omega)\tau(\mathbf{x}_0,\mathbf{x})}_{=L_{\text{att}}(\mathbf{x},\omega)} + \int_0^{\|\mathbf{x}-\mathbf{x}_0\|} \underbrace{L_i(\mathbf{x}_0+s'\omega,\omega)\tau(\mathbf{x}_0,\mathbf{x}_0+s'\omega)}_{L_{\text{scatt}}(\mathbf{x},\omega)}\,ds'. \tag{3.8}$$

$L_{\partial V}(\mathbf{x}_0,\omega)$ is the light that enters the medium at its boundary, and $L_i(\mathbf{x}_0 + s'\omega,\omega)$ in-scattered and emitted light described by the source term (Equation 3.7). Boundaries typically are the scene surfaces where light is reflected into the medium. Hence it is computed using the rendering equation,

$$L_{\partial V}(\mathbf{x}_0,\omega) = L_e(\mathbf{x}_0,\omega) + \int_\Omega f_r(\mathbf{x},\omega_i,\omega)L(\mathbf{x}_0,-\omega_i)\cos\theta\, d\omega_i.$$

Light incident on a surface is computed by the volumetric rendering equation (Equation 3.8). This is why the direction incident direction ω_i is negated.

CHAPTER 4

Global Illumination Algorithms

GLOBAL illumination (GI) has been an active research area since the beginning of the 1980s. In the early years, global illumination and photorealism did not go farther than computing direct illumination plus accounting for perfect specular reflections and refractions [Whitted, 1980]. In 1984 the radiosity algorithm, based on finite element methods, has been applied to light transport for purely diffuse surfaces [Goral et al., 1984]. At the same time, the first stochastic methods for handling soft shadows, motion blur, and depth of field started to appear [Cook et al., 1984]. Two years later, Kajiya [1986] introduced the first path tracing algorithm and the rendering equation, which ever since has been the formal foundation for all global illumination algorithms. In the following decade, research heavily focused on improvements and extensions of radiosity algorithms, until the mid 1990s, when particle tracing [Jensen, 1996; Keller, 1997] and Monte Carlo path tracing algorithms [Lafortune, 1996; Veach, 1997] gained momentum. With the ever increasing computational power and programmable flexibility of GPUs, the early 2000s ushered the era of interactive GI algorithms. In contrast to techniques from the 1990s, which were developed to cover the full spectrum of light transport, interactive GI algorithms are either tailored to specific problems or limited in the kind of phenomena they can compute. Those restrictions help to reduce the complexity of light transport, which is mostly a necessary evil to maintain interactivity.

In this chapter, we briefly summarize the core ideas and principles of traditional global illumination techniques, followed by a section on global illumination in participating media, and a survey on interactive global illumination algorithms.

4.1 Radiosity

Radiosity is a finite element method that is inspired by heat transport simulations and was introduced to computer graphics by Goral et al. [1984]. The core of the algorithm is to discretize the scene's surfaces into finite elements, also referred to as surface patches in classic radiosity. For each surface patch, radiosity is assumed to be constant. This can also be interpreted as a projection of the radiosity function onto a piecewise-constant basis. In this spirit, other basis functions, such as Wavelets [Gortler et al., 1993] and polynomial basis functions [Zatz, 1993] have been explored. Then light transport can be formulated as system of linear equations, in which each equation

$$B_i = B_{e,i} + \rho_i \sum_{j=1}^{N} F_{ij} B_j,$$

computes the radiosity of the i-th surface patch [Sillion and Puech, 1994]. Here ρ_i denotes the patch albedo and $B_{e,i}$ the emitted radiosity. The form factor

$$F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{V(\mathbf{x}_i, \mathbf{x}_j) \cos\theta_{\mathbf{x}_i} \cos\theta_{\mathbf{x}_j}}{\pi \|\mathbf{x}_i - \mathbf{x}_j\|^2} d\mathbf{x}_i d\mathbf{x}_j$$

encodes the fraction of radiosity of B_j that contributes to the reflected radiosity at the i-th patch. The linear system of equations can be solved by using numerical methods, such as Gauss-Seidel iteration [Sillion and Puech, 1994], Southwell relaxation, or progressive radiosity [Cohen et al., 1988].

Since its conception, research heavily focused on solving the pressing issues of radiosity. Form factor computation is the most time consuming operation involving numerous visibility queries, especially as the number of form factors grows quadratically with the number of surface patches. To alleviate this complexity, Hanrahan et al. [1991] proposed hierarchical radiosity, significantly reducing the number of form factor computations. Other improvements include lifting the restriction to purely diffuse surface reflectance [Immel et al., 1986] and improving quality [Heckbert, 1992; Lischinski et al., 1993]. As GPUs became more powerful and flexible, the first GPU-based radiosity solvers started to appear. While Coombe et al. [2004] ignored visibility, the later proposed Antiradiance method [Dachsbacher et al., 2007] shows that occlusion can be solved by propagating negative light. Besides these selected examples, radiosity methods proved to be fertile research ground, which brought many more extensions to life. Excellent introductions to radiosity can be found in dedicated books [Cohen et al., 1993; Sillion and Puech, 1994].

4.2 Monte-Carlo Integration

Monte-Carlo simulations are a class of stochastic techniques that makes it possible to estimate the solution of problems which are not analytical solvable. As such they are an invaluable tool for global illumination algorithms because they make it possible to estimate the solution of the reflection integral (cf. Chapter 3.2). In general, an integral can be estimated by a Monte-Carlo estimator of the form

$$\int_{\mathscr{D}} f(x)dx \approx \frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)}{p(x_i)},$$

where the values x_i are randomly picked from the integration domain $\mathscr{D}$ with probability $p(x_i)$. This estimator is said to be *unbiased* if the expected value of the error

$$E\left[\frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)}{p(x_i)} - Q\right] = 0,$$

for all $N \geq 1$, where Q is the value of the integral to be estimated. Following the law of large numbers ($N \to \infty$), the estimator becomes the expected value, which in turn is the solution to the integral:

$$\begin{aligned}\lim_{N\to\infty}\frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)}{p(x_i)} &= E\left[\frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)}{p(x_i)}\right]\\ &= \int_{\mathscr{D}}\frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)}{p(x_i)}p(x_i)dx\\ &= \int_{\mathscr{D}} f(x_i)dx.\end{aligned}$$

Due to being a stochastic method, the estimator deviates from the expected value for a finite number of samples, and the root-mean-square error (RMSE) reduces with a convergence rate of $O(N^{-\frac{1}{2}})$, regardless of the dimension of $\mathscr{D}$.

To improve the estimation, Monte-Carlo methods rely on variance reduction techniques. The most important technique in photorealistic image synthesis is importance sampling. Hereby a probability density function (PDF) is chosen that matches the integrand as closely as possible, and thereby samples are drawn preferably where the integrand has a significant contribution. Ideally the PDF is just the scaled integrand, i.e. $p(x) \sim cf(x)$, but

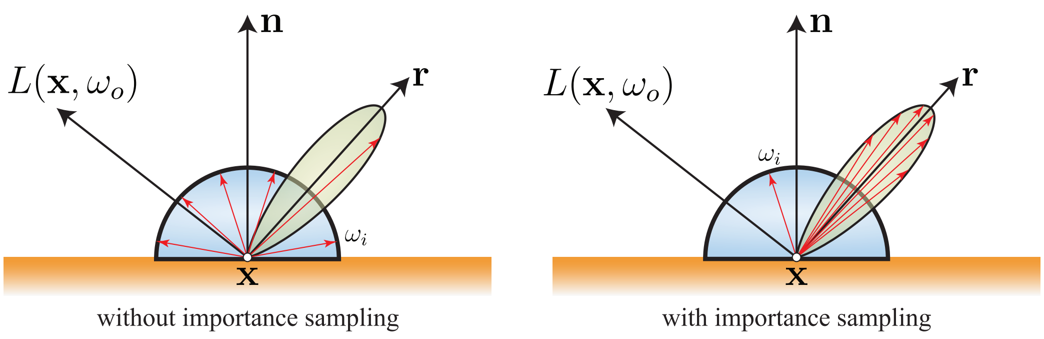

n
$L(\mathbf{x}, \omega_o)$
r
ω_i
x
without importance sampling
n
$L(\mathbf{x}, \omega_o)$
r
ω_i
x
with importance sampling

Figure 4.1: *Monte-Carlo integration of the reflection integral with glossy materials. Incident directions ω_i close to the perfect reflection* **r** *have a high contribution.* ***Left:*** *Uniform sampling considers all direction equally.* ***Right:*** *Importance sampling prefers directions for which the BRDF has higher contributions.*

this would make Monte-Carlo integration superfluous because the value of the integral would already be known ($c^{-1} = \int f(x)dx$). A common application of importance sampling is to distribute directions according to the BRDF when computing the estimator of the shading integral (Figure 4.1):

$$L(\mathbf{x}, \omega_o) \approx \frac{1}{N} \sum_{i=1}^{N} \frac{f_r(\mathbf{x}, \omega_o, \omega_i) \cos\theta_i}{p(\omega_i)} L(\mathbf{x}, \omega_i).$$

Here $p(\omega_i)$ is the probability of sampling the direction ω_i of incident light $L(\mathbf{x}, \omega_i)$ that contributes to the estimation, and $\cos\theta_i$ is the cosine of the angle between the normal at $\mathbf{x}$ and the incident direction ω_i.

Incident directions ω_i may be distributed uniformly, but make the estimation prone to high variance if this strategy is applied to glossy surfaces. This is because glossy surfaces reflect light over a narrow solid angle, and uniform sampling takes incident illumination mostly from directions with low contribution to the overall estimation. The variance can substantially be reduced by sampling directions for which the BRDF has a significant contribution to the overall estimation. A unified scheme for importance sampling BRDFs does not exist because it highly depends on the BRDF itself. For example, diffuse surfaces benefit from sampling according to the cosine of the incident angle [Pharr and Humphreys, 2010]. Importance sampling the Phong BRDF [Phong, 1975; Pharr and Humphreys, 2010] or Lafortune's model [Lafortune et al., 1997] considers the diffuse term and the specular term separately and combines results afterwards. The Ward model [Ward, 1992] applies a similar strategy, but uses exponential functions for the specular term. For the well-known Cook-Torrance BRDF [Cook and Torrance, 1982] analytical sampling does not exist, but factored representations can be used [Lawrence et al., 2004].

Unfortunately, not in all cases light is incident from directions for which the BRDF has high contribution. Ideally, samples are distributed according to the full integrand, i.e.

$$p(\omega_i) \sim c f_r(\mathbf{x}, \omega_o, \omega_i) L(\mathbf{x}, \omega_i) \cos\theta_i,$$

but this would require the solution to the integral (see above). Numerical approaches, however, are possible. On the one hand, Jensen [1995] used photon mapping to sample incident illumination and constructed a piecewise-constant numerical probability density function to approximate sampling according to the full integrand (see above) for path tracing, but his numerical approach is computational prohibitive. Veach [1997], on the other hand, computed two estimators using different importance sampling strategies and combined them optimally using multiple importance sampling.

Besides importance sampling, a large repertoire of other variance reduction techniques exist, such as control variates, stratified sampling, and low-discrepancy sampling among others [Veach, 1997].

4.3 Path Integrals and Path Tracing

Path integrals are another way of describing light transport and are closely related to the Neumann series (cf. Chapter 3.2). Often, this way is preferred because path integrals, as Veach [1997] points out, provide a global view onto the integration problem, are intuitive, and make integration techniques in the literature directly applicable. Key to the path integral reformulation is the infinite dimensional *path space*. A path $\overline{\mathbf{x}}_n$ is a concatenation of vertices $\mathbf{x_0 x_1 ... x_n}$ where $\mathbf{x}_0$ lies on the image plane and $\mathbf{x}_n$ lies on a light source. All vertices in-between lie on scene surfaces. The path is said to have length n, counting the edges required to connect pairs of successive path vertices. For instance, a path of length 2 consists of a vertex on the image plane, one on a scene surface and one on a light source. All paths of length n form the set $\mathscr{P}_n$, and the union of all $\mathscr{P}_n$ forms the path space.

Then light transport is formally expressed by the *measurement equation* that computes the response I_j of the j-th (pixel-)sensor to light carrying paths of all length (Figure 4.2):

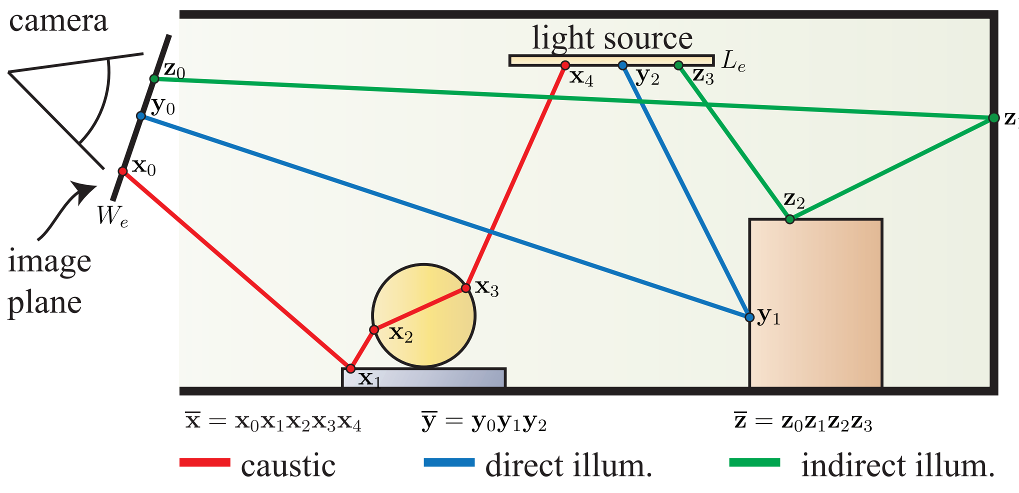


Figure 4.2: *Transport paths of different length. Paths that account for direct illumination ($\overline{\mathbf{y}}$) have length 2. One bounce indirect illumination stems from paths of lengths 3 ($\overline{\mathbf{z}}$). Any GI phenomena can be expressed within the path integral, e.g. caustics ($\overline{\mathbf{x}}$).*

$$I_j = \sum_{k=1}^{\infty} \int_{\mathscr{P}_k} W_e^j(\mathbf{x}_0 \leftarrow \mathbf{x}_1) G(\mathbf{x}_0 \leftrightarrow \mathbf{x}_1) V(\mathbf{x}_0 \leftrightarrow \mathbf{x}_1) \mathbf{T}(\overline{\mathbf{x}}_k) L_e(\mathbf{x}_k \rightarrow \mathbf{x}_{k-1}) d\mu(\overline{\mathbf{x}}_k) \tag{4.1}$$

Here $d\mu(\overline{\mathbf{x}}_k) = d\mathbf{x}_0 d\mathbf{x}_1 \cdots d\mathbf{x}_k$ is the *path measure*, and $W_e^j(\mathbf{x}_0 \leftarrow \mathbf{x}_1)$ the response function of the j-th (pixel-)sensor. Essentially, this is a pixel filter, e.g. a box filter, that accumulates all paths whose endpoints $\mathbf{x}_0$ lie within the boundaries of the pixel. The geometry term is defined as (cf. Chapter 3.2):

$$G(\mathbf{x}_j \leftrightarrow \mathbf{x}_{j+1}) = \frac{C(\mathbf{x}_j \rightarrow \mathbf{x}_{j+1}) C(\mathbf{x}_{j+1} \rightarrow \mathbf{x}_j)}{\|\mathbf{x}_j - \mathbf{x}_{j+1}\|^2},$$

with

$$C(\mathbf{x} \rightarrow \mathbf{y}) = \cos^+ \theta_{\mathbf{x} \rightarrow \mathbf{y}} = \max\left(0, \mathrm{dot}\left(\mathbf{n}(\mathbf{x}), \frac{\mathbf{y} - \mathbf{x}}{\|\mathbf{y} - \mathbf{x}\|}\right)\right),$$

where $\mathbf{n}(\mathbf{x})$ is the normal at $\mathbf{x}$. The binary visibility function $V(\mathbf{x}_j \leftrightarrow \mathbf{x}_{j+1})$ returns 1 if both points are mutually visible and zero otherwise. The *path throughput*

$$\mathbf{T}(\overline{\mathbf{x}}_i) = \prod_{j=1}^{i-1} f_r(\mathbf{x}_{j-1} \leftarrow \mathbf{x}_j \leftarrow \mathbf{x}_{j+1}) V(\mathbf{x}_j \leftrightarrow \mathbf{x}_{j+1}) G(\mathbf{x}_{j-1} \leftrightarrow \mathbf{x}_j)$$

is the fraction of emitted light $L_e(\mathbf{x}_k)$ that reaches the sensor at $\mathbf{x}_0$.

Random walks: Despite the complexity of the equation, its estimator can be easily computed. The only challenge that has to be overcome is how to sample random paths. The typical approach is to construct paths incrementally adding new path segments exploiting random walks. A random walk in its simplest form traces a path starting from the image plane at $\mathbf{x}_0$. Shooting a ray into the scene, a path $\overline{\mathbf{x}}_1$ of length 1 is constructed with the closest intersection point $\mathbf{x}_1$. From there on, path tracing incrementally adds new path segments by casting a new ray from the last vertex $\mathbf{x}_n$ found into a random direction ω_n in order to find the next path vertex $\mathbf{x}_{n+1} = h(\mathbf{x}_n, \omega_n)$ using the ray-casting function $h(.)$. With each additional path segment, the throughput changes as follows [Veach, 1997]:

$$\mathbf{T}(\overline{\mathbf{x}}_{n+1}) = \frac{f_r(\mathbf{x}_{n-1}\leftarrow\mathbf{x}_n\leftarrow\mathbf{x}_{n+1})C(\mathbf{x}_n\rightarrow\mathbf{x}_{n+1})}{p(\omega_n)}\mathbf{T}(\overline{\mathbf{x}}_n). \tag{4.2}$$

Path tracing terminates when a light source is hit. Alternatively, transported light can be computed at each newly added path vertex $\mathbf{x}_{n+1}$ connecting it to the light sources thereby accounting for each path length. As the random walk would continue infinitely, it must be terminated using Russian roulette [Arvo and Kirk, 1990].

4.3.1 Extension to Participating Media

Pauly et al. [2000] opted for a generalized form of the path integral to extend Metropolis Light Transport to participating media. The general form of Equation 4.1 remains, however, generalized variants of the geometry term, the visibility function, and the scattering function are introduced to account for the fact that path vertices either reside in the volume or on a surface. The measurement equation (Equation 4.1) is then redefined:

$$I_j = \sum_{k=1}^{\infty}\int_{\mathscr{P}_k} W_e^j(\mathbf{x}_0\leftarrow\mathbf{x}_1)G^*(\mathbf{x}_0\leftrightarrow\mathbf{x}_1)V^*(\mathbf{x}_0\leftrightarrow\mathbf{x}_1)\mathbf{T}^*(\overline{\mathbf{x}}_k)L_e^*(\mathbf{x}_k\rightarrow\mathbf{x}_{k-1})d\mu(\overline{\mathbf{x}}_k). \tag{4.3}$$

Light can be emitted from surfaces or the medium where it is partially absorbed (cf. Chapter 3).

$$L_e^*(\mathbf{x}_k\rightarrow\mathbf{x}_{k-1}) = \begin{cases} L_e(\mathbf{x}_k\rightarrow\mathbf{x}_{k-1}) & \mathbf{x}_k \text{ on surface} \\ \sigma_a(\mathbf{x}_k)L_e(\mathbf{x}_k\rightarrow\mathbf{x}_{k-1}) & \mathbf{x}_k \text{ in volume} \end{cases}$$

Visibility in the presence of participating media is non-binary because it additionally accounts for transmittance, i.e. $V^*(\mathbf{x}\leftrightarrow\mathbf{y}) = \tau(\mathbf{x}\leftrightarrow\mathbf{y})V(\mathbf{x}\leftrightarrow\mathbf{y})$.

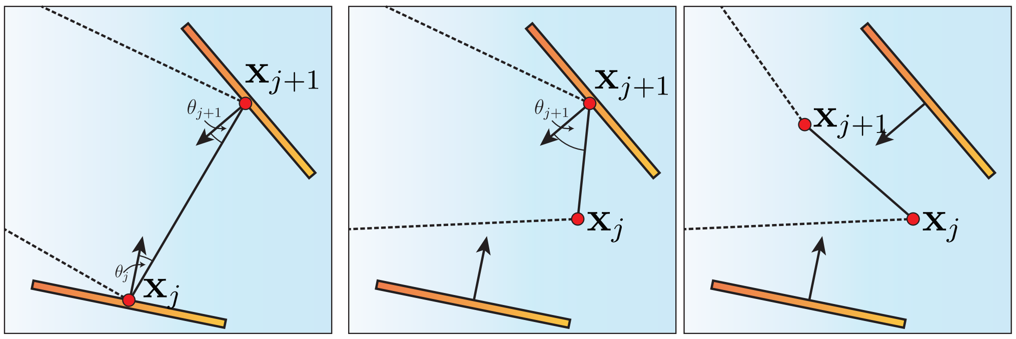


Figure 4.3: *Three configurations for two subsequent path vertices exist. Both vertices lie on surfaces (left), one of both vertices lies in the volume (middle), the other on the surface, or both vertices are in the volume (right). For volume vertices, the phase function and scattering coefficient contributes to the path throughput, for surface points the BRDF is evaluated.*

The generalized geometry term

$$G^*(\mathbf{x}_j \leftrightarrow \mathbf{x}_{j+1}) = \frac{C^*(\mathbf{x}_j \rightarrow \mathbf{x}_{j+1}) C^*(\mathbf{x}_{j+1} \rightarrow \mathbf{x}_j)}{\|\mathbf{x}_j - \mathbf{x}_{j+1}\|^2}$$

replaces the cosine functions with their generalized counterpart. If a path vertex lies in the volume, a normal, and therefore a cosine, does not exist, i.e.

$$C^*(\mathbf{x}_j \rightarrow \mathbf{x}_{j+1}) = \begin{cases} \cos^+ \theta_j & \mathbf{x}_j \text{ on surface} \\ 1 & \mathbf{x}_j \text{ in volume} \end{cases} .$$

The generalized throughput $\mathbf{T}^*(\overline{\mathbf{x}}_i)$ combines all generalized terms and is the fraction of emitted light that reaches the sensor at $\mathbf{x}_0$.

$$\mathbf{T}^*(\overline{\mathbf{x}}_i) = \prod_{j=1}^{i-1} f^*(\mathbf{x}_{j-1} \leftarrow \mathbf{x}_j \leftarrow \mathbf{x}_{j+1}) V^*(\mathbf{x}_j \leftrightarrow \mathbf{x}_{j+1}) G^*(\mathbf{x}_j \leftrightarrow \mathbf{x}_{j+1}),$$

with the generalized scattering function

$$f^*(\mathbf{x}_{j-1} \leftarrow \mathbf{x}_j \leftarrow \mathbf{x}_{j+1}) = \begin{cases} f_r(\mathbf{x}_{j-1} \leftarrow \mathbf{x}_j \leftarrow \mathbf{x}_{j+1}) & \mathbf{x}_j \text{ on surface} \\ \sigma_s(\mathbf{x}_j) f_p(\mathbf{x}_{j-1} \leftarrow \mathbf{x}_j \leftarrow \mathbf{x}_{j+1}) & \mathbf{x}_j \text{ in volume} \end{cases}$$

which evaluates the BRDF in case $\mathbf{x}_j$ lies on a surface, and otherwise accounts for in-scattering due to the phase function and the scattering coefficient. This differentiation leads to three light transport configurations for two subsequent path vertices (Figure 4.3). Light is either exchanged between two surfaces, between the medium and the surface, or only within the volume.

Random walks: Random walks have to account for path vertices in the volume where the throughput is updated as follows [Pauly et al., 2000]:

$$\mathbf{T}(\overline{\mathbf{x}}_{n+1}) = \tau(\mathbf{x}_n, \mathbf{x}_{n+1}) \frac{f^*(\mathbf{x}_{n-1} \leftarrow \mathbf{x}_n \leftarrow \mathbf{x}_{n+1}) C^*(\mathbf{x}_n \rightarrow \mathbf{x}_{n+1})}{p(\omega_n) p(s)} \mathbf{T}(\overline{\mathbf{x}}_n), \qquad (4.4)$$

with $\mathbf{x}_{n+1} = \mathbf{x}_n + s\omega_n$, $s \leq \|\mathbf{x}_n - h(\mathbf{x}_n, \omega_n)\|$. The distance s is smaller or equal to the distance of the closest surface in direction ω_n and chosen using free path sampling with probability $p(s)$. Free path sampling is used because the next path vertex may reside in the medium. For homogeneous media exists a simple analytic formula [Lafortune and Willems, 1996; Raab et al., 2008]:

$$s = \frac{-\ln(1-\xi)}{\sigma_t},$$

with the according probability density function $p(s) = \sigma_t \exp(-\sigma_t s)$. For media of non-uniform density exists only an implicit equation [Raab et al., 2008] that cannot be applied directly. Instead Woodcock tracking, a rejection sampling technique, can be used [Coleman, 1968; Raab et al., 2008; Yue et al., 2010]. The next path direction ω_n is determined importance sampling the phase function or the BRDF, respectively, depending on the location of $\mathbf{x}_n$.

Algorithms: The first path tracing algorithm was proposed by Kajiya [1986]. Veach [1997] developed the path integral, a robust mathematical framework in which many global illumination methods can be concisely expressed. In particular, photon mapping [Jensen, 1996] and instant radiosity [Keller, 1997] can be formulated as integration over paths [Pharr and Humphreys, 2010].

Based on the path integral formulation, Veach [1997] developed bidirectional path tracing (BPT), at the same time as Lafortune [1996]. BPT creates subpaths through random walks from the light sources and the camera. Full transport paths are then created connecting the sub-paths using multiple importance sampling [Veach, 1997]. The Metropolis-Hastings algorithm is a Markov chain Monte-Carlo technique that has been used by Veach [1997] to create Metropolis Light Transport (MLT). MLT uses Markov chains to create new path samples mutating existing paths according to different mutation strategies. Those mutations are then either accepted or rejected which helps faster conversion in difficult illumination scenarios, such as caustics. Both, BPT and MLT, have later been extended to participating media [Lafortune and Willems, 1996; Pauly et al., 2000].

4.4 Photon Mapping

Photon mapping [Jensen, 1996] is a particle tracing algorithm, but despite the name, those particles have nothing in common with photons as they are understood in physics. Photon mapping is a two-pass method. First it traces a set of light carrying trajectories through the scene, thereby computing a sparsely sampled distribution of illumination incident on the scene surfaces. Second, the incident illumination is reconstructed using reconstruction filters at surface points visible from the camera while evaluating the shading integral numerically. Optionally, this step is replaced by final gathering (see below).

Photon shooting: Particles with an initial power spectrum Φ_0 are emitted from the light sources, and their starting positions and directions in which to trace the particles are sampled. Using ray tracing, the first visible surface point on which a particle impinges is determined. There a record of the incident illumination is created and stored in the so called photon map. Using Russian roulette [Arvo and Kirk, 1990], it is decided whether a particle shall be absorbed or scattered into a new direction. When scattered, the power spectrum of the scattered particle Φ_{i+1} depends on the power spectrum of the incident particle Φ_i as follows:

$$\Phi_{i+1} = \frac{f_r(\mathbf{x}, \omega_i, \omega_o)\cos\theta_o}{p(\omega_o)}\Phi_i.$$

Density estimation: In the second pass, photon mapping computes the illumination at the surface points visible to the camera. This process, called density estimation, is essentially a reconstruction of the incident illumination from the sparsely sampled illumination. Light reflected towards the camera from a visible surface point $\mathbf{x}$ is approximated by

$$L(\mathbf{x}, \omega_o) \approx \sum_{i=0}^{N} f_r(\mathbf{x}, \omega_o, \omega_i)\kappa(\mathbf{x}, \mathbf{x}'_i)\Phi_p(\mathbf{x}'_i, \omega_i), \tag{4.5}$$

where κ is the normalized reconstruction filter and the Φ_p is the particle's spectral power that is incident at $\mathbf{x}'_i$ from direction ω_i. N is the total number of the photons in the photon map. Because the reconstruction filter usually has limited support, it suffices to gather only those photons for which the filter does not evaluate to zero.

In contrast to path tracing [Kajiya, 1986], this approach is *biased*, but *consistent*. The bias occurs because blurring is introduced by the reconstruction filter κ. With an infinite number of photons ($N \to \infty$), the filter's support can be arbitrarily small, and thus it is said that the method is consistent [Jensen, 1996].

Photon mapping works exceptionally well for caustics that appear on diffuse surfaces by focusing light through reflective and refractive objects. In caustic regions, the photon density is sufficiently high, and reconstruction yields acceptable results. If this focusing does not occur because of multiple subsequent diffuse or moderately glossy bounces, the photon density at surface points is in general not sufficient. Then the approximation of the reflection integral exhibits visible artefacts perceived as low frequency noise. To counter this issue without introducing new undesirable artefacts, final gathering can be used.

Final gathering: Final gathering calculates the amount of indirect illumination at a surface point and is typically used to improve the quality of approximate light transport methods, such as photon mapping. Typically final gathering is computed using Monte-Carlo integration (cf. Section 4.2) and ray tracing to sample the incident illumination. Wherever a sample ray intersects a surface, light reflected towards the integration point is computed using Equation 4.5. For satisfactory results, often hundreds of sample rays must be processed at each gather point. This makes final gathering the most time consuming part of photon mapping.

Not only photon mapping benefits from final gathering. Radiosity solutions suffer from distracting discontinuities in the illumination, and final gathering can also be used to improve quality [Lischinski et al., 1993].

Ray tracing is not the only algorithm that can be used to determine the surface from which to gather light. Christensen [2008] used a CPU-based method to speed up final gathering for diffuse and moderately glossy scenes using a point-based representation of direct illumination stored in an octree. At each gathering location, distant points are rasterized into a cube map and nearby points are raycast.

Building on this idea, we present *micro-rendering* in Chapter 8 that enables high quality final gathering on the GPU in interactive framerates.

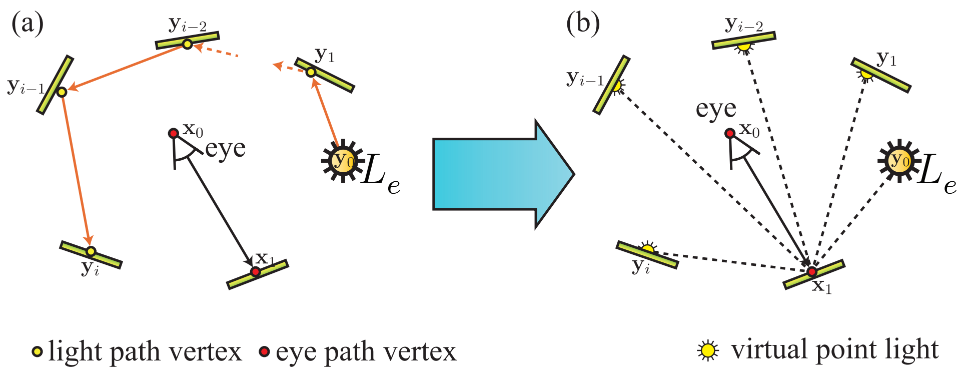


Figure 4.4: *(a) Instant radiosity traces particle trajectories and creates a VPL at each impinging location (b). Direct illumination with VPLs and primary light sources (L_e) yields full global illumination with multiple bounces of indirect illumination.*

4.5 Virtual Point Light Methods

The general idea of virtual point light methods is to represent the equilibrium distribution of radiative energy in the scene by a precomputed, sparse set of virtual point lights (VPLs). The first algorithm that exploited this idea was instant radiosity [Keller, 1997]. VPL generation is similar to photon mapping. They are created using a particle tracing algorithm. Particles are emitted from the light sources and traced through the scene, which essentially constructs a light path incrementally (each bounce of the particle creates a new path segment). At each vertex of the light path, a VPL is created (Figure 4.4). Ultimately, computing direct illumination from all VPLs and primary light sources yields full global illumination (Figure 4.4). A derivation of this approach and details for practical applications that involve generating VPLs will be described in Chapter 10.

4.5.1 Rendering with VPLs

Computing indirect illumination with VPLs is simply approximated by

$$L_r(\mathbf{x}_0 \leftarrow \mathbf{x}_1) = \sum_{i=0}^{k} f_r(\mathbf{x}_0 \leftarrow \mathbf{x}_1 \leftarrow \mathbf{y}_i) V(\mathbf{x}_1 \leftrightarrow \mathbf{y}_i) G(\mathbf{x}_1 \leftrightarrow \mathbf{y}_i) \hat{L}_i(\mathbf{x}_1 \leftarrow \mathbf{y}_i) \qquad (4.6)$$

Here the $\mathbf{y}_i$ denote the positions of the VPLs. This illumination step, as already demonstrated by Keller [1997], can be accelerated exceptionally well using commodity GPUs.

VPLs concentrate all energy into one single point which leads to infinite variance in the illumination. This is caused by the singularity in the geometry term

$$G(\mathbf{x}\leftrightarrow\mathbf{y}_i) = \frac{C(\mathbf{x}\rightarrow\mathbf{y}_i)C(\mathbf{y}_i\rightarrow\mathbf{x})}{||\mathbf{x}-\mathbf{y}_i||^2}$$

if the shading point $\mathbf{x}$ and the VPL at $\mathbf{y}_i$ are very close. In rendered images, this singularity causes exceptionally bright splotches. To remedy this situation, an artificial bound b on the geometry term is introduced, i.e. $G' = \min(G, b)$, and the *bounded geometry term* is used instead. Despite suppressing artefacts, bounding introduces new problems, for it removes energy from short distance illumination. This becomes visible in artificial darkening of creases, edges, and corners.

Fortunately, there exists a closed form expression for the lost short distance energy [Kollig and Keller, 2006]:

$$L_r'(\mathbf{x},\omega_o) = \int_{\Omega^+} \frac{\max(G(\mathbf{x},h(\mathbf{x},\omega_i)) - b, 0)}{G(\mathbf{x},h(\mathbf{x},\omega_i))} f_r(\mathbf{x},\omega_o,\omega_i)L(\mathbf{x},\omega_i)\cos\theta_i d\omega_i. \tag{4.7}$$

G is the unbounded geometry term, b the bound and $h(\mathbf{x},\omega_i)$ the ray casting function returning the first visible surface point seen at $\mathbf{x}$ in direction ω_i. An *unbiased* estimate of full global illumination then becomes the sum of the biased global illumination with VPLs (including direct illumination) and the compensation term (Equation 4.7), i.e.

$$L(\mathbf{x},\omega) = \underbrace{L_d(\mathbf{x},\omega)}_{\text{direct illum.}} + \underbrace{L_r(\mathbf{x},\omega)}_{\text{VPL illum.}} + \underbrace{L_r'(\mathbf{x},\omega)}_{\text{compensation}}. \tag{4.8}$$

The compensation term $L_r'(\mathbf{x},\omega)$ is computed using Monte-Carlo integration in conjunction with ray casting to sample incident illumination over the hemisphere aligned with the surface normal at $\mathbf{x}$. Note that on the right-hand side of Equation 4.7 the unbiased illumination $L(\mathbf{x},\omega)$ occurs (Equation 4.8). This implies that VPL illumination (Equation 4.6) and bias compensation (Equation 4.7) is necessary for all surface points returned by the ray casting function. Kollig and Keller [2006] have shown that VPL lighting with bias compensation can be interpreted as a variant of bidirectional path tracing [Lafortune, 1996], but can also be viewed as a variant of final gathering, that gathers incident illumination only from surfaces that lie within the distance

$$d_{\max} < \sqrt{\frac{\cos^+\theta_{\mathbf{x}\rightarrow\mathbf{y}}}{b}}.$$

The maximum distance $d_{\max}$ is a direct result from the bounded geometry term G' (see above) [Kollig and Keller, 2006]. Similar to final gathering for photon mapping, this bias compensation is the most expensive part of the entire algorithm.

In Chapter 11 we present a novel reformulation of the bias to derive an efficient screen-space compensation algorithm, which computes the compensation term solely from the illumination present in an image. This avoids costly ray-casting operations and achieves interactive framerates on commodity GPUs.

Besides bounding and bias compensation, other variants to avoid the inherent singularity have been proposed. Hašan et al. [2009] introduced virtual spherical lights (VSLs) that have finite extent. Incident illumination is integrated over the entire solid angle it subtends at shading points, but visibility is assumed to be constant. The approach by Davidovič et al. [2010] creates millions of virtual local lights that recover short distance illumination lost due to bounding the geometry term.

Other extensions to improve VPL algorithms have also been proposed. In the spirit of bidirectional path tracing [Lafortune, 1996; Veach, 1997], Segovia et al. [2006a] proposed bidirectional instant radiosity to select VPLs that actually contribute to the final image. Later, Metropolis sampling was also applied to VPL generation [Segovia et al., 2007]. Although VPL-based methods achieve stunning results in purely diffuse scenes even with only hundreds of VPLs, the number of VPLs required to accurately light (moderately) glossy objects is magnitudes larger [Křivánek et al., 2010]. The reason being that only relatively few paths (several hundreds) are created and thus the set of all possible light paths is only sparsely sampled. Since the computational demand of lighting with VPLs grows linearly with their number, rendering scenes with thousands to hundreds of thousands of VPLs is impractical. Walter et al. [2005] proposed light cuts, building a hierarchy of VPLs and selecting only those with significant contribution to shading points. This approach was later extended [Walter et al., 2006] to efficiently handle scenes with anti-aliasing, motion blur and depth-of-field. Matrix row-column sampling [Hašan et al., 2007] is an algorithm suitable for being implemented on the GPU. In contrast to light cuts [Walter et al., 2005], the algorithm does not compute different VPL sets per shading point, but clusters VPLs and sums up the clusters' contribution to the entire image, i.e. to all shading points.

Extension to participating media: It is possible to render full multiple scattering with only single scattering from virtual point light sources. How this and bias compensation is efficiently done is discussed in Chapter 10.

4.6 Global Illumination in Participating Media

Rendering participating media can look back onto a long history in computer graphics. The first physical-based scattering model [Blinn, 1982] was restricted to single scattering, uniform density, and infinite light sources, but was analytically solvable. Kajiya and Von Herzen [1984] presented a general two-pass algorithm that lifted those restrictions and supports multiple scattering. First, in-scattered light was projected onto spherical harmonics for each voxel in a 3D grid, and second, all in-scattered radiance was accumulated along a ray. Max [1986] computes single scattering only within the bounds of shadow volumes [Crow, 1977] using scanline rasterization, while Nishita et al. [1987] rely on ray casting, also supporting light volumes and media with varying density.

Unfortunately, single scattering is only a valid assumption for participating media with low albedo (i.e. low scattering), which is not true for most media where multiple scattering is crucial, e.g. in clouds or smoke.

Discrete ordinate methods [Chandrasekhar, 1960; Languénou et al., 1994; Fattal, 2009] discretize space (voxels) and directions (spherical parametrizations) and compute light transport using local transport operators exchanging energy between adjacent cells only. A similar approach is described by Stam [1995] who models multiple scattering effects using the diffusion equation. Beyond that, point spread functions for multiple scattering [Premoze et al., 2004] were derived from close analysis of light transport in participating media.

Radiosity: Global illumination algorithms originally developed for surface light transport were later extended to support participating media and multiple scattering. The zonal method [Rushmeier and Torrance, 1987] extends radiosity [Goral et al., 1984] to volumetric finite elements (voxels) and handles volumetric light transport with isotropic scattering. This was later extended by Bhate and A. [1992] who use spherical harmonics to represent a voxel's radiation which in turn allowed them to use anisotropic phase functions.

Path tracing: In contrast to finite element methods, Monte-Carlo algorithms do not rely on a discretization of the volume and can handle arbitrary density functions [Lafortune and Willems, 1996]. Light transport in participating media using bidirectional path tracing was first presented by Lafortune and Willems [1996]. They derived analytical distance sampling in media of uniform density and assume piecewise constant density properties for sampling in non-uniform media. Pauly et al. [2000] extend Metropolis light transport [Veach, 1997] to participating media, but also rely on ray marching for free path sampling and computing transmittance. The issue of distance sampling, or free path sampling, in inhomogeneous media was later picked up by Raab et al. [2008]. They introduced Woodcock-tracking [Woodcock et al., 1965], a technique from computational physics, and derived a practical algorithm for unbiased free path sampling. The performance of this approach depends on the maximum extinction in the entire volume. To improve performance, Yue et al. [2010] use cost heuristics to build kd-trees subdividing volume densities into regions of near uniformity which are sampled adaptively.

Photon mapping: Volumetric photon mapping made efficient rendering of scattering effects possible [Jensen and Christensen, 1998]. Ray marching was used to accumulate in-scattering along eye rays. At each step during ray marching, the volume photon map is queried, and in-scattered radiance, supporting arbitrary phase functions, estimated. Instead of ray marching and using disjoint photon map queries along the ray, cylindrical queries used by the beam radiance estimate [Jarosz et al., 2008] gather all photons in the proximity of a ray, speeding up the gathering process significantly. Despite those improvements, rendering quality strongly depends on the photon density in the photon map. Due to memory constraints, the density cannot be arbitrarily high. Progressive photon mapping [Hachisuka et al., 2008] lifted the memory constraints by combining results of several thousand photon maps progressively, accounting for several hundreds of millions of photons. Although this was first shown only for surface illumination, the method was extended to participating media shortly after [Jarosz et al., 2011; Knaus and Zwicker, 2011].

Many light methods: Many light methods are also suitable for volume illumination. It is briefly discussed in the context of instant radiosity by [Raab et al., 2008], who also extends bias compensation [Kollig and Keller, 2006] to arbitrary volume densities. Hierarchical VPL clustering methods [Walter et al., 2005, 2006] can help to reduce computation costs for volume lighting with many virtual point lights. Besides clustering VPLs only, multidimensional light cuts [Walter et al., 2006] also cluster samples at which in-scattering is computed. Kulla and Fajardo [2011] exploit the fact, that the contribution of a point light to in-scattering along a ray segment highly depends on its distance to the latter. They derive a new importance sampling technique that shows better convergence behaviour than traditional ray marching, and unbiased ray integration techniques based on Woodcock tracking [Raab et al., 2008; Jarosz et al., 2011].

4.7 Real-Time Global Illumination

Real-time global illumination is intricate. Even today's hardware is not powerful enough to support a straight forward implementation of the aforementioned algorithms. Efforts exist, but despite sophisticated and highly optimized data-structures [Zhou et al., 2008], algorithms [Popov et al., 2007] and careful programming of the underlying hardware [Novák et al., 2010], interactive image generation is either restricted to particular phenomena (e.g. caustics), to primary rays only, or is progressive at best.

In general, interactive and real-time algorithms cut back on the universality of light transport, are commonly tailored to specific problems or do not support the full spectrum of possible phenomena.

Early attempts at interactive global illumination [Drettakis and Sillion, 1997] maintain a precomputed radiosity solution, incrementally updating it after an object has been moved. Other approaches, now know as *precomputed radiance transfer* (PRT) [Sloan et al., 2002], compactly store incident radiance and the product of the BRDF and the cosine term (cf. Chapter 3.2) in a spherical harmonics representation, making it possible to express the shading integral as simple multiplication in frequency space which can be evaluated at run-time. Most of the PRT variants require static geometry, although some recent extensions also allow the movement of rigid objects [Iwaski et al., 2007], but fully dynamic scenes are out of question. Low-resolution dynamic scenes are possible, assuming low-frequency illumination [Ren et al., 2006; Sloan et al., 2007].

Lehtinen et al. [2008] presented an interactive PRT-based illumination method for static scenes using a hierarchical, point-based representation. Light transport was simulated in a similar manner to radiosity [Cohen et al., 1993]. Related to radiosity is antiradiance [Dachsbacher et al., 2007] which renders moderately complex scenes with multiple bounces of indirect illumination at interactive framerates. The required link hierarchy, however, is precomputed and dynamic scenes require more work [Meyer et al., 2009].

VPL methods: A popular method for real-time global illumination is instant radiosity [Keller, 1997] because it represents the equilibrium distribution of radiance as a set of virtual point lights (VPLs), which are highly amendable for efficient GPU evaluation. Direct light reflected by scene surfaces can be stored in shadow maps, storing position, normal and material properties. Although this information has first been used to render interactive translucency effects using translucent shadow maps (TSM) [Dachsbacher and Stamminger, 2003], it has later been extended to render one-bounce indirect illumination with reflected shadow maps (RSM) [Dachsbacher and Stamminger, 2005]. Therefore, the samples in the RSM are interpreted as VPLs whose sum contribute to one-bounce indirect illumination at each shading sample. For reasons of efficiency, the visibility to VPLs is ignored. Other global effects based on the RSM are also possible, i.e. caustics [Dachsbacher and Stamminger, 2006]. Creating shadow maps for each VPL to account for visibility is too expensive in each frame. Thus Laine et al. [2007] reuse already created shadow maps as long as possible and incrementally add new ones on demand. Imperfect shadow maps [Ritschel et al., 2008] achieve interactive framerates for moderately complex and fully dynamic scenes using approximate visibility, but ultimately fail to handle large scenes. Further, indirect shadows are generally smoothed out considerably. However these issues were later addressed using view-adaptive sampling schemes [Ritschel et al., 2011]. Holländer et al. [2011] propose, following in the footsteps of Micro-Rendering (cf. Chapter 8), point rendering with sophisticated level of detail techniques to generate many VPL shadow maps in parallel, achieving results with impressive speed.

Screen-space methods: Another popular class of rendering techniques are global illumination approximations in screen space. These GPU-friendly techniques are typically used when the need for high performance outweighs that of quality. However, they are output-sensitive, adapt to the current visible geometry in the scene and thereby avoid irrelevant computations. Many of them are based on reflective shadow maps (RSMs) [Dachsbacher and Stamminger, 2005]. Multi-resolution splatting [Nichols and Wyman, 2009; Wyman et al., 2010] computes indirect lighting (without visibility) from the RSM at lower resolution for smooth surfaces and falls back to high resolutions in image regions with varying geometry. Image-space radiosity [Nichols et al., 2009] uses a hierarchy for both RSM and surface elements stored in a geometry buffer [Saito and Takahashi, 1990]. Computing ambient occlusion (e.g. see [Mittring, 2007; Bavoil et al., 2008]) or volumetric obscurance [Loos and Sloan, 2010] in image-space is widely used nowadays and has been extended by Ritschel et al. [2009b] to account for directional lighting with coloured shadows and indirect illumination from nearby surfaces.

Participating media: Rendering participating media in real-time is intricate and most often limited to single scattering effects. Many approaches are based on shadow volumes to provide information on the regions of illuminated and shadowed space in front of visible surfaces. To identify these regions, the shadow volume polygons must be sorted back-to-front which introduces sorting cost [Venceslas et al., 2006; Mech, 2002; James, 2003]. Gautron et al. [2009] compute light cones (instead of shadow volumes) and compute the intersection of an eye ray and a cone to determine the length of the eye path through the light.

Wyman and Ramsey [2008] render in-scattering from textured and shadowed spot lights using ray marching and use shadow volumes to distinguish between lit and potentially shadowed parts of the scene. They also use naïve image space sub-sampling to reduce the number of rays. Tóth and Umenhoffer [2009] propose to use interleaved sampling in screen space exploiting the fact that nearby pixels cover similar geometry; the in-scattered light is computed using ray marching and shadow mapping.

Slicing techniques, known from volume rendering, can be used to emulate ray marching through participating media. Dobashi et al. [2002] and Mitchell [2004] use slicing to render volumetric shadows by rendering quads parallel to the image plane at varying distances from the eye. These slices are accumulated back-to-front using blending while the shadows on each slice are computed using shadow mapping. Imagire et al. [2007] reduce the number of slices by averaging the illumination over regions near each slice.

Mitchell [2007] presents a simple post-processing technique working in screen space that is very fast but has inherent limitations including non-textured lights and false light shafts from objects behind the light sources. Similar restrictions apply to the technique presented by Sousa [2008] that uses a radial blur on the scene depth buffer to obtain a blending mask for sun shafts.

Based on epipolar geometry, we have developed an algorithm that computes an optimized distribution of samples in image space where in-scattering along eye rays is computed (cf. Chapter 9). Baran et al. [2010] construct epipolar planes in which in-scattered light is sampled and volumetric shadows incrementally propagated. Exploiting partial sum trees, they efficiently accumulate all in-scattered radiance along eye rays. 1D mip map hierarchies on shadow maps rectified in epipolar space allowed Chen et al. [2011] to quickly locate lit ray segments for evaluating single scattering.

If visibility can be ignored (e.g. when accumulating light within a light volume), even analytical solutions to single scattering in homogeneous participating media exist [Lecocq et al., 2000; Sun et al., 2005; Pegoraro, 2009].

Although early work only supported isotropic phase functions, this limitation has been lifted and anisotropic scattering models have been integrated [Pegoraro et al., 2010].

Upsampling: Oftentimes, interactive global illumination algorithms are still too demanding and forbid satisfactory rendering performance for high resolution images. Ward et al. [1988] already noted that indirect illumination varies smoothly over the image, which led to the development of irradiance caching (IC). IC computes indirect illumination sparsely in image space and reconstructs missing information afterwards through interpolation. Since the approach was only applicable for surfaces with diffuse reflectance, Křivánek et al. [2005] extended IC to moderately glossy surfaces, of which even a GPU variant [Gautron et al., 2005] exists.

For real-time rendering, often much simpler interpolation schemes are used. GI is computed at a lower resolution of the target image across-the-board, and than reconstructed. Bilateral filtering [Tomasi and Manduchi, 1998; Sloan et al., 2007] avoids blurring over normal and depth discontinuities, detected in image space in a geometry buffer [Saito and Takahashi, 1990], where the illumination varies strongly.

More sophisticated upsampling techniques build discontinuity hierarchies in image space [Nichols and Wyman, 2009; Nichols et al., 2009] and chose the proper reduced resolution based on the smoothness of the surface within image regions. A reconstruction pass finally combines all hierarchy levels to generate the target resolution.

Interleaved sampling [Keller and Heidrich, 2001] uses blocks of irregular sampling patterns spanning blocks of pixels (usually 2×2 or 4×4 pixels) to distribute computations over the entire block. This becomes apparent as structured artefacts, and thus the image is subsequently blurred. Segovia et al. [2006b] present an efficient GPU implementation for this approach and applied it to increase performance for computing illumination from VPLs.

Unfortunately, bilateral upsampling, its variants and interleaved sampling do not take strong changes in the illumination due to shadowing in to account. This leads to visible blurring of indirect shadows, which can be highly distracting in strongly indirectly illuminated scenes.

CHAPTER 5

Directional Parametrizations

DIRECTIONAL parametrizations play an important role in computer graphics. They make it possible to parametrize the entire environment seen from a point over one or more two-dimensional textures. The term 'environment' refers to all surfaces visible from a point in all directions. More importantly, these surfaces reflect light, and hence an environment map stores incident radiance for each direction at the point it is created, which is the foundation for reflection mapping [Blinn and Newell, 1976] and image based lighting [Miller and Hoffman, 1984; Debevec, 1998].

5.1 Environment Mapping

Environment mapping projects the entire environment visible from a point into one or more two-dimensional textures. To that end, directional parametrizations map texture coordinates into direction vectors and vice versa. Blinn and Newell [1976] used this technique for the first time rendering perfect specular reflections onto surfaces of arbitrary shape. Under the assumption that the environment is at infinite distance at all shading points, a single environment map rendered at the reflecting object's centre proved to be sufficient.

Miller and Hoffman [1984] noted that environment maps are not limited to reflection rendering. They suggested to interpret an environment map as *illumination map*. Logically, this assumption allowed them to extend reflection mapping and solve the shading integral (Section 3.2.1) for arbitrary local reflection models. In contrast to the method of Blinn and Newell [1976], also referred to as *latitude-longitude mapping*, the environment or illumination map was not created rendering virtual scenes but from taking photographs of mirror spheres. These photographs, also called *light probes*, depict the reflection of the real world environment in the mirror. Light probes imply a directional parametrization that is not compatible with the one developed by Blinn and Newell [1976]. This is why Miller and Hoffman [1984] developed *sphere mapping*, at the same time as Williams [1983]. This approach was later picked up by Debevec [1998] for image based lighting with high dynamic range illumination maps [Debevec and Malik, 1997], and it is still used today for natural illumination.

Unfortunately the latitude-longitude and sphere projection exhibit severe drawbacks. Sphere mapping suffers from strong aliasing towards the silhouettes of the photographed sphere. The reflected environment is highly condensed in thin image regions close to the silhouette. Beyond that the sphere projection scheme exhibits a singularity directly on the silhouette. Latitude-longitude mapping shares similar problems. The distortion of the environment increases towards the poles of the sphere, and besides the singularities at the poles, reflection mapping makes the texture seams visible when the mapping wraps around in the latitude dimension.

Cube mapping proposed by Greene [1986] overcomes those aforementioned drawbacks. His parametrization is based on the unit cube where the environment is projected onto the cube's faces. Due to the linearity of the projection onto each cube face, this parametrization was the first that could be generated directly on graphics hardware using rasterization, but at the price of generating six individual images. To leverage the costs of creating environment maps on graphic hardware, Heidrich and Seidel [1998] proposed to project the environment map onto two paraboloids. As cube mapping, this projection can be carried out on graphics hardware. It requires only two images, but high tesselation factors to reduce approximation errors due to the non-linearity of the projection.

The field for applications of environment mapping is broad. Besides storing natural illumination [Miller and Hoffman, 1984; Debevec, 1998], which is equally used in offline rendering systems [Pharr and Humphreys, 2010] and interactive techniques [Sloan et al., 2002], directional parametrizations are also used for finite element methods [Cohen and Greenberg, 1985; Coombe et al., 2004] in order to store radiative energy incident at a shading point. Micro-Rendering (cf. Chapter 8) follows the same approach. Omnidirectional shadow maps store depth instead of illumination and are used for point light sources and virtual point lights [Ritschel et al., 2008].

5.1.1 Latitude-Longitude Mapping

The method by Blinn and Newell [1976] is also called latitude-longitude mapping. Their idea was to map u, v-texture coordinates directly to spherical coordinates (ϕ, θ),

$$\begin{aligned}\phi &= 2\pi u, \\ \theta &= v\pi\end{aligned} \tag{5.1}$$

from which the normalized direction vector $\omega = (\cos\phi \sin\theta, \sin\phi \sin\theta, \cos\theta)^T$ is obtained (Figure 5.1 left). Inversely, the normalized direction vector $\omega = (\omega_x, \omega_y, \omega_z)^T$ is mapped onto texture coordinates as follows:

$$\begin{aligned}u &= \operatorname{atan2}(\omega_y/\omega_x), \\ v &= \arccos(\omega_z).\end{aligned}$$

This kind of environment map can only be generated synthetically as there is no device that is capable of recording such an image. Common approaches to generate such images are ray casting or re-sampling other parametrizations, such as sphere maps (Section 5.1.2). The distortion that stems from Latitude-Longitude mapping is depicted in Figure 5.1 right. In the proximity of the equatorial region, the environment is undersampled by a factor of ≈ 1.6, comparing to optimal, uniform sampling. In proximity of the poles $(|\theta| \gtrsim 0.9)$, oversampling occurs and eventually reaches infinity at the poles (singularities).

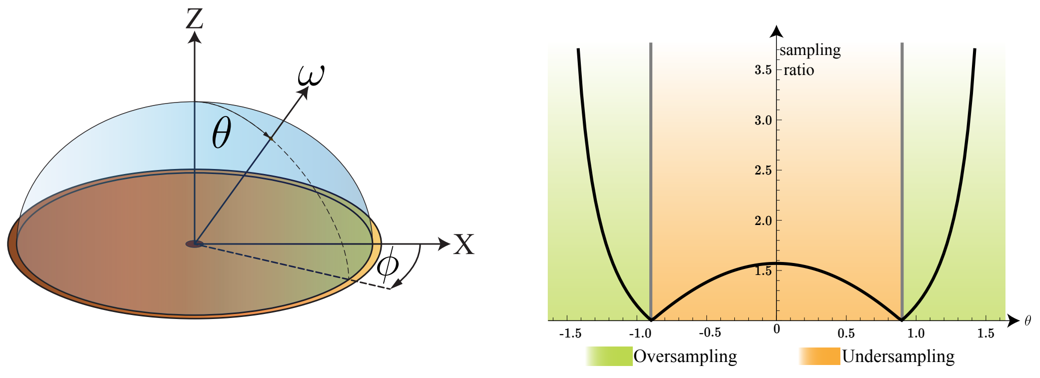


Figure 5.1: ***Left:*** *The direction vector ω is parametrized over the two spherical coordinates ϕ and θ, which are directly mapped onto texture coordinates.* ***Right:*** *Moving along a geodesic, the singularity appears at the poles. Compared to uniform sampling, the mapping oversamples in the proximity of the poles and undersamples in equatorial regions.*

5.1.2 Sphere Mapping

Sphere mapping is derived from the reflection seen in a mirror sphere assuming an orthogonal view. The mirror sphere is placed at the origin of the coordinate system, and the view direction is aligned with the negative z-axis (Figure 5.2 left). The intersection of the view direction with the xy-plane yields the uv-texture coordinates. These are first mapped to the sphere using Nusselt's parametrization

$$\mathbf{n} = \left(u, v, \sqrt{1-u^2-v^2}\right)^T$$

to obtain the normal vector on the sphere where the ray intersects its surface. At this point the reflected environment is visible through the reflection vector $\omega = \mathbf{v} - 2\langle \mathbf{n}, \mathbf{v}\rangle \mathbf{n}$. The projection of direction vectors onto texture coordinates can then be easily derived:

$$u = -\frac{\omega_x}{\sqrt{2(1-\omega_z)}}, v = -\frac{\omega_y}{\sqrt{2(1-\omega_z)}}.$$

One notices that the environment on the front side is captured for all uv-coordinates ($u, v \in [-1, 1]$) for which the constraint

$$\sqrt{u^2+v^2} \leq \frac{\sqrt{2}}{2}$$

holds. Otherwise the environment behind the light probe appears. Unfortunately, the sampling rate rapidly decreases on the backside. This becomes

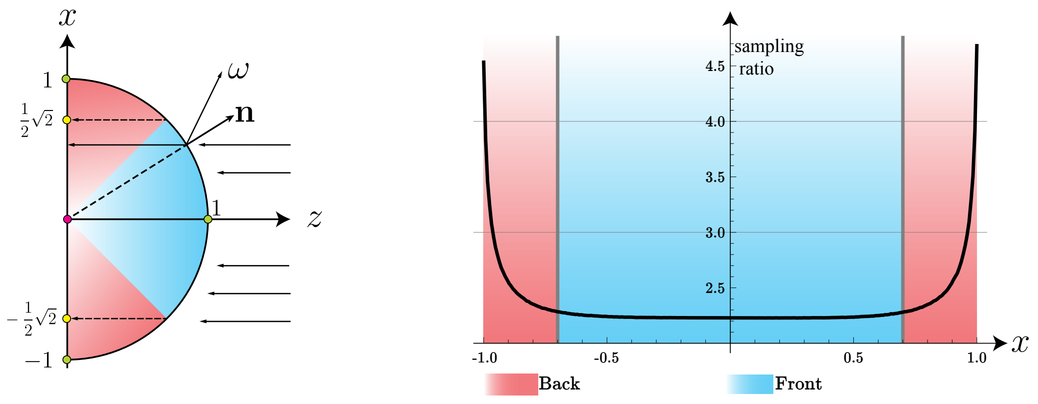


Figure 5.2: ***Left:*** *View vectors are aligned with the z-Axis, and the projection captures the front- and backside of the environment.* ***Right:*** *Sampling ratio of sphere mapping (SM) to uniform sampling. SM always undersamples and condenses the environment on the back side into small image regions.*

more and more apparent the farther the texture coordinates move towards the silhouette of the image of the mirror sphere, where the projection exhibits a singularity. Figure 5.2 right depicts the ratio of the solid angle captured by a reflection vectors to sampling the sphere uniformly.

5.1.3 Cube Mapping

The cube map [Greene, 1986] projects the environment onto the six faces of the unit cube (Figure 5.3 left). The projection $\mathbf{p} = (p_u, p_v, p_w)^T$ of a unit vector ω onto the cube map is simply obtained by

$$\mathbf{p} = \frac{\omega}{|\omega_i|}$$

were ω_i is the component of the direction vector with maximum extent. Here the texture coordinate is three-dimensional, and the components $p_i \in \{-1, 1\}$ encode the cube face. The inverse projection is simply the normalization of $\mathbf{p}$.

$$\omega = \frac{\mathbf{p}}{||\mathbf{p}||}$$

Using perspective projections with a field of view of 90°, rasterization can be used to generate cube maps, besides ray casting and resampling other parametrizations. Because look-up does not require costly, transcendental mathematical functions and generation can rely on rasterization, cube mapping has become the major, natively supported environment mapping

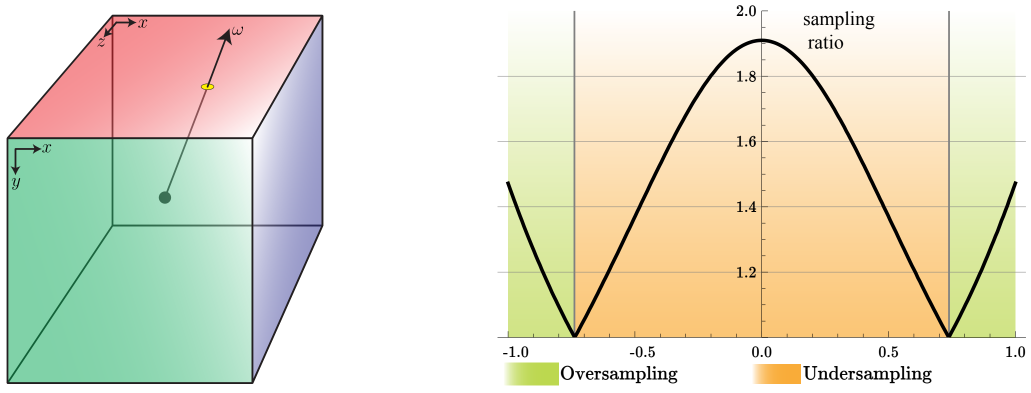


Figure 5.3: ***Left:*** *Direction vectors are projected onto the cube face corresponding to the vector's component with maximum extent.* ***Right:*** *Sampling ration to uniform sampling. The parametrization scheme undersamples in the centres of cube faces and oversamples close to the face borders.*

scheme on GPUs. In addition to that, it neither exhibits any singularity or excessive deterioration of the sampling rate as sphere and latitude-longitude mapping. Figure 5.3 right depicts the sampling behaviour of the cube map. It can be observed that the ratio of neither over- nor undersampling exceeds a factor of 2.

5.1.4 Paraboloid Mapping

Paraboloid mapping takes the same approach as sphere mapping. The environment is displayed as reflection on a mirrored paraboloid seen from an orthographic view, where the viewing direction is parallel to the negative z-axis (Figure 5.4 left). The environment appears in the reflection of the viewing vector at the paraboloid of the form

$$f(x,y) = \frac{1}{2} - \frac{1}{2}\left(x^2 + y^2\right).$$

Heidrich and Seidel [1998] showed, that the projection onto uv-texture coordinates ($u, v \in [-1;1]$) is simple and obtained from a reflection vector $\mathbf{r} = (r_x, r_y, r_z)^T$ as follows:

$$u = \frac{r_x}{(1+r_z)}, v = \frac{r_y}{(1+r_z)}.$$

Paraboloid mapping requires two textures, one for each hemisphere. The sign of the z-component of the reflection vector shows, whether the reflection occurred on the positive ($r_z \geq 0$) or negative ($r_z < 0$) hemisphere.

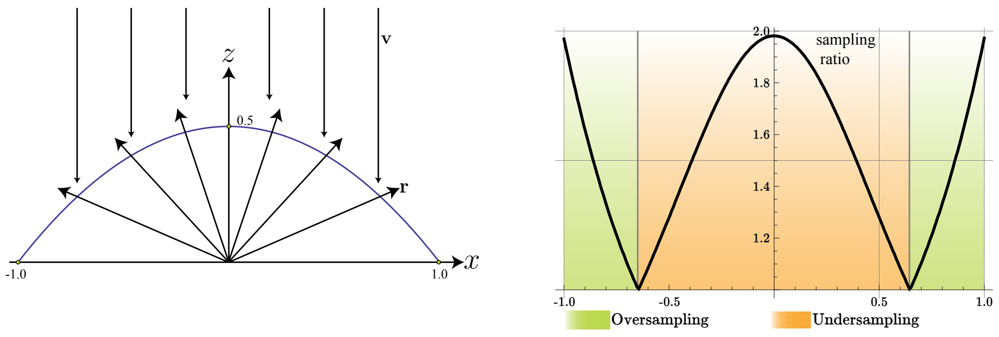


Figure 5.4: ***Left:*** *The environment is projected onto the paraboloid as if it was seen in the reflection* **r** *from an orthogonal viewing direction* **v**. ***Right:*** *Sampling ratio of the paraboloid projection to uniform sampling. The quality is comparable to cube mapping and neither over- nor undersampling exceeds a factor of* 2.

The inverse projection is found reflecting the viewing vector $\mathbf{v} = (0, 0, -1)^T$ at the surface of the paraboloid $\mathbf{s} = (u, v, f(u, v))^T$ with the normal $\mathbf{n} = \frac{\partial \mathbf{s}}{\partial u} \times \frac{\partial \mathbf{s}}{\partial v} / \| \frac{\partial \mathbf{s}}{\partial u} \times \frac{\partial \mathbf{s}}{\partial v} \|$.

The sampling behaviour is shown in Figure 5.4 right. As can be observed, the quality is comparable to that of cube mapping [Greene, 1986] and does not exhibit any singularity.

CHAPTER 6

Granular Visibility Queries

VISIBILITY determination is a widely studied topic in computer graphics and is required for many applications, such as walkthroughs for large scale environments [Funkhouser et al., 1996], computer games, and CAD modelling systems. Its ultimate goal is to identify occluded portions of a scene which do not contribute to the final image. This has been an active research topic since the 1970s, and many algorithms have been proposed to solve this occlusion culling problem (extensive overviews can be found in [Cohen-Or et al., 2003; Bittner and Wonka, 2003]).

More recent approaches in real-time computer graphics are based on hardware occlusion queries (HOQ) and predicated rendering (cf. Chapter 2.2.2), but these techniques introduce read-back latencies or do not guarantee culling. Furthermore, the finest granularity of both techniques is a single draw call, i.e. multiple queries are required if the visibility of parts of an object is to be determined, possibly resulting in a large number of draw calls. Consequently, culling does not become possible for instanced draw calls because individual instances cannot be queried separately. Visibility results are also not directly available in shaders (with predicates they are not available at all), where they can be useful to cull object parts or to control the level of detail of tessellation and other rendering techniques.

Determining the visibility of objects (in an output-sensitive manner) is basically a problem of counting pixels of rasterized geometry, and for that purpose we developed two approaches based on summed area tables and item buffers. These enable fine *granular visibility queries* on the GPU and make results directly available in shaders. [1]

[1]This chapter is based on our work [Engelhardt and Dachsbacher, 2009] presented at *Symposium on Interactive 3D Graphics and Games 2009 (I3D'09)*.

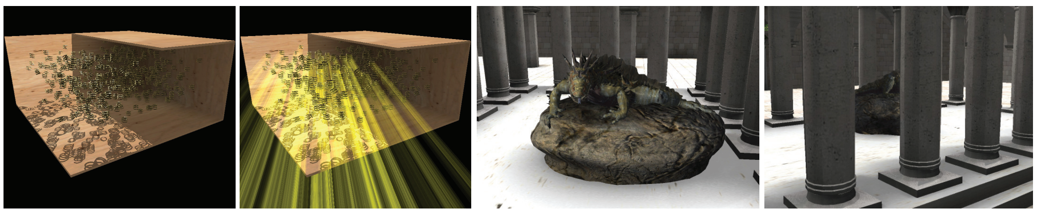


Figure 6.1: ***Left:*** *Shadow volumes are extruded in a geometry shader and granular visibility queries are used directly on the GPU to exclude unlit objects from the shadow volume generation.* ***Right:*** *Visibility determination on the GPU is also beneficial in costly rendering techniques, such as displacement mapping to cull individual, occluded primitives. Illustration: Engelhardt and Dachsbacher [2009].*

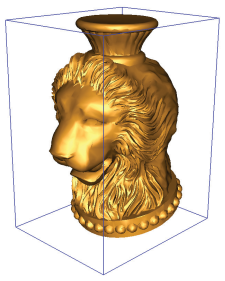
6.1 Granular Visibility Queries

The main goal of granular visibility queries is to provide efficient means for counting the number of pixels covered by the query objects on the screen. However in contrast to HOQs, we want to be able to query the visibility of subregions of objects individually and to make the results available to shaders directly on the GPU. We analyse two possible approaches for fast pixel counting: First, we use summed area tables to retrieve the average colour of an image region, and thus we retrieve the screen coverage of query objects once we rendered them appropriately coloured. Second, we introduce a new variant of item buffers which can be quickly rendered and evaluated on contemporary GPUs.

6.1.1 Pixel Counting with Summed Area Tables

The basic idea of our first approach is to render query objects into a colour texture and count the pixels belonging to each object afterwards. Two GPU-friendly methods exist to accelerate this process: summed area tables (SATs) and mip maps allow us to retrieve the accumulated or average colour within a rectangular image region efficiently. Both treat colour channels independently and by rendering query objects in pure red, green, blue, or into the alpha channel, we can distinguish pixels of up to four objects per image region (given a standard four channel texture). The crucial point for a practical algorithm is to develop a good heuristic for colouring the objects and constructing an as minimal as possible set of texture accesses to determine the visibility of an object. We focus our discussion in this section on SATs and will comment on mip maps in Section 6.1.4. Our heuristic is a three-step process performed every frame: First, we assign query objects to colour channels such that no overlapping and fusing in image space occurs (step 1). Obviously, this is not possible if more query objects overlap than colour channels are available. In this case we cannot assign such conflict objects as a whole to a colour channel – instead we assign portions of these objects to different colour channels (step 2). In step 3, we determine a set of texture accesses to the SAT for each query object to retrieve the visibility. Step 1 to 3 are executed on the CPU and require access to the objects' transformations and bounding boxes. The SAT computation takes place on the GPU, and hence the visibility queries are then possible in any shader program.

6.1.2 The Conflict Graph



The goal of the first step of our algorithm is to assign as many query objects as possible to colour channels in a way such that fusion in image space does not occur. For this we build a conflict graph: Each vertex of the graph corresponds to one query object. A conflict between two objects exists if their bounding rectangles in image space overlap; in this case the graph contains an edge between the two corresponding vertices. We chose this conservative overlap test particularly with regard to the use of SATs: In favour of fewer texture accesses we want to avoid non-rectangular regions in advance. Please note that we do not assume a depth sorting of the objects as this does not provide any advantages unless we perform an occlusion culling procedure beforehand. For colouring the conflict graph's vertices, we adapt the Chaitin-Briggs algorithm from the domain of compiler design [Muchnick, 1997]: In its original application it is used for register allocation. In our case, the algorithm must not assign two objects in conflict to the same colour channel. During graph colouring, conflict objects are identified and separated for special treatment afterwards. Optimal colouring is a NP-complete problem, the graph colouring algorithm (possibly yielding suboptimal results) is $O(n^2)$. The first phase of the algorithm is the graph decomposition: We find the vertex with the least incident edges, remove this vertex and all its edges from the graph, and store it on a stack. This is repeated until the entire graph has been decomposed. The second phase reconstructs the graph and assigns colours. For this purpose, the following steps are repeated until the stack is empty:

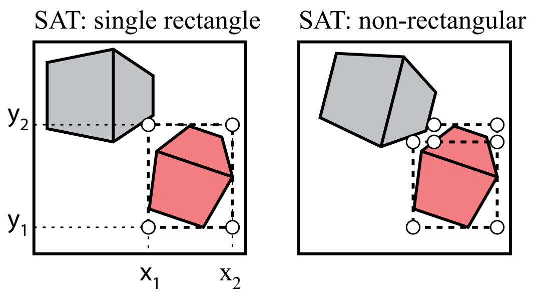


Figure 6.2: ***Left:*** *The query of a rectangular region from a SAT requires 4 texture lookups (shown as circles).* ***Right:*** *For non-rectangular query regions samples can often be reused. Illustration: Engelhardt and Dachsbacher [2009].*

- Take the top-most vertex from the stack, reinsert it into the graph and restore edges to adjacent vertices if they have already been reconstructed.
- If the vertex does not have any neighbours yet, assign it to an arbitrary colour channel, otherwise select a channel which is not used by one of its neighbours.
- If no colour channel is available, mark this vertex as conflict object and do not insert it into the graph (it will undergo special treatment afterwards).

This algorithm removes vertices with fewer incident edges with higher priority. During graph reconstruction, the vertices are reconstructed in the opposite order, effectively colouring objects in more difficult configurations first and colouring unproblematic objects (with few or no conflicts) later. The output of the graph colouring are two sets of objects: Non-conflict objects which have been assigned to colour channels as a whole and conflict objects which could not be assigned.

6.1.3 The Region Set

A SAT allows us to compute the sum of pixel colours in an axis-aligned rectangular region $[x_1, x_2] \times [y_1, y_2]$ in image space, with $0 \le x_1 \le x_2 \le 1$ and $0 \le y_1 \le y_2 \le 1$ easily. Each query of a rectangular region requires four lookups to the SAT. Non-rectangular shapes can be approximated with multiple rectangles and often samples can be shared among the queries (cf. Figure 6.2). Of course, the SAT will be generated at a finite resolution

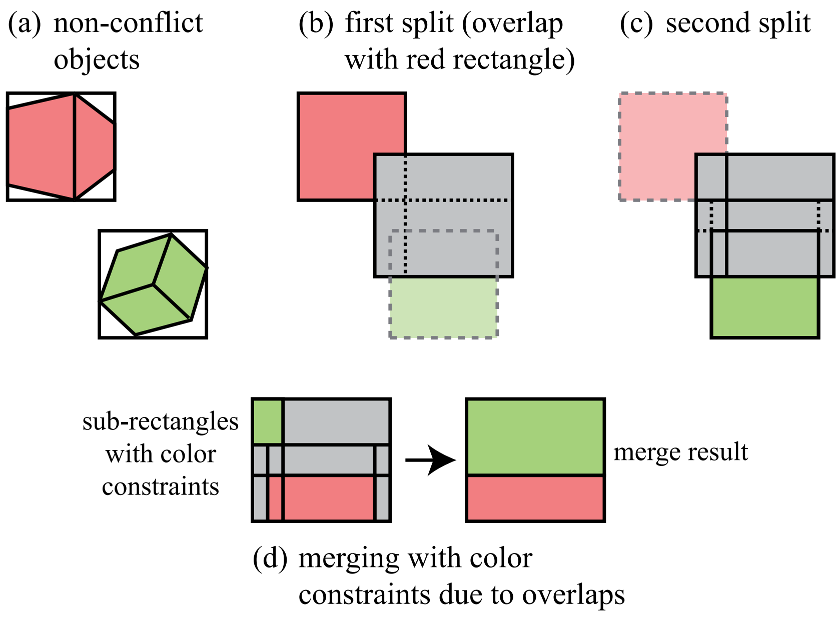

(a) non-conflict objects
(b) first split (overlap with red rectangle)
(c) second split
sub-rectangles with color constraints
merge result
(d) merging with color constraints due to overlaps

Figure 6.3: *A conflict object is split recursively (b and c); each rectangle keeps track of free colour channels (there are only two channels in this example). (d) Adjacent sub-rectangles are merged iteratively if they can be assigned to the same colour channel. Illustration: Engelhardt and Dachsbacher [2009].*

(1024^2 in our examples) and $x_i, y_i \in [0;1]$ are mapped to pixels accordingly. A region set represents the information where to lookup the SATs in order to determine the area covered by an object, i.e. it contains $i \geq 1$ SAT-lookups consisting of a rectangle and an associated colour channel: $([x_{i,1}, x_{i,2}] \times [y_{i,1}, y_{i,2}], c_i)$. The graph colouring assigns non-conflict objects to colour channels, and the visibility for every such object corresponds to a single SAT query: The rectangular region corresponding to the object's bounding rectangle (Figure 6.3a). Next, we need to split and assign parts of the conflict objects to colour channels. All operations are performed on the bounding rectangles in image space only, thus keeping the necessary operations simple and efficient. We process one conflict object at a time by splitting its bounding rectangle and keeping track of free and occupied colour channels for each sub-rectangle. The following steps are performed for each conflict object:

- Find the next overlapping SAT lookup (that has already been created) and split the conflict object into axis-aligned rectangles (Figure 6.3b shows a split operation).

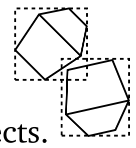
- For each sub-rectangle we store flags indicating which colour channels are already occupied by other objects. The flags of the overlap region (of the object to be inserted) are updated accordingly after the split.

- We proceed recursively for all sub-rectangles and test for overlaps with the remaining SAT-lookups (Figure 6.3c).

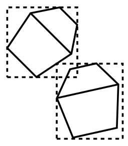
- The sub-rectangles form a partition of the initial bounding rectangle and in order to reduce the number of SAT queries for the conflict object, we iteratively merge sub-rectangles sharing an edge if their flags indicate that they can be assigned to the same colour channel (Figure 6.3d).

Difficulties and optimizations: So far the outlined algorithm is based on the assumption that there is always a free colour channel for each sub-rectangle. However, configurations exist where this is not the case and they occur more frequently with an increasing number of conflict objects and higher depth complexity. Obtaining accurate results with rectangular queries may also create an exceedingly large number of SAT lookups and thus splitting is to be terminated. By looking at the silhouette of the objects' bounding volumes in image space we distinguish the three possible overlap configurations that may occur in these cases and handle them – possibly by sacrificing accuracy – as follows.

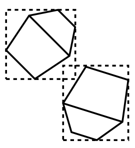
Overlap conflict: Both objects intersect the overlap region. Since we omit further splitting we overestimate visibility and share the overlap region across both objects, i.e. pixels in that region contribute to the visibility of both objects.



Overlap no conflict: If only one object intersects the overlap region then we split the rectangle of the respective other object. In favour of less SAT queries, we can over-estimate visibility in this case as well and abandon the rectangle split.

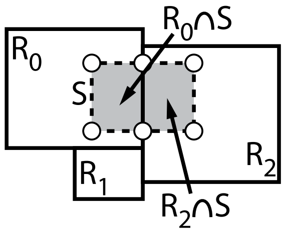


Empty overlap: The simplest case is when no object intersects the overlap region. If we detect this case, no rectangle needs to be split.



Creating the summed area table: The region set for each object is sent to the GPU via shader constants. Non-conflict objects are rasterized into the query texture into a single colour channel, whereas rendering the conflict objects is slightly more involved: In a fragment shader, we determine for each pixel to which rectangle it belongs and choose the colour accordingly. The summed area table itself is computed on the GPU using the fast generation algorithm by Hensley et al. [2005].

6.1.4 Querying the Visibility of Subregions



We can now query the visibility of arbitrary subregions of objects, e.g. a rectangular subregion $S = [x_1, x_2] \times [y_1, y_2]$. We determine the intersections of S and the region set rectangles $R_i = [x_{i,1}, x_{i,2}] \times [y_{i,1}, y_{i,2}]$, with

$$S \cap R_i = [\max(x_1, x_{i,1}), \min(x_2, x_{i,2})] \times [\max(y_1, y_{i,1}), \min(y_2, y_{i,2})]$$

All non-empty $S \cap R_i$ are evaluated using the SAT and their accumulated contribution yields the visibility of the sub-region.

Pixel Counting with mip-maps: Similar to SATs we can use mip-map pyramids (MMPs) to retrieve the average colour of larger image regions efficiently and we implemented this approach for comparison. However, look-ups with MMPs are less flexible due to their generation scheme. During the construction of the region set, we keep track of the occupancy of texels in the MMP using one quad-tree for each colour channel such that each quad-tree node corresponds to one colour channel in one texel in the MMP.

At first sight, MMPs are tempting as few lookups allow us to query large image regions and this proved right in scenes with few objects. However it turned out that the region sets, and with it the CPU overhead for maintaining the quad-tree updates, become very large when the number of query objects increases. In the end, the SAT approach outperformed MMPs for all reasonably complex test scenes and we consequently refrain from a detailed description.

6.2 Hierarchical Item Buffers

Our second approach is an extension to the well-known item buffer algorithm [Weghorst et al., 1984]. In its original formulation a unique ID (a

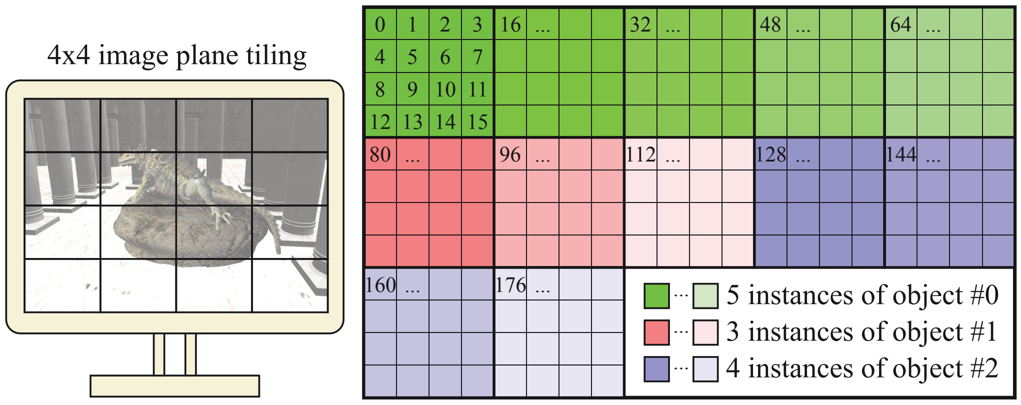


Figure 6.4: *A 2D histogram (computed from a hierarchical item buffer) with 192 bins due to a 4x4 image plane tiling and 12 instances from 3 different objects. Mip-mapping can be used to query the visibility of instances in individual screen tiles up to the whole screen. Figures in the coloured boxes denote the ID of the instance/tile pairs. Illustration: Engelhardt and Dachsbacher [2009].*

colour) is assigned to each object and the scene is rendered to a buffer. Then the visibility can be determined by counting the pixels storing the respective IDs. Scheuermann and Hensley [2007] demonstrate a fast histogram computation on GPUs which can be used for item buffer counting: The item buffer is bound to the input stage of the graphics pipeline and interpreted as a point list. The vertex shader scatters each point by computing a bin index from each pixel and a point primitive is rendered (with additive blending turned on) into the histogram texture. We adapted their method to the needs in our visibility determination. First, we use a special layout and compute 2D histogram textures such that we can query the visibility of objects (and instances) with a granularity of a predefined screen tiling. To this end, we virtually divide the screen into $2^t \times 2^t, t \geq 0$, tiles. Thus the 2D histogram texture (Figure 6.4) contains one square region of $2^t \times 2^t$ bins for each object instance. When rendering into the item buffer, the ID can be computed in a pixel shader from the base ID of the object, the instance index and the screen tile where the pixel resides:

$$ID = \text{baseID} + \text{instanceIndex} \cdot 2^{2t} + x + y \cdot 2^t$$

where $x, y \in [0 \cdots 2^t - 1]$ denote the horizontal and vertical screen tile location of the pixel. Please note that an even finer granularity is imaginable, e.g. by assigning different IDs to triangle groups or clusters within an instance.

Next the histogram is computed from the item buffer; each ID used in the item buffer refers to a bin in the 2D histogram as shown in Figure 6.4. In item buffers, pixels containing the same ID are often clumped together. During the histogram generation these pixels are sent to the same histogram bin thus hindering GPU parallelization (this problem occurs rarely for image histograms). To alleviate this, we apply a simple reordering trick to the scattering operations and process the item buffer pixels in randomized order using a precomputed index buffer.

Figure 6.4 shows an example for a 2D histogram texture with a layout for $t = 2$ and a total of 15 instances from 3 different objects. It can be used to directly lookup the visibility of each object instance within each screen tile. We call this method the hierarchical item buffer (HIB), as we can also use the 2D histogram texture together with its mip maps to efficiently query visibility of both whole objects and sub-regions: The HIB allows us, for example, to query the visibility of the first instance in the screen tiles 2, 3, 6, and 7 with a single lookup into mip level 1 (the full resolution texture is mip level 0) or the total visibility of instances in the mip level 2. The mip maps can be automatically generated by the GPU with very little overhead.

An interesting fact is that the granularity has no negative effect on the speed of the histogram generation. If at all, finer granularity creates less GPU stalls and is even faster. The only consideration is the amount of texture memory that is required for storing the histogram when many instances are to be queried.

The HIB method is very easy to implement and does not introduce CPU overhead, since all ID computation and pixel counting is offloaded to the GPU. It is worth noting that it is the only method that can be used to query the individual visibility of instanced geometry fully on the GPU. This is particularly tempting as – depending on the number of draw calls, shader cost and pipeline throughput – a tremendous increase in rendering performance due to instancing has been reported [Dudash, 2007].

6.3 Results and Applications

In this section we present timings and results of our method. We applied granular visibility queries to a shadow volume implementation using the GPU to extrude the objects' silhouettes, and to a displacement mapping technique intensively using geometry shaders.

	NVIDIA 280GTX			
#objs	nc	SAT	HIB	pred.
100	53	217	194	217
200	27	99	121	113
300	19	63	87	70
400	14	38	48	37
500	11	26	31	27

Table 6.1: *Average frames per second for the shadow volume example achieved with the SAT and HIB with instancing for a varying number of randomly placed objects. For comparison, we used Direct3D 10 occlusion predicates and no culling (nc). All culling techniques use a* 512^2 *resolution render target for determining visibility. The rendering resolution was* 1600×1050*, each object consists of 606 triangles. Table: Engelhardt and Dachsbacher [2009].*

6.3.1 Shadow Volume Culling

For testing our method in practical applications, we used it to speed up the shadow volume algorithm [Crow, 1977], which is best suited for rendering accurate shadows from point lights, but has also been extended to generate soft-shadows [Assarsson et al., 2003]. Geometry shaders allow the generation of the shadow volumes directly on the GPU which is beneficial for real-time applications, especially with dynamic or deforming geometry. In addition to the geometry shader, the rasterization of the shadow volumes is costly as well, as it consumes a lot of fill-rate. One solution to reduce this cost has been presented in [Lloyd et al., 2004] and a GPU-friendly version is described by Eisemann and Décoret [2006]. The culling of superfluous shadow volumes can be easily implemented using our methods. In Figure 6.1 (left images) we show an example scene where we determined the visibility of each object with respect to the light source in the geometry shader. Shadow volumes are excluded from rendering if an object is fully in shadow. Table 6.1 shows the rendering performance and compares it to predicated/conditional occlusion queries; Table 6.3 (page 90) details the CPU timings for the SAT method.

6.3.2 Image Space Culling: Displacement Mapping

In our second example we use granular visibility queries to speed up rendering by culling subregions of objects. This is possible, as SATs and HIBs do not only contain information about the number of visible pixels per object, but also of their spatial distribution.

culling method	full vis.	partial vis.	occluded
none	71	90	90
predicates, 594 × 12 triangles	25	47	149
predicates, 891 × 8 triangles	20	30	110
item buffer, 8 × 8 tiles	61	30	110
item buffer, 16 × 16 tiles	61	105	186
item buffer, 32 × 32 tiles	61	123	186
item buffer, 90 × 90 tiles	66	131	186
summed area table	57	89	112

Table 6.2: *Frames per second measured on a NVIDIA 280GTX for the lizard displacement mapping for full and partial visibility (Figure 6.1). The results indicate that HIBs do not suffer from many IDs. Timings for full occlusion are given to estimate the overhead of the methods. The static scene part consists of 134414 triangles, the lizard had 7132 triangles. Table: Engelhardt and Dachsbacher [2009].*

We use the granular visibility queries for a displacement mapping technique [Wang et al., 2004], specifically its adaptation in the DirectX10 SDK (shown in Figure 6.1), where the geometry shader extrudes triangles and splits the resulting prism into three tetrahedra. Therein gradients are constant and a fragment shader computes the intersection of a view ray and the height field. We modified the geometry shader such that it determines the bounding rectangle of each prism, and next, it determines if any pixel belonging to the object is visible within this rectangle. If no visible pixels are found, the geometry shader terminates without creating geometry. Please note that we conservatively extrude the triangle mesh in the first render pass (to create the SAT or HIB) by moving its vertices in normal direction such that it encloses the displaced surface. When using SATs the visibility determination computes the intersection with every query in the region set as described in Section 6.1.2. For the HIBs implementation we test for visibility in all overlapped screen tiles. Table 6.2 shows the measured rendering performances.

6.4 Discussion

Differentiation: Our method is meant to query the visibility of objects efficiently after rasterizing many objects, as it is common for all item buffer methods. That is, stop-and-wait algorithms, e.g. [Bittner et al., 2004], are orthogonal to our approaches. Similar to GVQs, a software implementation

#objs	overlap test	colouring	split & merge	conflicts	⌀error pixels	⌀error percent
100	0.01	0.05	0.1	10	1.6	2.3
200	0.35	0.17	0.8	50	1.7	2.6
300	0.6	0.33	3.7	116	2.7	4.4
400	1.1	0.5	11	166	6.3	9.8
500	1.6	0.7	23	249	7.4	11.6

Table 6.3: *Detailed timings in milliseconds of the CPU tasks when using the SAT approach in the shadow volume example. We also give the average relative and absolute pixel error of the queries. Table: Engelhardt and Dachsbacher [2009].*

of the hierarchical z-buffer method allows the culling of portions of objects. However, the HZB is bound to the fixed construction scheme as mip maps are, and thus requires many texture lookups to test for visibility of arbitrary regions, and counting visible pixels is impractical.

Accuracy: As expected the query results when using SATs show deviations which are due to the SAT resolution, but mainly due to approximative region sets (Section 6.1.3). However, we believe that this negligible for many applications, such as those using aggressive culling strategies. The conservative region sets return more pixels than actually visible and the deviation and the CPU load grows with the number of conflicts Table 6.3. Thus the SAT approach can be recommended for a moderate number of query objects only. Please note that multiple render targets, and thus more than 4 colour channels, can be used to reduce the number of conflict objects.

Non-binary visibility: Typically the visibility function for a pixel is of a binary nature – it is visible or not. Interestingly the SAT method does away with this restriction: Instead of writing purely coloured pixels to query textures, we can also output pixels with any brightness. We believe that this has various tempting applications, e.g. to estimate blocking from a transparent occluder, or to accumulate the tolerable visibility threshold as used with perceptual rendering methods [Drettakis et al., 2007]. A similar result can be obtained with the item buffer approach by storing a (scalar) value in addition to the ID per pixel. During histogram computation, we would sum up the values instead of incrementing the bin by one.

Method	Granularity	CPU Load	GPU Load	Instancing	Accuracy
HOQ	draw call	very low	render proxy^1	no	exact
POQ	draw call	very low	render proxy^1	no	no feedback
SAT	arbitrary sub regions	high	additional render target	no^2	approx.
HIB	fixed (screen) tiling	very low	additional render target	yes	exact

Table 6.4: *This table compares the properties and possibilities of natively supported visibility queries and our methods.* ^{1} *the scene has to be rendered twice to resolve mutual occlusion.* ^{2} *instancing is possible, if the instance transformations are evaluated on the CPU. Table: Engelhardt and Dachsbacher [2009].*

Querying the visibility on the GPU: By using predicates and standard occlusion queries (with a read back to the application) we can prevent geometry from being sent to the graphics pipeline at all, and this is obviously the fastest method for culling large chunks of geometry. The strengths of GVQs lie in the culling of individual primitives and the controlling of shader level of detail in (partly) occluded regions. Geometry shaders are often used to extrude geometry and subdivide triangles. The rendering performance does not only depend on the shader instructions but also on the amount of geometry that is output. GVQs can be used to determine local visibility information in order to omit primitives and shader output as demonstrated in our examples. If early-z culling becomes deactivated, e.g. due to shaders that output depth, GVQs can be used to avoid costly pixel shader executions: Geometry can be omitted in geometry shaders, or “culled” in vertex shaders by transforming it outside the view frustum to prevent rasterization. As mentioned before, it is imaginable to estimate the number of occluded pixels per primitive to obtain a fractional visibility and by this control the accuracy or detail in pixel shader computations. Table 6.4 summarizes properties of the different query mechanisms.

6.5 Conclusions

We presented two methods for granular visibility queries which allow us to determine the visibility of objects and subregions of objects in shader programs on the GPU. We demonstrated how costly rendering techniques, in particular these relying on geometry shaders, gain significant performance increases. We also introduced a hierarchical item buffer which enables efficient visibility queries for instanced objects and subregions simultaneously. This is important for many rendering techniques as geometry is quickly excluded from being processed by the GPU pipeline. This is especially important for rendering techniques that themselves perform heavy geometry processing, such as direct rendering of octahedron environment maps (cf. Chapter 7). Granular visibility queries and non-binary visibility functions are particularly interesting for more involved rendering methods, such as real-time perceptual rendering and level of detail control. We believe that our method also has potential use in other GPU rendering techniques.

CHAPTER 7

Octahedron Environment Maps

In this chapter we introduce the *octahedron environment map (OEM)* as a scheme to represent environment lighting or light fields for real-time applications as well as offline rendering systems. [1] Beyond that, environment maps can be generated at each shading location seen from the camera for GPU-based final gathering (cf. Chapter 8). Since any kind of information can be associated with environment maps, they can also store an omnidirectional depth map for visibility from point light sources. This is particularity useful for techniques that build on virtual point lights, such as our approximate bias compensation (cf. Chapter 10) and screen-space bias compensation (cf. Chapter 11). Environment maps are generated projecting the entire environment surrounding a point in space into a single quadratic or rectangular image. This is achieved on the foundation of the octahedron as a parametrization for the environment. We discuss the creation of OEMs by the means of two different projection schemes and carry out an analysis considering rendering performance for interactive applications as well as quality of representation. To conclude we discuss applications that benefit from this parametrization over other approaches (cf. Chapter 5).

7.1 The Octahedron Environment Map

The octahedron as a possible parametrization scheme for the spherical domain has already been used by Praun and Hoppe [2003], who utilized the 1-to-1 mapping of the spherical domain to a 2D texture to store spherical geometry images. We examine the octahedron with regard to the use

[1] This chapter is based on our work [Engelhardt and Dachsbacher, 2008] presented at the Vision, Modeling and Visualization Workshop 2008 (VMV'08).

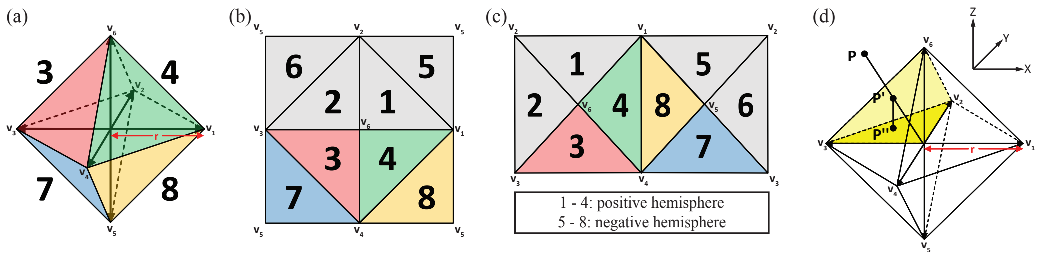


Figure 7.1: *The octahedron (a) can be unfolded and packed into a single quadric (b) or rectangular texture (c). Planar projection (d) first projects a point* **p** *onto the octahedron faces, obtaining* **p**$'$*, and afterwards orthogonally onto the XY-plane. This yields the final projection* **p**$''$*. Illustration: Engelhardt and Dachsbacher [2008].*

as environment maps for real-time rendering applications, including the generation of and look-up to octahedron environment maps (OEMs) with graphics hardware. Each side of the octahedron represents one octant of directions, and we describe two projections to map directions to points on the sides. Further, the octahedron sides may be arranged differently in two-dimensional textures. In the following, we assume that all coordinates are given relative to the octahedron's coordinate system as illustrated in Figure 7.1d. This of course can be easily achieved setting up the appropriate transformations during rendering.

7.1.1 Planar Projection

The first projection scheme that we analyse uses a projection similar to a spherical projection. A point $\mathbf{p} = (p_x, p_y, p_z)^T$ lies on the sphere with radius r, if $p_x^2 + p_y^2 + p_z^2 = r^2$. Similarly, for a octahedron with vertices $(\pm r, 0, 0)^T$, $(0, \pm r, 0)^T$, $(0, 0, \pm r)^T$, a point resides on the surface of the platonic solid, if $|p_x| + |p_y| + |p_z| = r$. Hence, the projection $\mathbf{p}$ of a point onto the platonic solid is derived as

$$\mathbf{p}' = \frac{\mathbf{p}}{|p_x| + |p_y| + |p_z|}. \qquad (7.1)$$

We denote this projection *planar projection*. To unfold the octahedron into a quadratic texture and to obtain texture coordinates, points on the positive hemisphere, i.e. $p_y > 0$, are orthogonally projected onto the XZ-plane (Figure 7.1). The lower hemisphere is unfolded by splitting all edges adjacent to $(0, 0, -r)^T$ ($\mathbf{v}_5$ in Figure 7.1a). Hence two-dimensional texture coordinates $\mathbf{p}_q'' \in [-1; 1]^2$ are obtained by the following transformation:

$$\mathbf{p}''_q = \begin{cases} \begin{pmatrix} p'_x \\ p'_y \end{pmatrix} & p'_z \geq 0 \\ \begin{pmatrix} \sigma(p'_x)(1-\sigma(p'_y))p'_y \\ \sigma(p'_y)(1-\sigma(p'_x))p'_x \end{pmatrix} & p'_z < 0 \end{cases}, \tag{7.2}$$

with $\sigma(x)$ the sign function:

$$\sigma(x) = \begin{cases} 1, & x > 0 \\ 0, & x = 0 \\ -1, & x < 0 \end{cases}.$$

Our second unfolding approach maps the octahedron to a rectangular texture with an aspect ratio of 2 : 1, as shown in Figure 7.1c. First equation 7.1 is applied and next points are projected orthogonally on the XY-plane, and the coordinates from the 2 hemispheres are mapped into the layout by transformations.

$$p''_r = \begin{cases} \begin{pmatrix} p'_x - p'_y - 1 \\ p'_x + p'_y \end{pmatrix} & p'_z \geq 0 \\ \begin{pmatrix} p'_x - p'_y + 1 \\ p'_x + p'_y \end{pmatrix} & p'_z < 0 \end{cases} \tag{7.3}$$

According to the layout mapping $\mathbf{p}''_r \in [-2; 2] \times [-1; 1]$. For rendering, the unnormalized texture coordinates obtained from either projection scheme have to be normalized.

7.1.2 Perspective Projection

Cube maps [Greene, 1986] can be generated rasterizing the scene for each cube face under a perspective projection with a field of view of 90°. The same approach can be applied to generate octahedron environment maps. The scene can be rendered once for each octahedron side, each set up with its own projection matrix. The frusta in this case are of tetrahedral shape and have a triangular view port. Since graphics hardware does not support clipping against such viewports natively, clipping has to be performed either manually, or in image space.

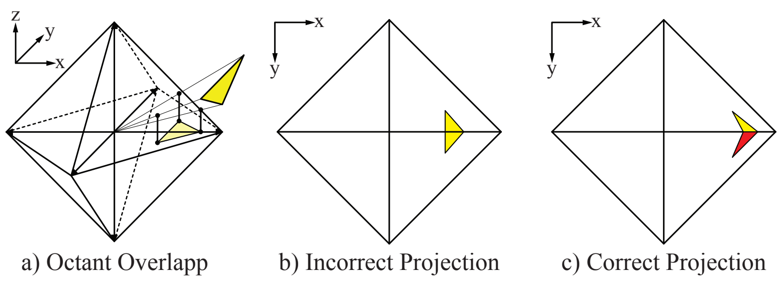


Figure 7.2: *(a) The triangle overlaps two octants. (b) When ignoring the octant-overlap rasterization of the projected triangle yields incorrect results. (c) Splitting the triangle before projection at the principle planes, ensures correct rasterization results. Illustration: Engelhardt and Dachsbacher [2008].*

The look-up to octahedron maps using these projections work analogously to cube maps: First the octant of a look up vector has to be determined, and next the projection transformation and the mapping to the octahedron layout (as described above) is computed.

We examined this parametrization for the sake of completeness, but compared to a cube map, there are no advantages to be expected (8 projections instead of 6, plus additional clipping). Thus, in the remainder of this chapter, we focus on the planar parametrization, which is simpler and thus beneficial for some applications.

7.2 Implementation

In this section we describe the creation of octahedron environment maps on graphics hardware. Look-ups are also discussed. In particular, care must be taken when filtered texture look-ups are to be used with OEMs.

7.2.1 Direct Rendering of Octahedron Maps

For planar projections we cannot utilize matrix operations, as they do not support absolute values that are required for this projection (cf. Equation 7.1). Nonetheless, planar projections can be otherwise realized thanks to programmable graphics hardware. Unfortunately, triangles overlapping octant boundaries cannot be handled correctly: Transformations and projections on graphics hardware are performed per-vertex and scanline conversion or rasterization (assuming linear triangle edges) produces wrong results whenever a triangle intersects more than one octant (illustrated in

```
// projection onto octahedron
d /= dot(1, abs(d));

// out-folding of the downward faces
if(d.y < 0.0f)
   d.xy = (1-abs(d.zx)) * sign(d.xz);

// mapping to [0;1]x[0;1] texture space
d.xy  = d.xy * 0.5 + 0.5;
color = tex2D(octaMap, d.xy);
```

Figure 7.3: *Shader pseudo code which computes the texture look-up coordinate for the quadratic layout corresponding to a direction vector* **d***. Illustration: Engelhardt and Dachsbacher [2008].*

Figure 7.2b). This is somewhat similar to the projection errors that occur rasterizing a paraboloid environment map [Heidrich and Seidel, 1998]. Similarly, we can keep the error low using finely tessellated geometry, but in order to produce correct results, additional effort is necessary.

Linear interpolation is correct within each octant, i.e. we can solve the problem correctly by splitting triangles at each octant boundaries. We can leverage this by using geometry shaders: A triangle is split at the XY-, YZ-, ZX-planes into smaller triangles (up to 27), such that each output triangle resides in only one octant. After clipping, the projected coordinates are computed and the affine transformation according to the layout (quadratic or rectangular) is applied.

7.2.2 Texture Look-ups

Unfiltered texture look-up: The texture coordinates for the look-up into an octahedron map for a given direction vector can be computed analogously to the aforementioned projection and layout transformation. The unfiltered look-up is very efficient when used together with the quadratic layout. Computing the texture coordinates for a direction vector **d** given in the octahedron coordinate system can be achieved with only a few shader instructions as demonstrated in Figure 7.3

Filtered texture look-up: For a mip map look-up, we need to consider two aspects. First, when determining the required mip level for a texture look-up, the decision is based on the extend of the quadrilateral region spanned by the screen space pixel cell transformed into texture space. These extends are derived from the texture coordinates and its partial derivatives [Microsoft Corporation, 2010]. Please note, that for per-pixel computed reflections, the mip level is typically computed manually depending on surface curvature or glossiness. We denote the partial derivatives in texture space by dX and dY. The mip-map level l is then derived by:

$$\begin{aligned} e &= \max(\langle dX, dX\rangle, \langle dY, dY\rangle) \\ l &= \log_2 \sqrt{e} \end{aligned} \tag{7.4}$$

In the case of the octahedron map the quadrilateral region might cross two adjacent octants which have been separated due to the unfolding. For the quadratic layout, this case occurs when the extents of a pixel in texture space overlaps more than one octant. A possible situation is depicted in Figure 7.4 (centre) – similar configurations occur for the rectangular layout as well.

Thus, we need a special treatment when a cell spans to disjoint regions in the texture space. Fortunately, the symmetry of the octahedron allows us to reflect the coordinates of pixel cells in the lower to the upper hemisphere, thus resulting in a contiguous quadrilateral which is then used to compute the correct mip level (Figure 7.4, right). In practice, we compute the mip level of both, the original and the mirrored cell, and use the minimum of both values.

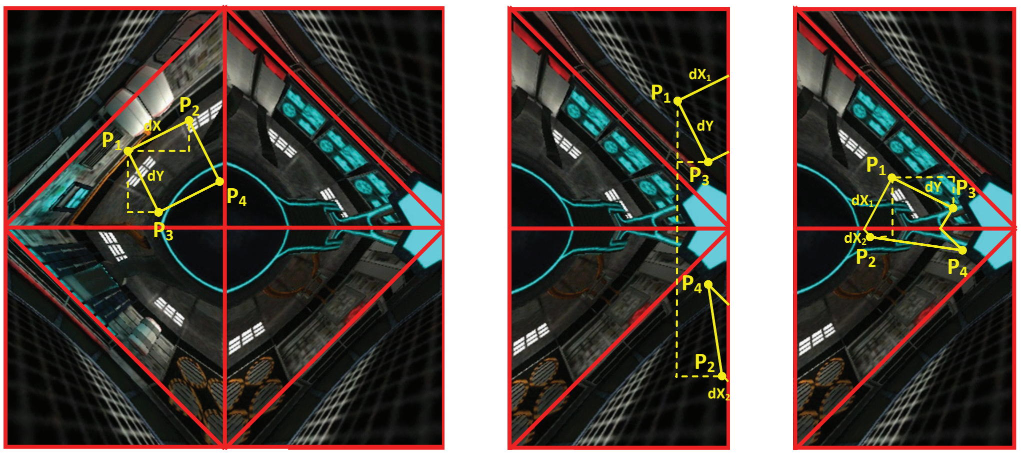


Figure 7.4: *The quadrilateral region in texture space corresponding to a pixel cell in the negative hemisphere is mapped to disjoint regions (centre). This yields incorrect mip-levels, however, by exploiting symmetry we reflect at the XY-plane and obtain the correct area estimate for mip map level determination (right). Illustration: Engelhardt and Dachsbacher [2008].*

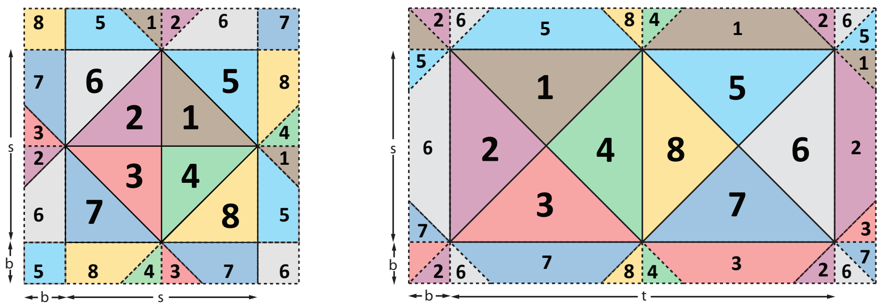


Figure 7.5: *Quadratic OEM textures of dimension s × s and rectangular OEM textures of dimension t × s are both extended with a texture border of width b to ensure correct linear interpolation. For rectangular layouts texture wrapping in the t-dimension can also be used instead of an explicit border. Illustration: Engelhardt and Dachsbacher [2008].*

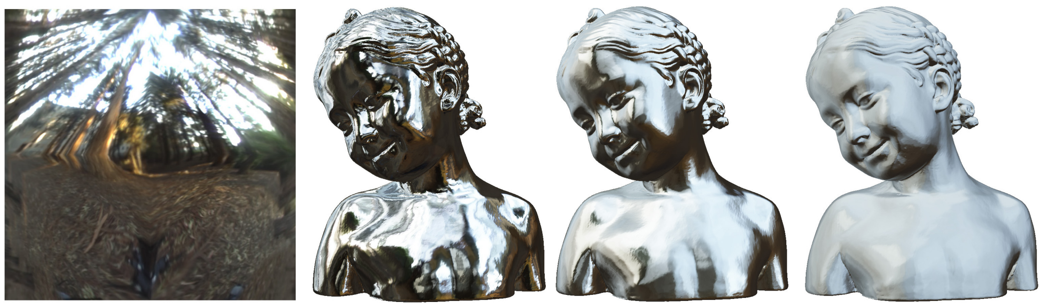


Figure 7.6: *This image shows screen-shot of a rendering with SATs created from octahedron environment maps to adjust surface glossiness at real-time. The creation of the* 512 × 512 *pixel SAT takes about* 3.3*ms on a NVIDIA 8800GTX. Illustration: Engelhardt and Dachsbacher [2008].*

The second aspect to considered is that due to unfolding a wrap-around access to the texture does not yield correctly bilinearly interpolated colour values for texture filtering (this applies to parabolic, and to some extend to cube maps as well). This can best be solved introducing a 'safety-border', that is duplicating pixels as shown in Figure 7.5. The border ensures that samples for bilinear interpolation and mip-mapping are available. For rectangular layouts, hardware assisted wrap-around access works only in one dimension (cf. Figure 7.5).

Please note, that the maximum correctly represented mip level depends on the width of the safety-border. We also used octahedron maps with summed area tables for rendering glossy reflections [Hensley et al., 2005]. In this case we used a safety-border width determined by the maximum

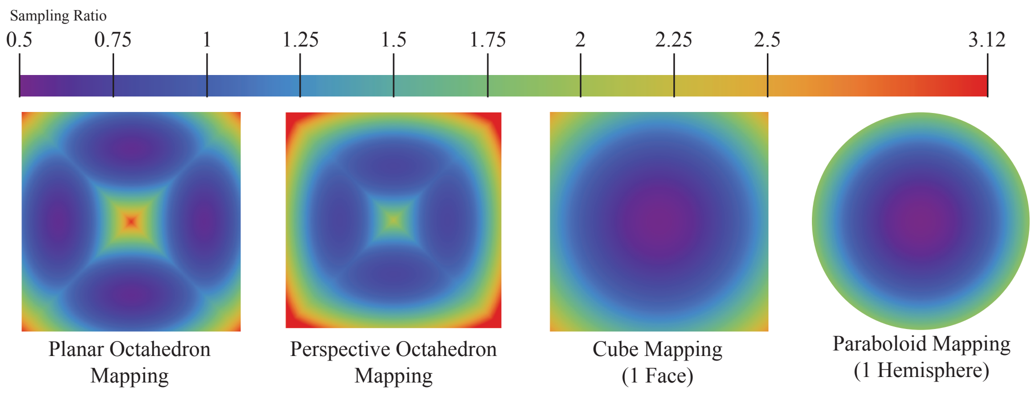


Figure 7.7: *Visualization of the sampling ratio compared to uniform sampling over one hemisphere. Dark blue and purple regions indicate areas of undersampling, while cyan to red mark regions of oversampling.*

glossiness of the BRDF (Figure 7.6 shows an example). Please note, that for accurate BRDF modeling, multiple SAT queries are required which holds for all spherical parametrizations and (pre-)filtered environment mapping.

7.3 Results

In this section we present results for our OEM. We evaluate quality and compare to other environment mapping techniques. Further, we asses rendering speed for planar and perspective projection, and discuss our results, and highlight advantages of our representations and possible applications.

7.3.1 Sampling

In order to evaluate the sampling quality of the octahedron map, we compute the solid angle covered by each pixel (relative to the solid angle of a sphere over the total number of pixels). Comparisons of the octahedron parametrizations to a cube map [Greene, 1986] and the paraboloid map [Heidrich and Seidel, 1998] are shown in Figure 7.7. The highest sampling rate is close to the octahedron vertices but stays within an acceptable range across the entire surface. As expected, the planar projection yields a more balanced sampling than the perspective projection; the overall quality is comparable to either cube and paraboloid mapping.

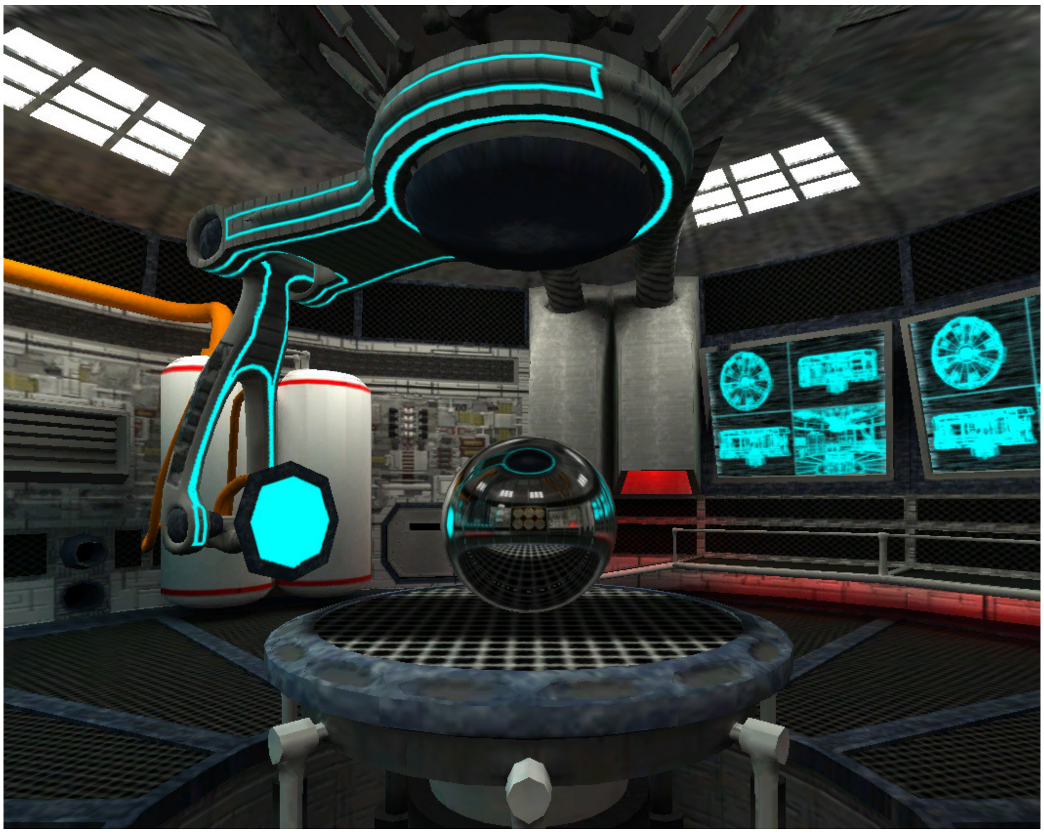


Figure 7.8: *This image shows our test scene for dynamic environment map generation, which consists of approximately* 105*k triangles. Illustration: Engelhardt and Dachsbacher [2008].*

7.3.2 Direct Rendering

We further measure the rendering performance for an adequately complex scene (Figure 7.8) creating three environment maps of approximately equal pixel count. We directly render a 512 × 512 quadratic layout *(QL)* OEM, a rectangular layout (*RL*) OEM of two 362 × 362 textures and a cube map of six 209 × 209 textures. For careful performance evaluation, we have implemented different rendering approaches which are mentioned in the following.

Strict split: We have implemented both the planar and perspective projection with explicit clipping in the geometry shader stage as described in Section 7.2.1. We denote this approach *strict split (S-Split)*.

Clip mask: As mentioned in Section 7.1.2 the clipping for perspective OEMs can also be implemented discarding pixels in the pixel shader that do not fall into the triangular viewport. We call this method *Clip Mask*.

XY-split: For the rectangular layout we present results for the planar projection and only examine two less accurate clipping techniques: The clipping performance of the geometry shader (GS) heavily depends on the maximum amount of vertices that can be generated and output to the succeeding pipeline stages. This proved to be the bottleneck in our implementation of the *strict split* (see above), for clipping potentially creates 27 triangles which amounts to 91 vertices in total. To relax the pressure on the GS output we opt to generate less vertices in each GS invocation. Exploiting the fact that the rectangular OEM layout maps both hemispheres to contiguous regions we only split triangles that overlap the XY-plane. Although this creates three triangles we can use triangle strips and output at most 7 vertices in total.

Edge strip: The second approach, we call *Edge-Strip*, proceeds with the vertices output by the XY-split in both half-spaces (positive and negative z-axis) independently. Those vertices form a convex polygon (per half-space) whose edges are split by the remaining split planes (Figure 7.9, *Edge Split*). Afterwards two naïve triangle strips are created from the so obtained points (Figure 7.9, *Triangle Strip*). Please note, that while the naïve stripping preserves the correctness of the projection on both polygon boundaries, it does not guarantee correct interpolation, e.g. of texture coordinates, within the triangles of those strips. Of course this is not correct, but rather tolerable than a wrong primitive shape. This approach outputs a maximum of 15 vertices in total.

7.3.3 Applications and Advantages

Despite the difficulties introduced with direct rendering of OEMs, they provide two unique features. First, OEMs have the ability to generate summed area tables (SATs) over the the entire spherical domain. When creating SATs on GPUs, e.g. with the method by Hensley et al. [2005], the input and output parametrization can be chosen independently. This is because multiple render passes are required for the generation and the first one can include a reparametrization, i.e. also easily renderable cube or paraboloid maps can be used as input. Furthermore, a look-up to an OEM is an efficient means to make directional data available to all shader stages.

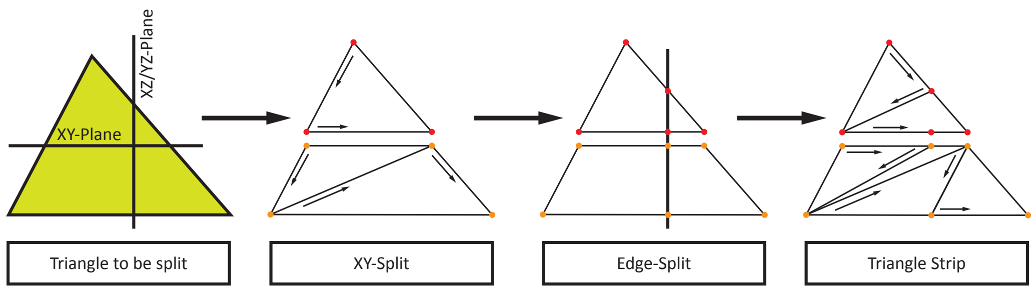


Figure 7.9: *This alternative triangle splitting scheme generates primitives with correct boundaries in the OEM. Less vertices are created, however the interpolation of the interior is not correct. Red vertices are mapped to the upper half-space ($y \geq 0$) while orange vertices are mapped to the lower half-space ($y < 0$). Illustration: Engelhardt and Dachsbacher [2008].*

The octahedron parametrization also provides compact storage when multiple (hemi-)spherical domains are to be stored, e.g. (hemi-)spherical shadow maps for instant radiosity methods [Laine et al., 2007; Ritschel et al., 2008] or for sparse-sampling of scene radiance [Greger et al., 1998]. Instead of parabolic parametrizations which waste up to 20 percent of texture space, OEMs can be packed tightly into texture atlases utilizing all available space.

As OEMs, latitude-longitude environment maps [Blinn and Newell, 1976] enable the parametrization of the spherical domain over a rectangular texture, and consequently, they would also enable to compute SATs, provide compact storage, and look-up in vertex and geometry shaders. However, sampling is severely distorted in polar regions. Beyond that, the mapping shows a singularity at each pole, and the look-up requires trigonometric, and thus expensive, operations. Finally, rendering to latitude-longitude maps is not possible on GPUs without severe errors or high tesselation, due to the non-linearity of the projection.

7.4 Conclusions

We examined octahedron environment maps as an alternative for environment map rendering which represent the entire spherical domain of directions in a single quadratic or rectangular texture while providing efficient texture look-ups. Although the planar projection and unfolding transformations are rather simple, and provide good sampling (comparable to the cube map), the direct rendering is not yet suitable for interactive applications due to the geometry shader performance.

EnvMap	FPS[s]
Planar Proj. QL, *S-Split*	16
Planar Proj. RL, *XZ-Split*	34
Planar Proj. RL, *Edge-Strip*	3
Perspective Proj. QL, *S-Split*	16
Perspective Proj. QL, *Clip Mask*	95
Cube Map	250

Table 7.1: *Rendering performance of our test scene for a screen resolution of* 1920 × 1200*. All timings were taken on an Intel Q6600 Core2Duo processor with 2.4GHz, a GeForce 8800GTX running Vista x64 with 4GB of main memory. Table: Engelhardt and Dachsbacher [2008].*

Further, OEMs offer benefits not known from other techniques, namely the ability to create spherical summed area tables and tight-packing into texture atlases. Especially the latter allows us to store omnidirectional shadow maps efficiently. Efficient rendering is also feasible when alternative rendering methods are used [Ritschel et al., 2008]. OEMs an be directly applied to imperfect shadow maps which are particularly useful in global illumination techniques that build on virtual point light sources (VPL), such as our approximate bias compensation method (cf. Chapter 10) and screen-space bias compensation algorithm (cf. Chapter 11). Both algorithms build on virtual point light sources in order to approximate multiple scattering and indirect illumination. Furthermore, OEMs can also be used to store indirect illumination at shading points and are therefore an ideal candidate for micro-rendering (cf. Chapter 8) which relies on directional parametrizations that fit into a single texture or image.

CHAPTER 8

Micro-Rendering

FINAL Gathering is a method for accurately computing indirect illumination for surfaces visible to the camera. There it evaluates the shading integral numerically and in order to do so, incident light from other surfaces in the environment must be determined. Therefore ray tracing is commonly used to sample the visible surfaces in the environment of the gather location, where reflected light is typically computed using a less accurate GI algorithm, such as photon mapping [Jensen, 1996], or radiosity [Lischinski et al., 1993]. Ray tracing is conceptually simple, but computationally expensive, as often hundreds of so called gather rays have to be traced per gather location for high-quality results, making it infeasible for interactive applications. Alternatively, we can generate low-resolution environment images (e.g. OEMs) at each shading point employing GPU rasterization, but the linearity of it forbids importance sampling and setup costs become predominant.

We opt to lift these restrictions and introduce *micro-rendering (MR)*, an efficient, scalable GPU-based method, that achieves high-quality final gathering for dynamic scenes at interactive framerates[1]. Christensen [2008] demonstrated that point-based representations of the scene are suitable for global illumination algorithms. Similarly, we make without ray tracing and employ a point-based rendering technique to rasterize a hierarchical point representation of the scene into a micro-buffer. The latter is additionally importance-warped and thereby accounts for BRDF importance sampling, which is crucial for glossy surfaces. For further acceleration, we combine MR with GPU-friendly bilateral upsampling to reduce the sampling rate of gather locations in image space and demonstrate different applications with multiple indirect bounces and final gathering for photon mapping.

[1]This chapter is based on our joint work presented at SIGGRAPH Asia 2009 [Ritschel et al., 2009a]

8.1 Point Hierarchies for Final Gathering

As mentioned in the introduction, micro-rendering employs a hierarchical, point-based representation of the scene which is rasterized into micro-buffers. The advantages of such representations are twofold. On the one hand, point-based rendering has lower setup and rasterization costs for low image resolutions compared to conventional triangle rasterization. On the other hand, efficient level-of-detail rendering is easily incorporated [Dachsbacher et al., 2003] based on simple point selection criteria, as are used by QSplat [Rusinkiewicz and Levoy, 2000]. Similar to QSplat, we construct a bounding volume hierarchy using bounding spheres for the entire scene. The leaf nodes store surface elements together with their respective bounding sphere. A surface element is essentially an oriented disc with centre position $\mathbf{p}$, normal $\mathbf{n}$, and radius r. Internal nodes store an enclosing bounding sphere for all their children, i.e. the entire collection of surfels, or surface samples, they represent.

Rasterizing a point hierarchy into a micro-buffer traverses the hierarchy beginning with the root node. The original QSplat [Rusinkiewicz and Levoy, 2000] algorithm projects the bounding sphere of each traversed node into screen space and accepts it when its projected size in the image is smaller than a given threshold (e.g. a pixel). Otherwise the node is further refined by descending the hierarchy. Micro-rendering proceeds in a similar manner but replaces the 'size-in-screen-space'-test with a test that is based on the solid angle a node subtends. This ultimately allows us to incorporate BRDF importance sampling defining a warped projection scheme, mapping the micro-buffer pixels to directions where the BRDF has significant contribution.

8.1.1 Point Hierarchy Construction

The point hierarchy is constructed generating surface elements (surfels) in an offline preprocessing step first. For surfel generation we use a best candidate sampling pattern to distribute points proportional to the surface areas of the triangles. These points $\mathbf{p}$ are the surfel centres, and the radius is derived from the initial point density. Because this density is constant, each surfel is created with the same radius. Additionally we store the triangle index and the barycentric coordinates with each surfel, which allows us later to recompute position, normal, and surfel radius under deformation [Ritschel et al., 2008]. The latter is recomputed to compensate for the thereby induced varying point density.

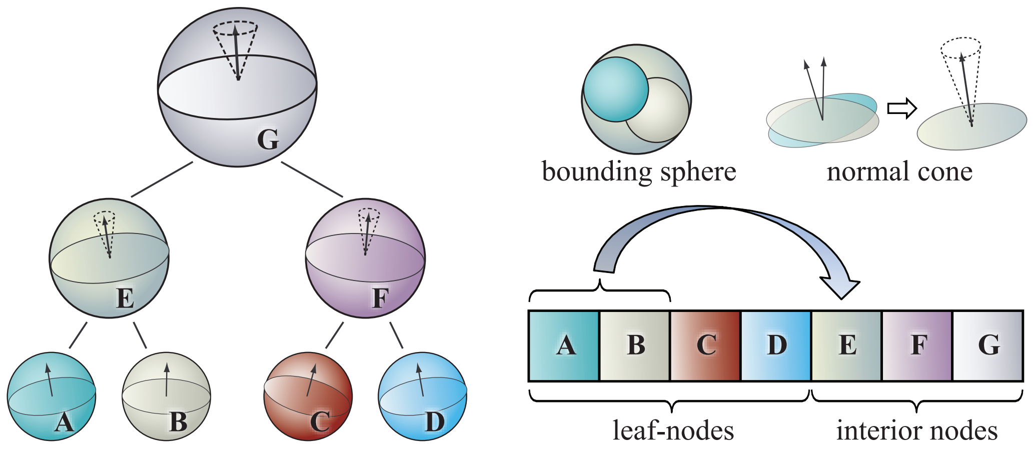


Figure 8.1: *The surfaces of the scene are stored as point hierarchy. We always use complete binary tries, because it allows us to traverse and update the structure efficiently at run-time Illustration: Ritschel et al. [2009a].*

In the second step we build the point hierarchy, in which the previously created surfels become the leaf nodes (Figure 8.1). We construct the hierarchy using a binary space partition scheme dividing the space at the coordinate axis along which all points have maximum extent. Afterwards, a splitting plane is placed, so each produced subspace contains an equal number of points. Then the scheme continues recursively in the same manner until each subspace contains only two points. In total this yields construction costs of $O(nlog_2 n)$ for n points. In our cases, the number of points is always a power of 2, and hence we always obtain complete trees. This has one noteworthy advantage: We can place nodes and hierarchy levels consecutively in memory (Figure 8.1) and thus avoid storing additional skip-pointers for traversal in each node. These can be computed on-the-fly.

Consider the example in Figure 8.1. Nodes A to D are the leaf nodes after sorting. The tree is complete, and nodes A and B are children of E, and so on. When constructing the hierarchy, we compute the minimum bounding sphere for enclosing all children, as well as the cone of normals (stored as direction plus cone angle), and lay out nodes and hierarchy levels consecutively in memory. The normals and normal cones are used for lighting computation and back-face culling during the point hierarchy traversal (Section 8.1.2).

Deforming and moving geometry: Since we have stored a triangle index and barycentric coordinates with each point in the hierarchy, we can restrain from completely reconstructing it under deforming geometry. Instead we only update the per-node data at run-time at the beginning of each frame. This is a bottom-up process: First, we recompute the normal and position of each node, and second, we successively propagate these changes upwards in the hierarchy. During propagation we merge two nodes at a time and thereby adjust their minimum enclosing bounding sphere and its cone of normals. Because we have to update each leaf node, update costs sum up to $O(n)$ for n leaves. This scheme cannot handle moving geometry, and we have to use separate hierarchies for such objects.

8.1.2 Hierarchical Cuts for Hemispherical Rasterization

Final gathering evaluates the shading integral to compute indirect illumination at each point $\mathbf{p}_c$ visible to the camera. For this purpose, it must determine the incident light reflected from the environment visible through the upper hemisphere at $\mathbf{p}_c$. Typically, the reflecting surfaces of the environment are determined using ray tracing at $\mathbf{p}_c$, sampling the visible surfaces in the environment with gather rays. For good results, several hundred gather rays are often sufficient. Similar results can be achieved rendering low resolution (24×24) images of the environment using hemispherical projections (cf. Chapters 5 and 7), but as stated earlier, this becomes inefficient for rasterization due to repeated geometry processing and triangle setup.

In this section we describe how point hierarchies can efficiently be used to render a low-resolution image of the environment under hemispherical projections, the so called micro-buffer and how it is used to compute indirect illumination in the spirit of final gathering.

Micro-buffers: The environment at a surface point $\mathbf{p}_c$ is visible in each direction ω. To record the environment in a two-dimensional image, called the micro-buffer, we define a mapping $\Phi(x, y) = \omega$ relating a pixel (x, y) to a gather direction ω. We denote the solid angle subtended by the pixel in the micro-buffer under this mapping as $\Omega(x, y)$ (see Figure 8.2). Although any hemispherical mapping is possible, we use our own importance-warped mapping that is described in Section 8.1.3. To keep storage costs for the micro-buffer low, and in order to correct shortcomings of the hierarchy cuts (see below), we store an index to the nearest visible surface and its distance at every pixel, instead of explicitly storing incident illumination. Note this is similar to geometry-buffers [Saito and Takahashi, 1990]. Visibility is deter-

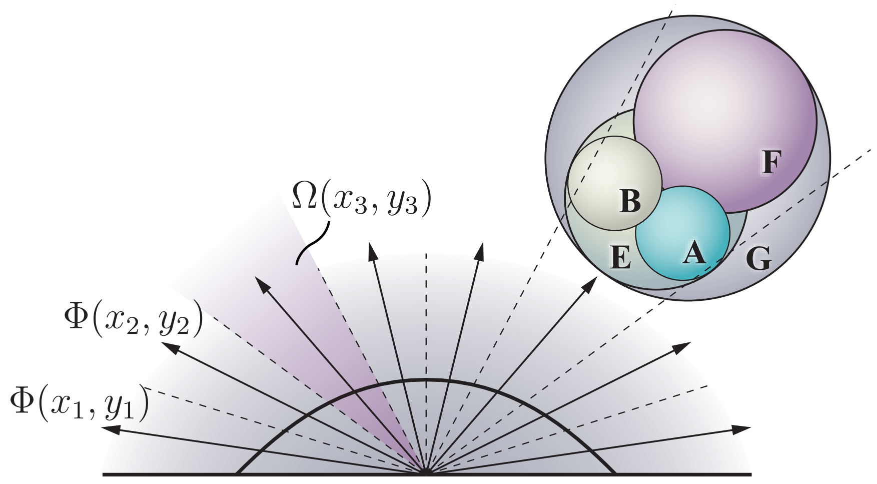


Figure 8.2: *Every pixel (x_i, y_i) of a micro-buffer is mapped onto a direction $\Phi(x_i, y_i)$. Under this mapping, the pixel subtends a solid angle $\Omega(x_i, y_i)$. While traversing the hierarchy, nodes are refined until a node does not project onto more than one pixel. Illustration: Ritschel et al. [2009a].*

mined first, then shading is executed (cf. Chapter 2.2.2). As stated earlier, a micro-buffer is typically small, ranging from 8×8 to 24×24 pixels in our examples.

Point hierarchy cut: In order to determine the visible surfaces in the environment of $\mathbf{p}$, we compute a point hierarchy cut using a depth-first search. The search starts at the root node and for each node we evaluate whether it will be refined, and the search continues, or whether it is sufficient, and it is accepted in the cut. This decision is based on a simple selection criterion. We compute the direction ω_i to the node's centre and the solid angle Ω_i that it subtends. Using the inverse hemispherical mapping, $\Phi^{-1}(\omega_i)$ we determine the pixel (x_i, y_i) the direction ω_i maps to:

$$(x, y) = \Phi^{-1}(\omega_i).$$

If we detect that the projection will span more than 1 pixel, i.e.

$$\Omega_i > \Omega(\Phi^{-1}(\omega_i)),$$

we will proceed with the child nodes. Otherwise we compare the node's distance to the existing depth value in the micro-buffer at pixel (x_i, y_i). In case the node passes the depth test, i.e. it is closer to $\mathbf{p}_c$ than the one in the buffer, the index and depth values in the micro-buffer are updated.

On-demand ray casting: It may occur that leave nodes project onto several pixels with possible distorted shapes. Unfortunately, they cannot be further refined. One possibility is to project them only into the pixel $(x_i, y_i) = \Phi^{-1}(\omega_i)$ that corresponds to the direction towards the node's centre. However, we have found that this leads to undesirable artefacts in the final rendering (Section 8.3). Hence we opt to resolve visibility for such nodes exactly. Fortunately, this can easily be computed using ray casting. To this end, we append such nodes to a post-traversal list which is processed after computing the cut. For each pixel in the micro-buffer, we obtain the according direction by the hemispherical projection scheme, define a ray, compute the intersections with all nodes in the post-traversal list, and update the micro-buffer accordingly. From this we obtain an accurate micro-rendering with water-tight surfaces (Section 8.3).

Convolution: After ray casting, visibility determination is complete, and the micro-buffer stores indices to each visible node in the point hierarchy for every pixel. The next step is to compute incident radiance $L(\omega_i)$ for each pixel in the micro-buffer. This in fact is the radiance reflected from the visible nodes. Reflected radiance can be computed using different approaches as we show in Section 8.3. For example, we can use primary and secondary light sources (VPLs) with shadow maps for this purpose, but we can also precompute reflected radiance using photon mapping (Figure 8.8) if the nodes are assumed to be diffuse. Then reflected radiance can be stored within the point hierarchy itself [Christensen, 2008].

After reflected radiance from the visible nodes has been gathered, it is weighted with the respective solid angle of the according pixel in the micro-buffer, multiplied with the BRDF, and summed up, yielding indirect illumination at the visible surface points $\mathbf{p}_c$.

8.1.3 BRDF Importance Sampling

So far we have not specified the mapping $\Phi(x, y)$ that relates a pixel to a direction ω. Many mappings can be thought of, for instance Nusselt's analog, which is also known in radiosity [Cohen et al., 1993]:

$$\Phi(x, y) = \left(x, y, \sqrt{1 - x^2 - y^2}\right)^T .$$

Unfortunately, such a simple approach bears a problem. For glossy surfaces, much of the information in the micro-buffer is virtually irrelevant. This is due to such BRDFs which reflect incident light mostly within a small solid

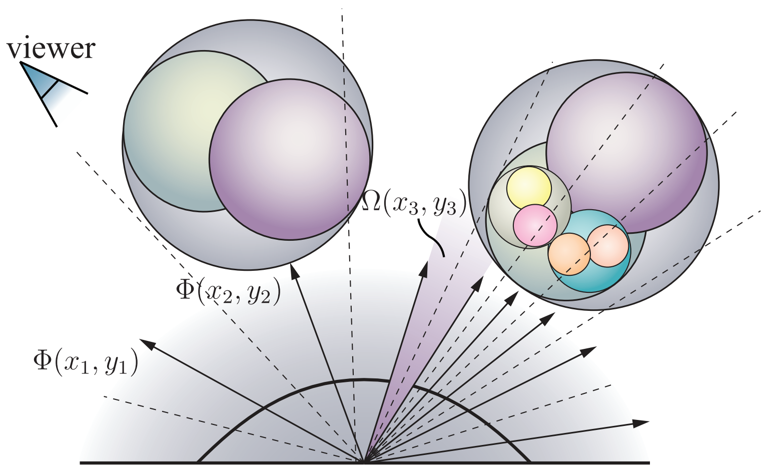


Figure 8.3: *Using a warping function $\Phi(x_i, y_i) = \omega_i$ we map pixels in the micro-buffers to directions distributed according to the BRDF. Hence the solid angle $\Omega(x_i, y_i)$ varies as well. Illustration: Ritschel et al. [2009a].*

angle, and consequently only a few pixels in the micro-buffer have a significant contribution to the reflected illumination, i.e. most light reflected towards the viewer stems from only a small number of pixels.

Ray tracing avoids this issue by BRDF importance sampling. This technique shoots rays preferably into directions from which high contributions to the reflection are expected according to the BRDF (cf. Chapter 4.2).

We opt to mimic this behaviour, i.e. our goal is to define a mapping that reserves more pixels for direction for which the BRDF has high contribution. This is ultimately achieved, defining an appropriate mapping $\Phi(x, y)$. For a given point $\mathbf{p}_c$ in the scene, we know the BRDF $f_r(\mathbf{x}, \omega, \omega_i)$ and the viewing direction ω_o. By keeping $\mathbf{p}_c$ and the viewing direction ω_o fixed, we obtain the 2D light-dependent slice $f_r^{\omega_o}(\omega_i) = f_r(\mathbf{p}_c, \omega_o, \omega_i)$.

Additionally we assume a local coordinate system at $\mathbf{p}_c$, where the z-axis is the normal and hence we can parametrize ω_i by its x- and y-coordinates, and thereby the 2D light-dependent slice of the BRDF: $f_r^{\omega_o}(\omega_{i,x}, \omega_{i,y})$. This can then be turned into a 2D probability density function (PDF)

$$p(\omega_{i,x}, \omega_{i,y}) = c f_r^{\omega_o}(\omega_{i,x}, \omega_{i,y}),$$

with the normalization constant

$$c^{-1} = \int\int f_r^{\omega_o}(\omega_{i,x}, \omega_{i,y}) d\omega_{i,x} d\omega_{i,y}.$$

From the PDF, the cumulative marginal and conditional distributions $M(\omega_{i,x})$ and $C(\omega_{i,y}|\omega_{i,x})$ can be derived. By the means of the corresponding inverse

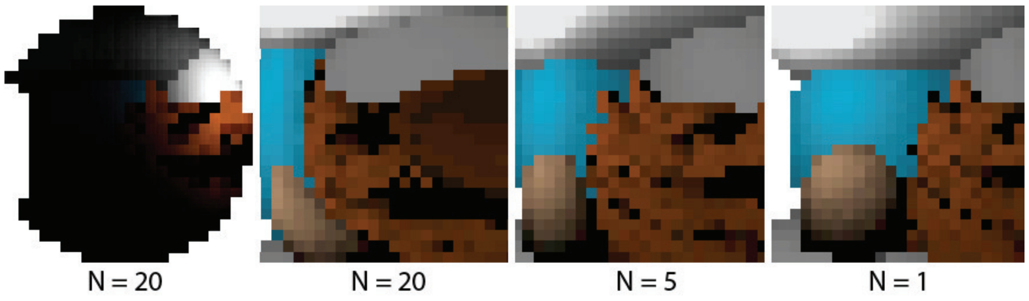


Figure 8.4: *Micro-buffer for a glossy BRDF. From left to right: standard hemispherical mapping and importance warped mappings for (Phong) $N = 20, 5, 1$. Using the warping function makes full use of the available space. Illustration: Ritschel et al. [2009a].*

functions we can map uniformly distributed x and y (the pixels) to

$$\omega_x(x) = M^{-1}(x) \text{ and } \omega_y(y) = C^{-1}(y|\omega_x(x)).$$

which are then used to define the importance-warped mapping.

$$\Phi(x,y) = \left(\omega_x(x), \omega_y(y), \sqrt{1-\omega_x(x)^2-\omega_y(y)^2}\right).$$

This parametrization of the BRDF slice implicitly includes the cosine term ω_z, i.e. according to $\omega_{i,z} f_r(\mathbf{p}, \omega_o, \omega_i)$. Figure 8.3 illustrates a BRDF-based mapping function $\Phi(x,y)$ and its associated $\Omega(x,y)$. A concrete example for importance-warped micro-buffers with varying glossiness is shown in Figure 8.4.

For special cases, such as the specular Phong component, this mapping can be derived analytically. Since we opt to maintain generality, we restrain from using such specialized solutions and always tabulate the PDF and compute the inverse distributions numerically, which allows us to support any other reflection model. Besides the inverse mapping $\Phi^{-1}(\omega) = (x,y)$ that projects nodes onto micro-buffer pixels, we also compute the forward mapping $\omega = \Phi(x,y)$, which is required for ray casting. Additionally, we need to compute the solid angle $\Omega(x,y)$ that each pixel maps onto in order to be able make a decision whether to accept or refine a node during traversal (cf. Section 8.1.2). For this we rely on the first-order approximation by taking the magnitude of the gradient of $\Phi(x,y)$. As the micro-buffer is now importance-warped, a multiplication with the BRDF or the pixel's solid angle is no longer required, and we can compute convolved indirect illumination summing all pixels in the buffer and weighting them with c^{-1} (see above). Further, we also jitter the local coordinate system slightly at every gather sample to avoid banding artefacts.

Rendering with highly glossy materials is not without problems. The resolution of the warping table is limited to the resolution of the micro-buffer, and consequently the numerically obtained (inverse) projection function might miss important features, i.e. it effectively limits glossiness. But also if the hierarchy has been constructed with an insufficient number of point samples, highly glossy surfaces can make the structure of the hierarchy apparent, because surfels become visible in the reflection. In order to capture fine details, such as edges and texture plausibly in glossy reflections, more point samples are required. Lastly, the likelihood of surfels being projected onto more than only one pixel increases with glossiness and additional ray casting is required.

8.2 Implementation

Micro-rendering has been designed to exploit the parallelism of GPUs. We implemented our method using NVIDIA's CUDA, and thus we will use the respective terminology in this section.

8.2.1 Data Structures

Micro-rendering requires three data structures: the point hierarchy, the micro-buffers, and the post-traversal list, which are all stored in contiguous global GPU memory.

Point hierarchy: After the point hierarchy has been created (Section 8.1.1) it is tightly packed into an array of nodes, where each node consumes 128 bits. We use four half-floats (4×16 bits) to store position and radius. Normal and surface albedo are both quantized to 3×5 bits each and packed into a single 32 bit value. After accounting for additional 8 bits for the cone angle, 24 bits remain unused. We experimented with packing reflected radiance into the unused area but discovered that 8 bits per channel corrupt the results. Thus we allocate an additional radiance buffer reserving 3×16 bits per colour channel.

Micro-buffers: Micro-buffers also reside in global memory. Each micro-buffer entry consumes 32 bit for depth (8 bit) and node index (24 bit). The latter implies that the total number of nodes is limited to 2^{24}, i.e. 2^{23} point samples, which was sufficient for our examples.

Post-traversal lists: Post traversal lists store indices of nodes that project to more than one pixel. For each micro-buffer, we allocate an according traversal list of the same size. Like the micro-buffer, it stores only indices to nodes.

8.2.2 CUDA Implementation

To exploit the high parallelism of GPUs, we launch one CUDA thread per gather location. Each thread then performs four tasks. First, the tabulated BRDF warping is computed (Section 8.1.3). Second, the point hierarchy is traversed and the cut filling the micro-buffer and the post-traversal list computed (Section 8.1.2). The third step processes the post-traversal list. For each pixel in the micro-buffer, a ray is constructed according to the tabulated warping function (Section 8.1.3) and intersected with the oriented disc a node represents, and if necessary, the micro-buffer is updated. In our experiments, ray casting was necessary for only 10% to 30% of the micro-renderings (i.e., the post-traversal list is empty for the other 70% − 90%). Fourth and finally, the indices in the micro-buffer are resolved and the incident radiance is gathered from according nodes in the hierarchy, summed up and handed over to OpenGL for display.

To achieve the highest performance possible, it is crucial that memory accesses are highly coherent on the GPU. This is expected for micro-rendering threads that compute similar hierarchy cuts. We ensured this by ordering the gather locations $\mathbf{p}_c$ according to a 3D Morton-order space-filling curve.

8.2.3 Bilateral Upsampling

The number of gather locations for which micro-rendering is performed can still be overwhelming for interactive rendering. A simple but effective method to reduce workload is computing indirect illumination for a low resolution image first and then upsampling it to the full target resolution. This yields acceptable results because indirect illumination is mostly low-frequency, and artefacts are masked by direct illumination (which is always computed at full resolution). Filtering over strong normal and depth discontinuities must be avoided, as this can lead to perceivable artefacts. Discontinuities are detected during bilateral upsampling [Sloan et al., 2007].

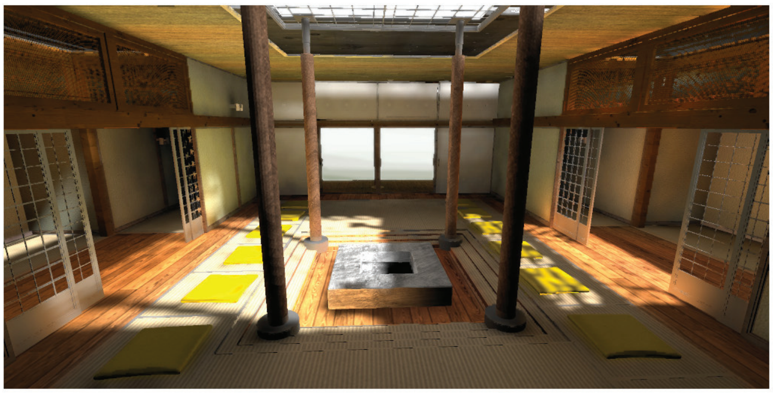


Figure 8.5: *The scene consists of 700k triangles and was converted to 1 million points. Micro-rendering is performed at* 0.74 *fps. Illustration: Ritschel et al. [2009a].*

Nonetheless, for some pixels bilateral upsampling may fail because no viable information is available in the low resolution image. In these cases all interpolation weights are (nearly) zero, and we replace the affected pixels in a second micro-rendering pass, similar to [Dachsbacher and Stamminger, 2005].

Rendering performance was typically 4 to 10 fps for 1/16-th of the target resolution (1/4-th in each dimension) and 0.5 to 1.0 fps for full simulations. Unfortunately, the performance gain remained below our expectations, which can be explained with the high overhead due to additional memory transfers in our CUDA implementation.

8.3 Applications

Micro-rendering is highly flexible and can be used in various ways to produce high-quality interactive renderings and full-resolution renderings on par with offline ray tracing methods [Pharr and Humphreys, 2010]. In this section we outline four applications, one-bounce illumination, multi-bounce diffuse illumination with (instant) radiosity, and final gathering for photon mapping. We use Heckbert's notation [Heckbert, 1990] to assess the variety of light paths that are supported by micro-rendering.

One-bounce indirect illumination: One-bounce indirect illumination can be easily realized by micro-rendering in two steps. First, direct illumination from primary light sources with shadow mapping [Eisemann et al., 2011]

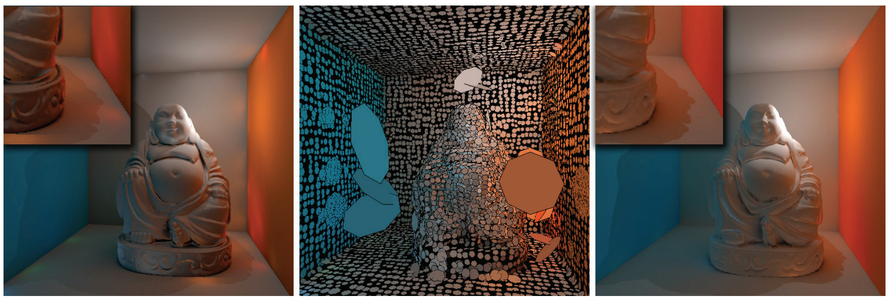


Figure 8.6: *Instead of rendering directly with virtual point lights (left), the point hierarchy is illuminated instead (middle). Rendering with micro-rendering (right) removes many VPL typical artefacts. Performance is* 0.7 *fps at* 512×512 *(full resolution). Illustration: Ritschel et al. [2009a].*

is computed for all nodes in the hierarchy. Second, micro-rendering gathers reflected light from the previously lit nodes and thereby accounts for indirect illumination. Combining indirect illumination with direct illumination (and shadow mapping) makes it possible to capture all $L\{S|D\}^2E$ light paths, as shown in Figure 8.5.

Multiple bounces with instant radiosity: Multiple diffuse bounces can be easily realized using instant radiosity [Keller, 1997]. This technique creates a set of secondary, so called virtual point light sources (cf. Chapters 4.5 and 10) with which the point hierarchy is illuminated using shadow mapping [Ritschel et al., 2008]. This makes rendering of all $LD^*\{S|D\}^2E$ paths possible, and above all, micro-rendering suppresses typical artefacts such as bright splotches and shadow aliasing.

Multiple bounces with radiosity: In the spirit of hierarchical radiosity, micro-rendering can be used for light transport with a Jacobi-iteration [Cohen et al., 1993] scheme. First, point samples are initialized with direct illumination and shadow mapping. Second, final gathering is computed for each point sample. This adds an additional indirect bounce to reflected light which can be repeated until a desired number of indirect bounces is reached or convergence is detected. We have also observed that it is typically sufficient to perform gathering only for nodes on an interior level and then updating the hierarchy using push-pull [Cohen et al., 1993]. Finally we display the radiosity solution performing final gathering at each pixel (see Figure 8.7). Similar to rendering multiple indirect bounces with instant radiosity, this approach also renders all $LD^*\{S|D\}^2E$ light paths.

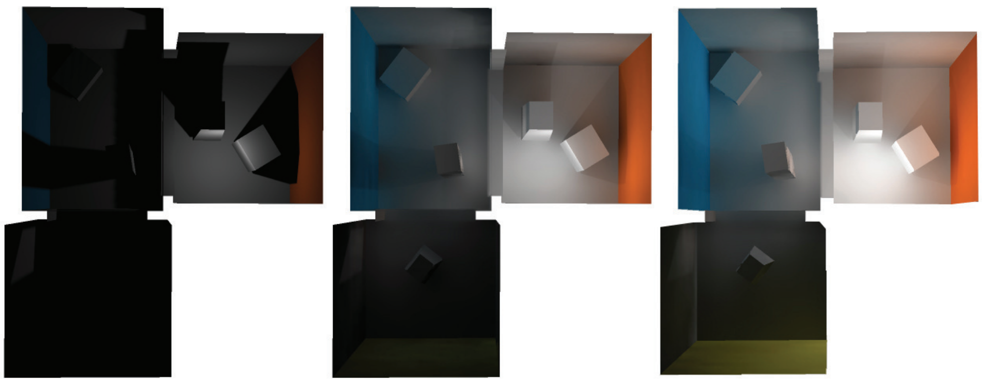


Figure 8.7: *Micro-rendering supports multiple diffuse bounces using a Jacobi-iteration scheme. From left to right: direct illumination, one-bounce indirect illumination, and two-bounce indirect illumination. For multiple bounces gathering is computed for all* 4k *nodes on level 12. Using bilateral upsampling from 1/16-th of the target resolution* (512×512)*, the example runs at* 5 *fps. Illustration: Ritschel et al. [2009a].*

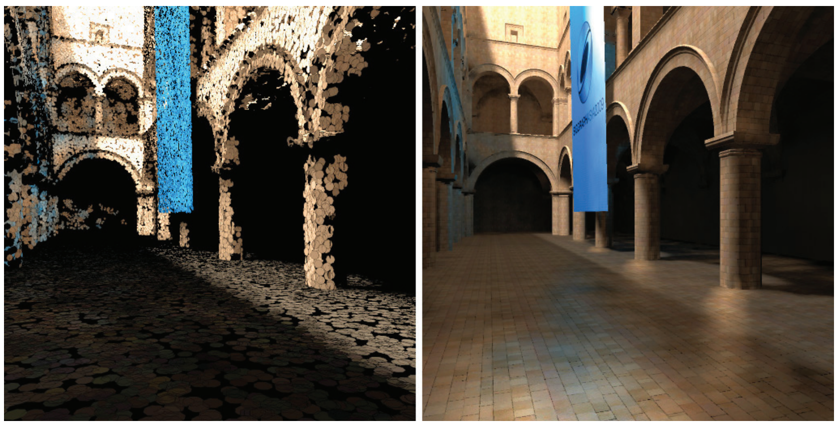


Figure 8.8: *Final gathering for photon mapping: Density estimation at each point sample is precomputed offline. Final gathering is performed at* 2 *fps using bilateral upsampling from* 128^2 *to* 1024^2*. Illustration: Ritschel et al. [2009a].*

Photon mapping: Micro-Rendering enables interactive walkthroughs for photon mapped scenes. However, photon tracing and density estimation for each point sample must be precomputed in an offline step. Radiance values for the interior nodes of the hierarchy are computed using pull steps. Second, final gathering with micro-rendering is used to display the final solution at interactive framerates (Figure 8.8).

8.4 Results

In this section we present our results and findings. All performance measurements were taken on a quad-core 2.4Ghz CPU with an NVIDIA GeForce 280 GTX, for image resolutions of 512^2 pixels, and 24^2 micro-buffer pixels, if not mentioned otherwise. All images show one-bounce indirect illumination. Direct illumination for the image and the point samples is rendered using shadow maps.

Comparision to ISMs and VPL methods: In Figure 8.9 we compare our method to imperfect shadow maps (ISM) [Ritschel et al., 2008]. Using bilateral upsampling, we adjust the number of gather locations to roughly match the speed of the ISM. As can be observed, micro-rendering achieves more distinct indirect shadows. Beyond that, VPL methods can only handle low-glossy scenes efficiently. For highly glossy scenes, millions of VPLs are required [Křivánek et al., 2010].

Bilateral upsampling: The quality of bilateral upsampling heavily depends on the scene's complexity. Higher geometric complexity and glossy materials require more samples. This is illustrated in Figure 8.10, which compares quality for different numbers of gather locations. For diffuse surfaces 1/16 resolution yields visually pleasant results. This is not true for glossy surfaces, where artefacts become apparent resolutions lower than 1/4 are used.

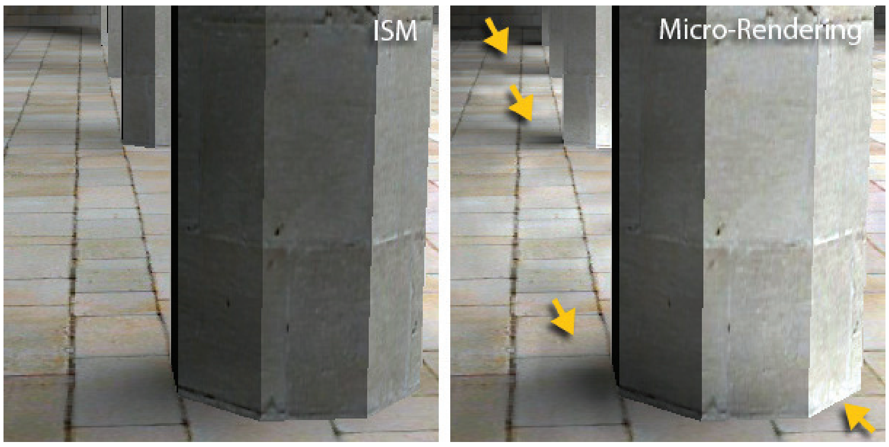


Figure 8.9: *Compared to ISMs [Ritschel et al., 2008], our proposed method achieves higher quality at the same rendering speed (5 fps at* 512×512 *resolution). Illustration: Ritschel et al. [2009a].*

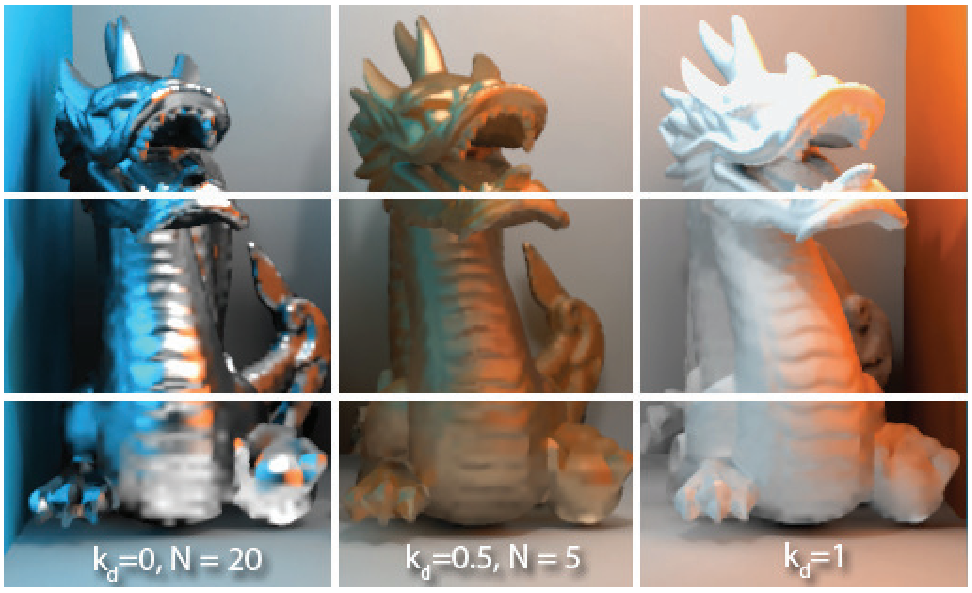


Figure 8.10: *Final gathering at every pixel (0.83 fps, top),* 1/4 *of all pixels (2.2 fps, middle), and* 1/16 *(4.2 fps, bottom). Preview quality can be achieved with a very low number of gather samples for diffuse or low-glossy materials. Illustration: Ritschel et al. [2009a].*

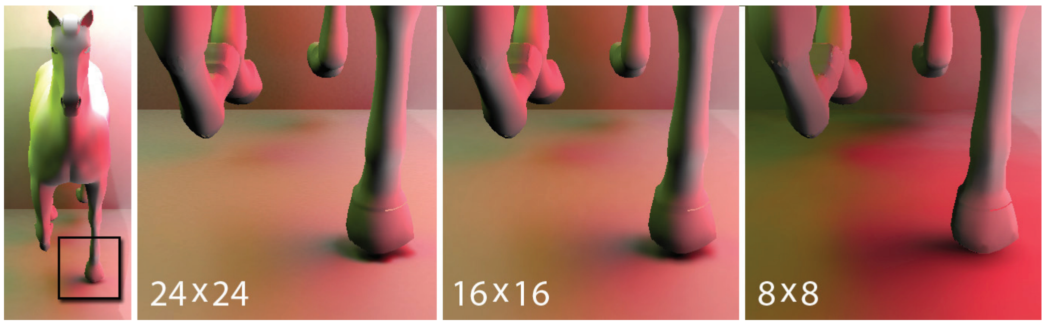


Figure 8.11: *Using small micro-buffers, the rendering performance increases, but quality suffers. We use* 8×8 *(3.2 fps),* 16×16 *(1.5 fps), and* 24×24 *(0.7 fps). Framerates shown for* 256×256 *resolution. Illustration: Ritschel et al. [2009a].*

Micro-buffer resolution: In Figure 8.11, we compare the influence of the size of the micro-buffer on rendering quality. 24×24 pixels (used for all other figures) is the maximum we can fit into local memory on our GPU but even lower resolutions still achieve acceptable results at a higher frame rate.

Performance: The increase of run-time is sub-linear in the number of input points. This is supported by Figure 8.12. The red curve depicts the increase in run-time depending on the complexity for the dynamic horse scene (Figure 8.11) while the blue curve shows the hierarchy update time. Updates are sub-linear as well, but we have experienced that the update time increases significantly at around 2^{20} point samples. We presume hardware limitations cause this peculiarity. The green curve in Figure 8.12 shows that more gather samples can be processes per second when the total number of gather samples increases. This is obviously the case because coherency for cut computations increases as well. We also show that ordering gather locations according to a 3D Morton code increases coherency even further, i.e. more gather locations can be processed than without this sorting. The computation time of our GPU-based final gathering technique is (roughly) split as follows. 1.5% is spent on updating the hierarchy, 2% on building the view-dependent per-pixel mappings $\Phi(x,y)$, 18% on evaluating them, 60% on rasterizing the point hierarchy, 8% on ray casting, and 11% on bilateral upsampling, tone mapping, and direct lighting.

Quality: Figure 8.13 highlights the importance of post-traversal ray casting. Without this step, holes appear in the micro-buffers. This becomes especially noticeable around edges, which are darkened due to missing in-

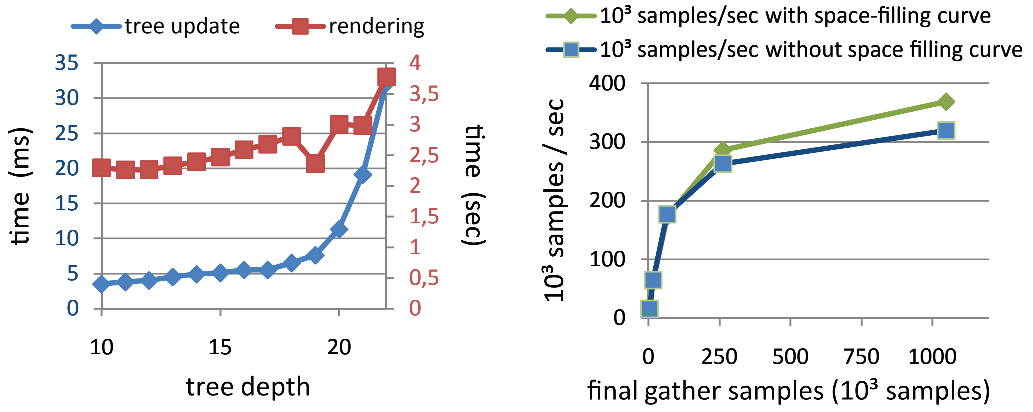


Figure 8.12: ***Left:*** *We show computation time vs. scene complexity, i.e. the number of points (2^N, N: tree depth). The plots indicate sub-linear complexity. The blue curve outlines tree update time.* ***Right:*** *We show the benefit of using a space filling curve for improving memory coherency. More gather samples can be processed with the space-filling curve (green) than without (blue). Illustration: Ritschel et al. [2009a].*

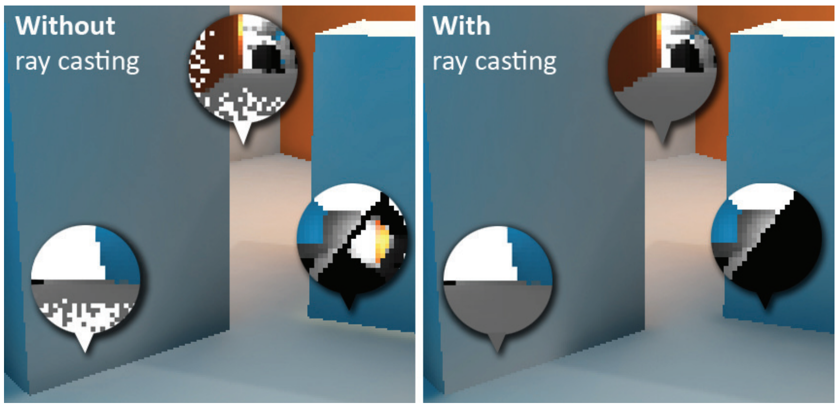


Figure 8.13: *Micro-rendering with (2.5 fps) and without (2.9 fps) the post-traversal ray casting* (256×256)*. Ray casting is an integral step that is needed for high-quality results, especially around edges. Illustration: Ritschel et al. [2009a].*

cident light. For high-quality renderings, this step cannot be omitted.

In Figure 8.14 we compare micro-rendering with and without bilateral upsampling (rendered at 1/16-th of the full resolution) to references produced with the PBRT path tracer [Pharr and Humphreys, 2010]. The differences between the approaches are minor. As expected, they are slightly more perceptible when micro-rendering is used in conjunction with bilateral upsampling. Micro-rendering, however, is interactive, achieving up to 10 fps, whereas the path tracer produces noise-free results within minutes to hours.

8.4.1 Discussion and Limitations

We have shown, that micro-rendering is a fast, robust method, that can handle about 150M gather locations in combination with BRDF importance sampling per second. This is by far not matched by CPU-based ray tracing, which achieves a maximum throughput of about 10M rays per second per core for highly coherent rays [Shevtsov et al., 2007]. Even GPU-based ray tracers do not match our number. It has been reported, that in dynamic scenes about 20M rays can be traced per second, including dynamically updating the acceleration structure [Zhou et al., 2008].

Our method, however, does not come without limitations. Naïvly reducing image resolution and relying on bilateral upsampling afterwards, is not ideal for glossy materials. In this case, more sophisticated approaches, such as by Křivánek et al. [2005], should be used. We also mentioned that we jitter the local coordinate system to avoid banding artefacts, and thus noise becomes visible in our renderings. In our implementation radiance exitant from a node is merely represented by one scalar per channel. This implies that indirect illumination can only be gathered from diffuse surfaces, and more than two specular bounces are not possible. Transparent surfaces and refractions are also not supported.

8.5 Conclusions

We have shown that micro-rendering is a highly scalable GPU efficient method for computing final gathering for high quality global illumination as well as interactive previews, which is magnitudes faster than current CPU implementations. The core of the algorithm is a point-based representation of the scene which allows us to render environment maps (c.f. Chapters 5 and 7) or light probes at each shading point seen from the camera with high speed. Because common directional parametrizations are not suitable for capturing incident light in low-resolution images on glossy surfaces, we additionally developed an importance-warped projection scheme to make this feasible. Although multiple scattering in participating media is not naturally supported, we believe that our method can easily incorporate this functionality, e.g. by holding another set of micro-buffers within the volume along eye rays.

Despite its advantages, our approach is not without limitations. For rendering highly specular surfaces a sufficient number of point samples is required. Beyond that, the regular sampling scheme of bilateral upsampling is not a practical means for improving run-time performance when dealing with highly glossy and specular surfaces. For future work we like to address this issue building on adaptive sub-sampling schemes based on radiance caching.

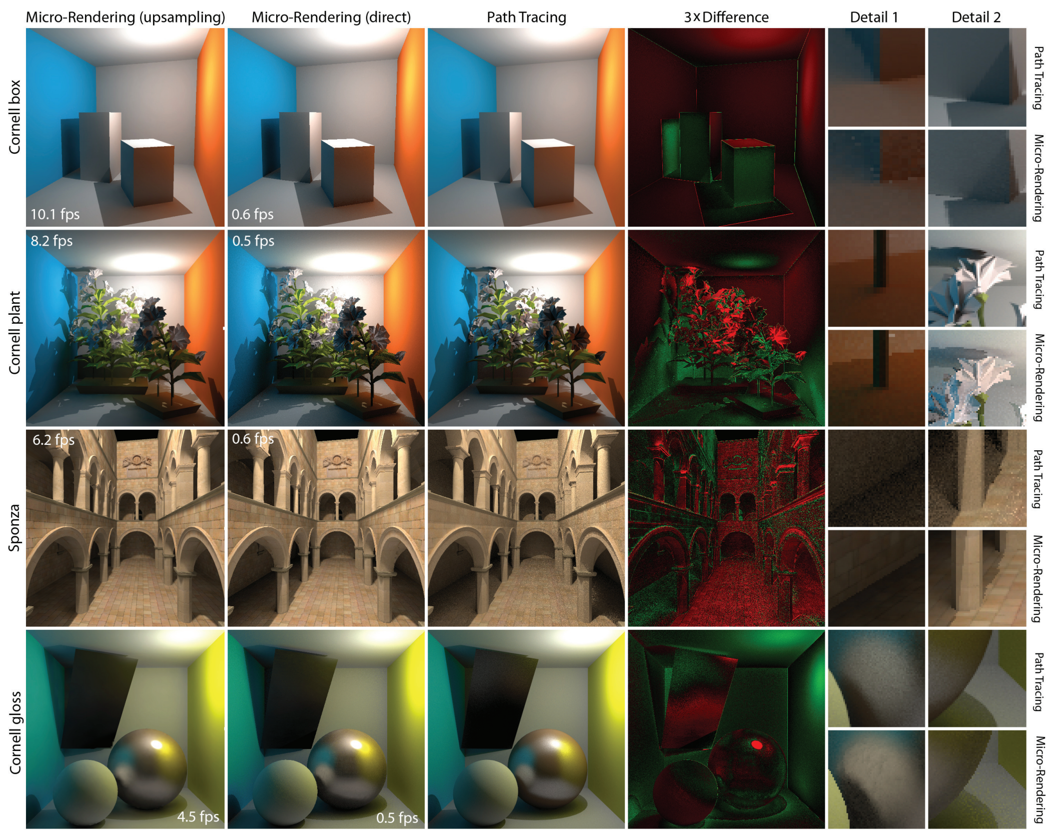


Figure 8.14: *This comparison shows (1) fast preview images rendered at 1/16 of the target resolution (512 × 512) with upsampling, (2) micro-rendering at full resolution and PBRT path tracing. For simple and purely diffuse scenes such as the Cornell box, high quality results can be achieved even with bilateral upsampling. In geometrically complex scenes we can detect slight differences, which we attribute to the discrete micro-buffers. For the Sponza scene, we cannot make out visual differences from the reference rendering. In fact, bilateral upsampling removes noise and produces visually pleasing results. For glossy scenes our approach achieves results close to path tracing. However, bilateral upsampling changes the glossy reflection on the sphere slightly. The error images are scaled by a factor of three. Illustration: Ritschel et al. [2009a].*

CHAPTER 9

Efficient Single Scattering with Epipolar Sampling

SCATTERING in participating media, such as fog or haze, generates volumetric lighting effects known as crepuscular or god rays. This effect greatly enhances the realism in accurately lit virtual scenes (Chapter 8). Their appearance, e.g. light piercing through tree canopies, can be plausibly reproduced assuming single scattering media only. This is because crepuscular rays emerge when in-scattered illumination from primary light sources is modulated using a textured light source or due to volumetric shadowing.

Interactive applications typically rely on ray marching [Perlin and Hoffert, 1989] to accumulate in-scattered light along eye rays, where illumination must be computed at each evaluated sample point. Even for modern GPUs, the computational costs prevents high framerates (≥ 30 fps). A simple way to reduce the cost is to compute single scattering only for a low-resolution image and then up-scaling it to the target resolution [Wyman and Ramsey, 2008]. This blurs the crisp features of god rays which can be explained by the means of epipolar geometry: the image of crepuscular rays can be viewed as epipolar lines which emanate radially from the projection of the light source onto the image plane. Along these lines, the scattering image varies mostly smoothly, whereas strong variations are visible between neighbouring lines (Figure 9.1).

In this chapter we present an algorithm based on epipolar geometry that efficiently renders single scattering effects from point lights in homogeneous media.[1] As such, it is also directly applicable to VPL methods that target multiple scattering in particpating media (Chapter 10).

[1]This chapter is based on our work [Engelhardt and Dachsbacher, 2010] presented at the *Symposium on Interactive 3D Graphics and Games 2010 (I3D'10)*.

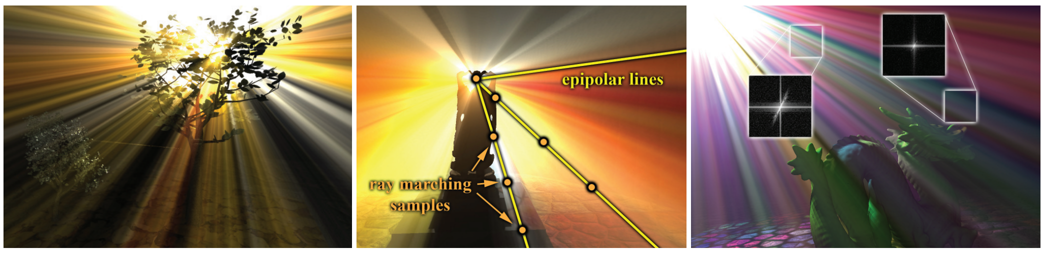


Figure 9.1: *Volumetric shadows and crepuscular rays rendered in real-time with our method. Placing ray-marching samples along epipolar lines (centre) reduces computational costs. This strategy is motivated by the observation that scattering varies mostly smoothly along epipolar lines. This is confirmed by the frequency analysis (right). Illustration: Engelhardt and Dachsbacher [2010].*

9.1 Single Scattering

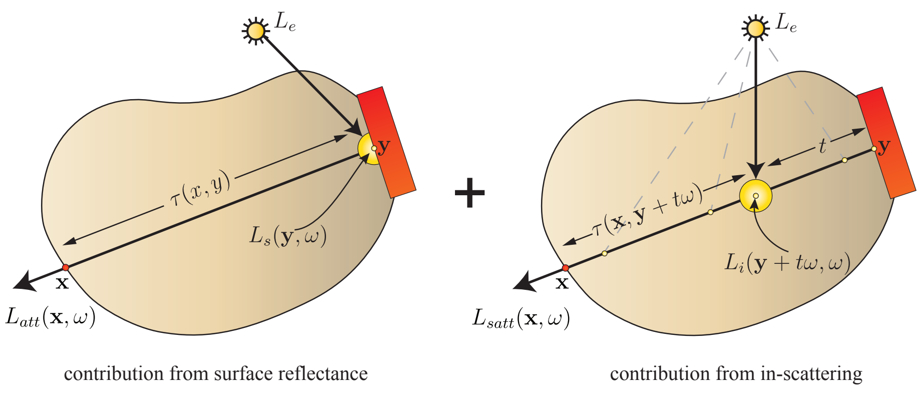
First recall from Chapter 3.3 that radiance arriving at an observer at $\mathbf{x}$ from direction ω in the presence of participating media is defined as the sum of attenuated, reflected surface radiance $L_{\text{att}}(\mathbf{x},\omega)$ and in-scattered radiance $L_{\text{scatt}}(\mathbf{x},\omega)$ (Figure 9.2):

$$L(\mathbf{x},\omega) = \underbrace{L_S(\mathbf{y},\omega)\tau(\mathbf{x},\mathbf{y})}_{=L_{\text{att}}(\mathbf{x},\omega)} + \underbrace{\int_0^s L_i(\mathbf{y}+t\omega,\omega)\tau(\mathbf{x},\mathbf{y}+t\omega)dt}_{=L_{\text{scatt}}(\mathbf{x},\omega)}.$$

Here $s = \|\mathbf{x}-\mathbf{y}\|$ is the distance between the point of in-scattering and the ray origin, and, assuming single scattering,

$$L_i(\mathbf{x},\omega) = \sigma_s(\mathbf{x}) \int_\Omega f_p(\mathbf{x},\omega,\omega_i)\tau(\mathbf{x},\mathbf{z})L_e(\mathbf{z},\omega_i)d\omega_i$$

is light from primary light sources at $\mathbf{z}$ scattered once in the medium at $\mathbf{x}$ into direction ω. The phase function $f_p(\mathbf{x},\omega,\omega_i)$ accounts for the probability of the occurrence of a scattering event. In this chapter we always assume single-scattering participating media with uniform density, i.e. the scattering (σ_s) as well as the absorption (σ_a) coefficients are spatially invariant. Then the transmittance, the fraction of light that passes the medium without being absorbed or scattered, is expressed by the simple analytic formula $\tau(\mathbf{x},\mathbf{y}) = e^{-\sigma_t\|\mathbf{x}-\mathbf{y}\|}$, with $\sigma_t = \sigma_a + \sigma_s$ being the extinction coefficient.



L_e
$\tau(x, y)$
$\mathbf{y}$
$L_s(\mathbf{y}, \omega)$
$\mathbf{x}$
$L_{att}(\mathbf{x}, \omega)$
contribution from surface reflectance
+
L_e
$\tau(\mathbf{x}, \mathbf{y} + t\omega)$
t
$\mathbf{y}$
$L_i(\mathbf{y} + t\omega, \omega)$
$\mathbf{x}$
$L_{satt}(\mathbf{x}, \omega)$
contribution from in-scattering

Figure 9.2: *The total radiance arriving at* $\mathbf{x}$ *is the sum of attenuated surface radiance* L_{att} *(left) and accumulated in-scattered radiance* L_{scatt} *(right). Illustration: Engelhardt and Dachsbacher [2010].*

9.2 Epipolar Sampling

In this section we detail how our sampling strategy is derived from the concepts of epipolar geometry. This in particular includes how ray-marching samples are determined to compute and interpolate in-scattered light L_{scatt} for every pixel. Epipolar sampling is inspired by the appearance of crepuscular rays that emanate radially from a (textured) light source. The reason for this can be easily explained with epipolar geometry (see e.g. [Hartley and Zisserman, 2004]).

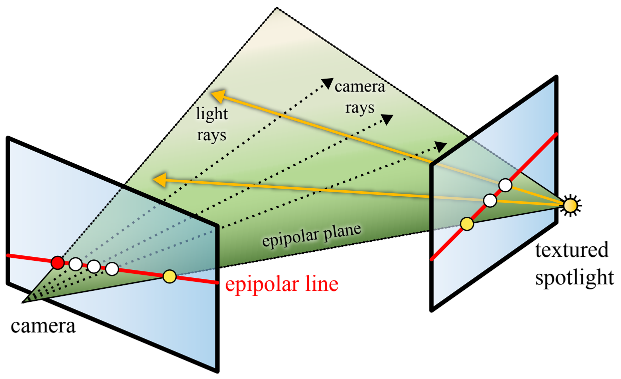


Figure 9.3: *We render crepuscular rays from textured spot lights. Epipolar geometry explains why we can sample sparsely along epipolar lines (when detecting discontinuities properly). Illustration: Engelhardt and Dachsbacher [2010].*

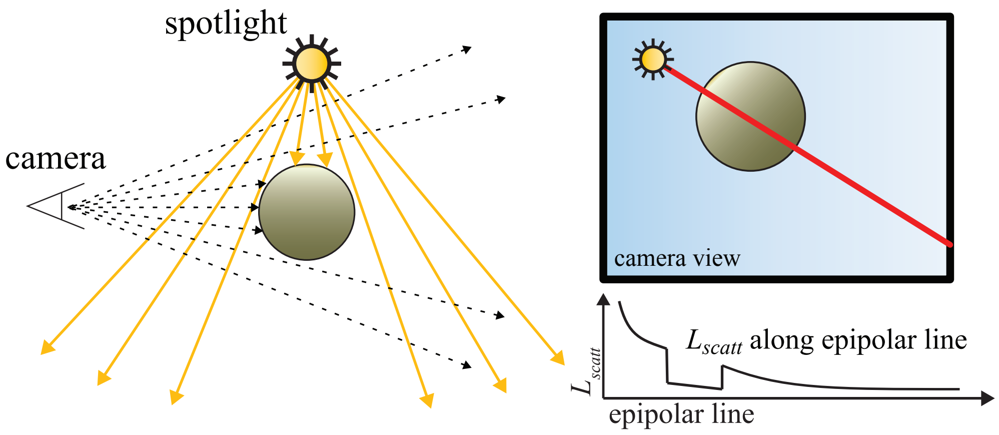


Figure 9.4: *Left: shadowed regions do not contribute to in-scattering. Right: Consequently, the in-scattered light L_{scatt} reaching the camera exhibits abrupt changes (plot). Illustration: Engelhardt and Dachsbacher [2010].*

Looking at a set of rays from the camera going through pixels that lie on one of these crepuscular rays, which are epipolar lines on the screen (Figure 9.3, left), then we observe that these rays map to another epipolar line on the "image plane" of the spot light (Figure 9.3, right). That is, when computing the in-scattered light, these camera rays all intersect the same light rays. All these intersections form an epipolar plane. The only difference is how far these intersections are away from the light source, and thus how much light was attenuated on its way to the intersection. Attenuation varies smoothly for camera rays along an epipolar line while camera rays on another epipolar line intersect different light rays and thus potentially produce different in-scattering. In the presence of occluders, however, the in-scattered light along an epipolar line can also exhibit discontinuities. This is because of two reasons (see Figure 9.4): the distance to the first visible surface along a camera ray changes abruptly from one ray to another, and shadowed regions in space do not contribute to the ray integral. Taking advantage of these observations our method for placing the ray-marching samples requires three steps:

1. Selection of a user defined number of epipolar lines, and refinement of an initial equidistant sampling along these lines to capture depth discontinuities (Section 9.2.1).

2. Ray marching all samples (in parallel on the GPU; Section 9.2.2)

3. Interpolation along and between the epipolar line segments (Section 9.2.2).

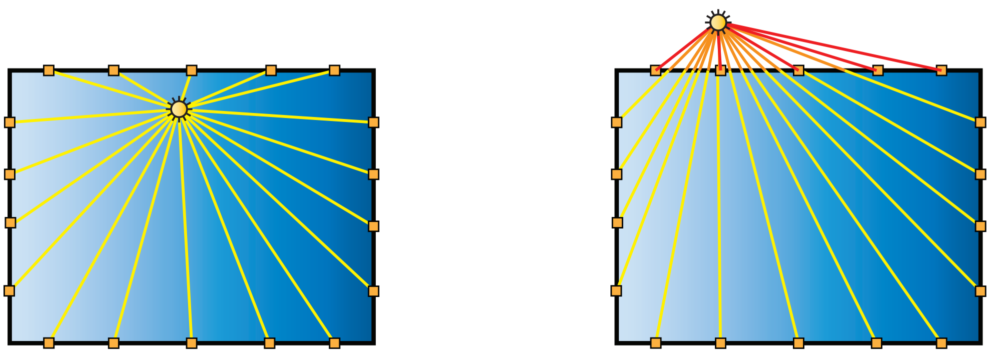
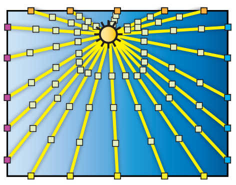


Figure 9.5: ***Left:*** *The light source projects onto a point in the image and epipolar lines are equidistantly placed.* ***Right:*** *The light projects to a point outside the image region. Invisible lines (red) are culled, all visible lines (yellow) are clipped against the screen. Illustration: Engelhardt and Dachsbacher [2010].*

9.2.1 Generating Epipolar Lines

The integral step of our method is to place samples along epipolar lines, and thus we have to determine these first. The construction of epipolar lines depends on the projection of the light source onto the image plane, for which we have to consider the following cases.

Light source in front of the camera: In the simplest of all cases, the light source is in front of the camera, and we project the light source's position onto the image plane. Then we create a set of epipolar lines, connecting equidistantly sampled positions on the border of the screen with the projected position of the light source (Figure 9.5, left). If the projection of the light source is outside the screen borders, some epipolar lines are invisible and are culled prior to the subsequent steps. Epipolar lines that map to outside of the light frustum are omitted from further computation as well. For the remaining lines, we compute the location where they enter and exit the visible part of the image plane (Figure 9.5, right).



For each epipolar line, we create a low number of initial samples along the epipolar lines, typically 8 to 32, which is enough the capture the variation of the in-scattered light in absence of occluders. This divides the rays into epipolar line segments (right).

Light source behind the camera: If the light source is behind the camera, its projected position is the vanishing point where the (anti-)crepuscular rays converge at infinity. As our sampling does not distinguish whether rays emanate from, or converge to a point, we can proceed exactly as in the aforementioned case.

Light source and camera coincide: If the camera and the light source coincide then the crepuscular rays and the camera rays are parallel. In this barely happening case the epipolar lines degenerate to points, and our sampling scheme reduces to a naïve screen space sub-sampling [Wyman and Ramsey, 2008].

9.2.2 Sampling Refinement

After the generation of the epipolar lines and the initial samples, we need to detect depth discontinuities along the line segments. The in-scattered radiance might change abruptly at discontinuities (Figure 9.4), and thus we need to place an additional sample directly before and after these locations, otherwise distracting interpolation artefacts appear. For this we use the depth buffer of the scene and linearly search along each epipolar line segment for adjacent locations that exhibit a large threshold. Of course we have to generate the depth buffer first, but we have to render the scene anyway in order to account for attenuated surface radiance. Instead of a linear search, we also experimented with 1-dimensional min-max depth mip maps along epipolar lines to detect discontinuities (similar to [Nichols and Wyman, 2009], but in 1D). Obviously, this produces the same results, but as we need to detect each discontinuity only once, the construction of the mip map did not amortize, presumably as it requires additional render passes. Beyond that, we have also experimented with a different refinement criterion, taking the difference of in-scattered light at the end points of an epipolar line segment in order to detect discontinuities in the in-scattering term. However, this has two inherent problems: first and foremost, this criterion might miss volumetric shadows if the line segment spans a shadowed region (for example in situations like the one depicted in Figure 9.4). Second, this criterion requires the computation of the in-scattered light to be interleaved with the sample refinement. Our experiments showed that this is hard to parallelize efficiently on a GPU.

The resulting sampling then matches the observations that we made from the appearance of the crepuscular rays: we have more samples close to the light source where the angular variation of the in-scattered radiance

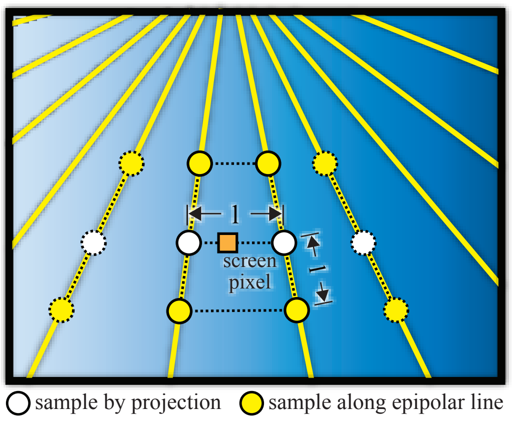


Figure 9.6: *We interpolate the in-scattered radiance for a pixel on the screen using a bilateral filter sampling the closest two epipolar lines. If these taps are invalid, we move to the next outer epipolar lines (sample taps shown with dashed lines). Illustration: Engelhardt and Dachsbacher [2010].*

is stronger, a predefined minimum sampling to capture the attenuation of the in-scattered light, and additional samples to account for discontinuities.

9.2.3 Ray Marching and Upsampling

After the sample placement, we compute the in-scattered radiance for all samples on the epipolar lines. From these samples we obtain a piecewise linear approximation of accumulated L_{scatt} along each epipolar line. From this sparse sampling of the image plane, we need to reconstruct the in-scattering for each pixel of the final image by interpolation. We found that a bilaterial filter, drawing the filter taps directly from the epipolar lines, is well-suited to interpolate the ray-marching samples while preserving edges at the same time.

In order to compute the in-scattered radiance for a pixel on the screen we first determine the two closest epipolar lines. We then interpolate bilaterally along, followed by another bilateral interpolation between the epipolar lines. First, we project the pixel's location onto the two closest lines (white dots in Figure 9.6) and denote the distance between the two projected points as l. Along each of these two epipolar lines we use a 3 to 5 tap 1D-bilateral filter. The filter taps are taken at distances with a multiple of l along the epipolar lines towards and away from the light source, where L_{scatt} is interpolated linearly from the ray-marching samples (that have been placed along the epipolar lines). The filter weight for the i-th tap is determined by its distance (l_i, a multiple of d) and depth difference d_i to the pixel. We compute the weight by a multiplication of two radial basis functions taking the l_i and d_i as input, respectively. We tried weighting similar to Shepard's scattered data interpolation as $w_S(x) = ax^{-2}$, using a Gaussian $w_G(x) = e^{-ax^2}$ (a is a user-defined, scene-dependent constant), and simple thresholding (for d_i only). We found that all three yield comparable results once the appropriate parameter is adjusted.

The interpolation along each epipolar line yields an interpolated colour and we also keep track of the sum of weights, denoted as w_{left} and w_{right}. Next we interpolate bilaterally between the two colours, according to the weights. However, it may occur that no suitable sample taps are found along an epipolar line (i.e. all taps have weights close to 0), in particular in situations where object discontinuities are aligned with the epipolar lines. In these cases w_{left} or w_{right} are obviously (close to) zero as well, and we take the respective next outer epipolar line to obtain valid taps for the screen pixel.

This rather simple bilateral filter yields visually pleasing results in most cases. Of course, any such interpolation can cause artefacts due to missing information in some geometric configurations. We also experimented with a bilateral filter that bases its decision on a distinction of taps whose depth is close to the pixel's depth, significantly larger or smaller (similar to the implementation of Wyman and Ramsey [2008]). We found that this yields comparable results, however at increased computation cost. Note that this approach might be more suitable to favour one sort of artefacts over others, e.g. by preferring distant taps to close ones and thus preventing dark pixels.

9.3 Implementation

In this section we detail the individual steps of the implementation of epipolar sampling. Our method has been designed with modern graphics hardware in mind and implemented using Direct3D 10. The rendering of the final image comprises 3 steps: (1) rendering the scene with direct lighting into an off-screen buffer, (2) determining the in-scattered light for the entire image using epipolar sampling and interpolation, (3) and additively combining both contributions while accounting for attenuation.

The epipolar sampling and the computation of the in-scattered light consist of the following steps:

- Determine the epipolar lines and place the initial samples.
- Detect all discontinuities along the epipolar lines and determine for

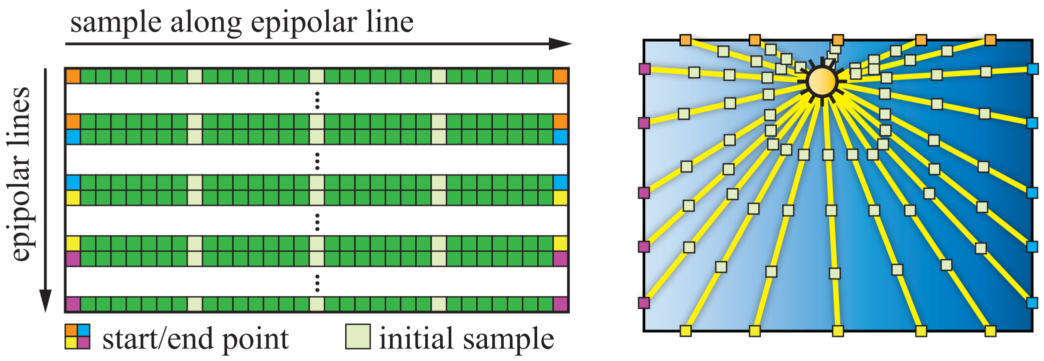


Figure 9.7: *Each epipolar line is mapped to one row in a 2D texture. Each pixel represents one sample along one of these lines. Illustration: Engelhardt and Dachsbacher [2010].*

which samples in image space the in-scattered light has to be computed using ray marching and where an interpolation is sufficient.

- Compute the in-scattering with ray marching and interpolate along the epipolar lines. Finally interpolate between epipolar lines to obtain L_{scatt} for the entire image.

9.3.1 Epipolar Line and Initial Sample Generation

For efficient computation and interpolation, we store the in-scattered light along the epipolar lines in a 2D texture: each texture row corresponds to one epipolar line, and each column corresponds to one location along that epipolar line (Figure 9.7). The number of columns in this texture determines the maximum number of ray-marching samples along that line. However, we do not necessarily ray march for every sample as the in-scattered radiance might also be interpolated for a texel (typically it is interpolated for most of them).

The first step is to relate the texels in this 2D layout to coordinates on the viewport and store this information in the *coordinate texture*. For this we setup the epipolar lines in a pixel shader as described in Sect. 9.2.1: each texture row corresponds to the epipolar line from the projected light source position to one location on the screen's border (Figure 9.7). Next, these lines are clipped against the light frustum and the viewport to obtain the contributing epipolar lines and the respective visible segments thereof. After clipping, we know the entry and exit locations of each line and compute the location of every texel by interpolation between the segment's endpoints. Recall from Sect. 9.2.1 that line segments that do not overlap the viewport are excluded from subsequent computation. We use the stencil buffer to mask these epipolar lines and rely on the GPU to automatically omit computations in later render passes.

9.3.2 Discontinuity Detection and Sampling Refinement

After the construction of the epipolar lines, we determine for which samples the in-scattered light, L_{scatt}, has to be computed using ray marching (*ray-marching samples*) and for which interpolation is sufficient (*interpolation samples*). For interpolation samples, we also need to determine from which other two samples the in-scattered light is interpolated. This information is stored in another 2D texture (*interpolation texture*) with the same layout as the one described above.

Our implementation carries out the classification of texels and the determination of interpolation sources in a single pixel shader. For each texel in the interpolation texture we regard the entire line segment to which it belongs (line segments are determined by the initial sample placement). For every sample in the segment we copy the depth values from the scene's depth buffer at the corresponding locations (obtained from the coordinate texture) into a local array.

Next we determine from which samples in that segment we interpolate the in-scattered light for the currently considered texel. The case that this texel becomes a ray-marching sample is detected in the same procedure (see below). The following pseudo-code illustrates the search for the interpolation samples (indexed by `left` and `right`) in the local depth array `depth1d` of size `N`, for the current texel at location `x` (Figure. 9.8):

```
left = right = x;
while ( left > 0 ) {
  if(abs( depth1d[left-1], depth1d[left] ) > threshold)
    break;
  left --;
}
while ( right < N-1 ) {
  if(abs( depth1d[right], depth1d[right+1] ) > threshold)
    break;
  right ++;
}
```

Note that if no discontinuities are found then the detected interpolation samples are the endpoints of the initial line segment. If the epipolar line crosses a discontinuity at `x`, then the corresponding sample will detect itself as interpolation sample. Samples that interpolate from themselves ob-

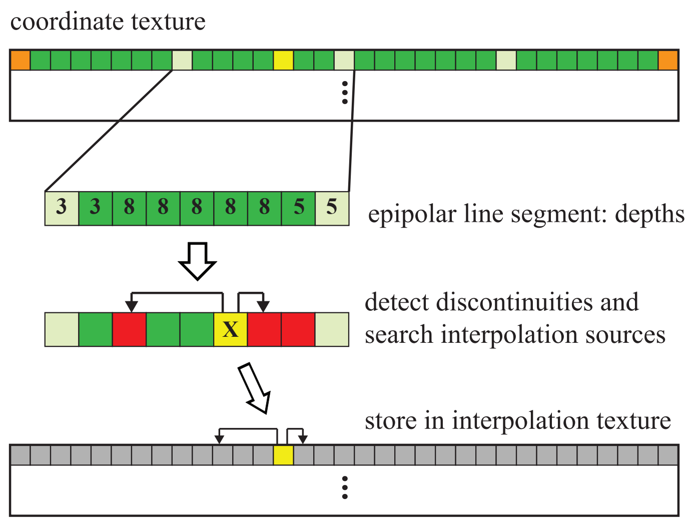

coordinate texture
3
3
8
8
8
8
8
5
5
epipolar line segment: depths
X
detect discontinuities and
search interpolation sources
store in interpolation texture

Figure 9.8: *For each pixel in the interpolation texture we search depth discontinuities along the epipolar line segment to which it belongs. Samples that require ray marching are marked, and for all other samples we store from which other samples we interpolate the in-scattered radiance into the interpolation texture.Illustration: Engelhardt and Dachsbacher [2010].*

viously are the ray-marching samples. The number of texture accesses required for the discontinuity search for every epipolar line is: the maximum number of samples per epipolar line times the samples per initial line segment. Note that for typical settings this has negligible cost on contemporary GPUs.

9.3.3 Ray Marching, Interpolation, and Upsampling

The next step is to determine the in-scattered radiance for every sample along the epipolar lines. These values will be stored in a third texture (again the same extent as the aforementioned textures) that we name *radiance texture*. After the discontinuity detection some texels have been classified for interpolation while others are marked for ray marching. For the latter we need to execute the costly ray marching operations, but we do not want the GPU to spend time on the other samples. We experimented with rendering point primitives to trigger ray marching operations for individual samples, but this approach showed bad performance. Again the stencil buffer proved to be the best choice, and we mark the texels corresponding to ray-marching samples first. Following that we render a full-screen quad over the entire radiance texture (which is bound as render target) using a pixel shader that performs the ray marching. The early-z optimization of GPUs takes care that this shader is only executed where required. Next, a simple shader performs the interpolation: for every texel it looks up both interpolation sources and interpolates according to its relative position. Note that we can use the (already existent) stencil buffer to distinguish between the sample types, but in the discontinuity detection all ray-marching samples detected themselves as interpolation source anyway. We always ray-march at a fixed step size restricted to the light frustum (i.e. we do not ray march in definitely unlit parts of the scene).

At this point we generated all information that is necessary to determine the in-scattered radiance for all pixels in the final image. This procedure is illustrated in Figure 9.9. First we compute the ray from the projected light source position to the pixel and intersect this ray with the border of the screen. Next we determine the epipolar line that is closest to this intersection. To facilitate this computation, we use a small 1D texture that stores the closest epipolar line for every pixel on the screen border. Note that this texture depends on the number of epipolar lines only and is precomputed and stored once.

Finally, we project the pixel's location onto the two closest epipolar lines to obtain the taps for the bilateral filter. The filtering needed to obtain the interpolated in-scattered radiance is implemented exactly as described in Sect. 9.2.3: after interpolating bilaterally along each of those epipolar lines, we interpolate in-between. If no suitable taps, i.e. no taps with large weights, are found on one of the two closest epipolar lines, we move to the next outer line. In our experiments, this procedure was sufficient to find an appropriate radiance sample except for very rare cases.

9.4 Results

We implemented our method in Direct3D 10 and measured all results using an ATI Radeon HD4850 on a 2.4GHz Core2Quad processor; all timings are measured for rendering at 1920×1080 resolution using the scattering model by Hoffman and Preetham [2003]. The major cost in the rendering is ray marching which indicates that the epipolar sampling (including line generation, depth discontinuity search, and interpolation) introduces a small overhead only. The detailed timings for the left image in Figure 9.1 are: epipolar line generation 1.2ms, discontinuity search 6.0ms, ray marching 16.7ms (500 steps per ray), interpolation 4.4ms, for 1024 epipolar lines and 32 initial ray-marching samples (recall that the initial segment size influences the cost for searching discontinuities); less then 120.000 ray-

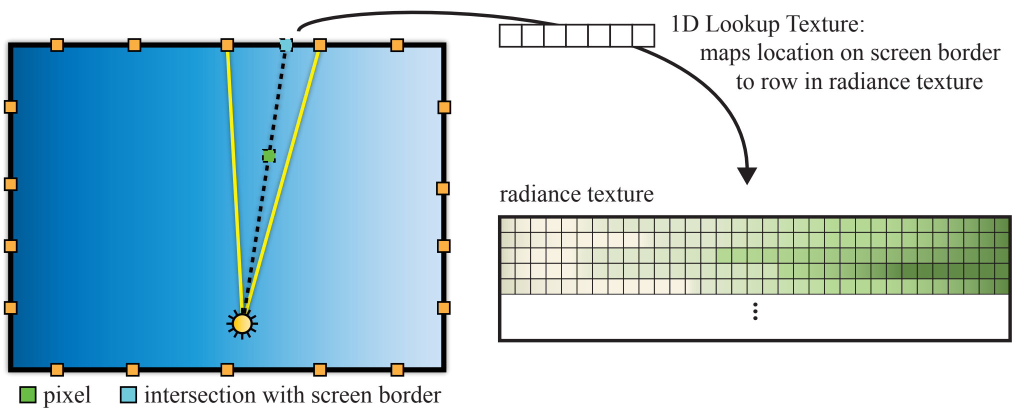

1D Lookup Texture:
maps location on screen border
to row in radiance texture
radiance texture
🟩 pixel
🟦 intersection with screen border

Figure 9.9: *For interpolation of the in-scattered radiance for a pixel we determine the closest epipolar line using a 1D lookup texture.Illustration: Engelhardt and Dachsbacher [2010].*

marching samples have been generated, i.e. ray marching was performed for only 5.8% of the pixels in the image. To our knowledge there is no other method reducing the ray-marching samples in a non-naïve manner. To this end, we measure the performance gain over brute-force rendering (ray marching for every pixel) and naïve sub-sampling, and it naturally increases with the number of steps for the ray marching, as we significantly reduce the number of ray-marching samples itself.

Figure 9.10 shows renderings of the "yeah right" model comparing epipolar sampling to naïve sub-sampling. With the settings used for the top-right image in Figure 9.10, our method yields results indistinguishable from per-pixel ray marching (65k samples), while our method ray marches only about 17k rays. Figure 9.11 demonstrates the influence of the number of epipolar lines; as expected, fewer lines yield blurrier images.

Note that the images rendered with epipolar sampling show a crisper appearance which is due to the sample placement that matches the frequency content of the image (see Figure 9.1, right). However, our method tends to oversample along crepuscular rays when the light source is close to the screen border and some epipolar lines are short (noticeable in Figure 9.7, right). This is because we place a fixed number of initial ray-marching samples along each line. Although this can be easily adjusted, we opted for an as-simple-as-possible implementation. It would also be possible to further optimize the discontinuity search by using larger depth thresholds for distant parts of the scene (similar optimizations have been proposed by Wyman and Ramsey [2008]). When using our method in dynamic scenes, e.g. under light and camera movement, we observed stable and flicker-free rendering.

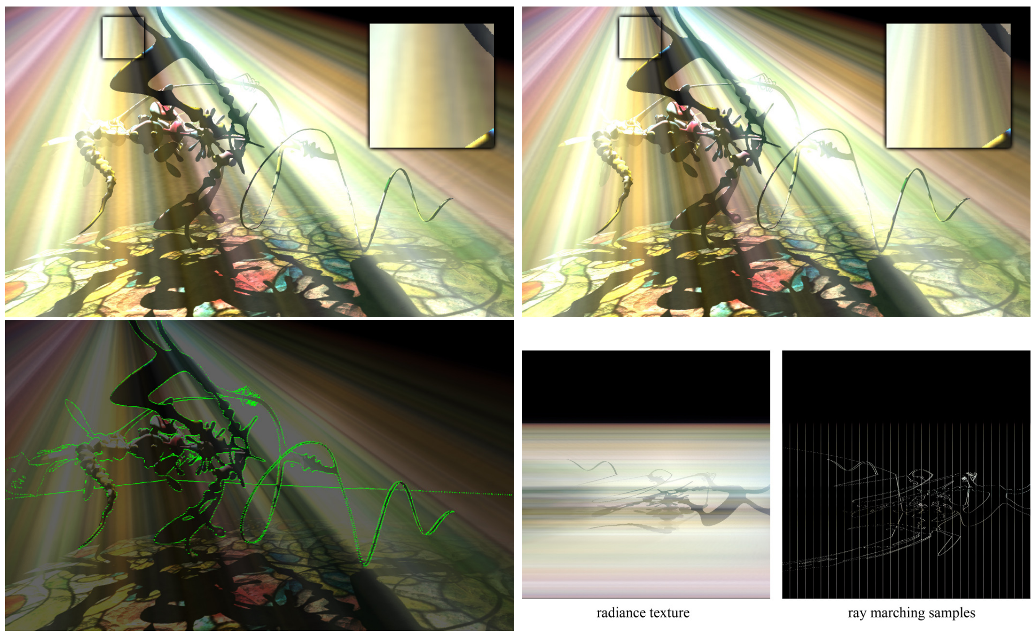


Figure 9.10: *Top left: naïve sub-sampling and ray marching (150 steps) with* 256×256 *samples at 40 fps. Top right: epipolar sampling with 1024 lines and 8 initial samples per line yields much better and crisper results at comparable speed (46 fps) using only about 17.000 ray-marching samples. Bottom: the locations where discontinuities are detected (green), and the radiance texture. Illustration: Engelhardt and Dachsbacher [2010].*

When using a lower number of epipolar lines, we have to pre-filter the light texture accordingly.

9.5 Conclusions

Epipolar sampling naturally places ray-marching samples where they are required and significantly reduces the cost for rendering single-scattering participating media effects with (textured) point-light sources in real-time. The implementation is simple and can be easily integrated into existing renderers and combined with other per-ray optimizations. So far, our implementation is limited to a single light source. An extension to support multiple light sources would be to place ray-marching samples preferably at the intersection of the epipolar lines of both light sources. Many-light techniques for rendering participating media also benefit from our approach.

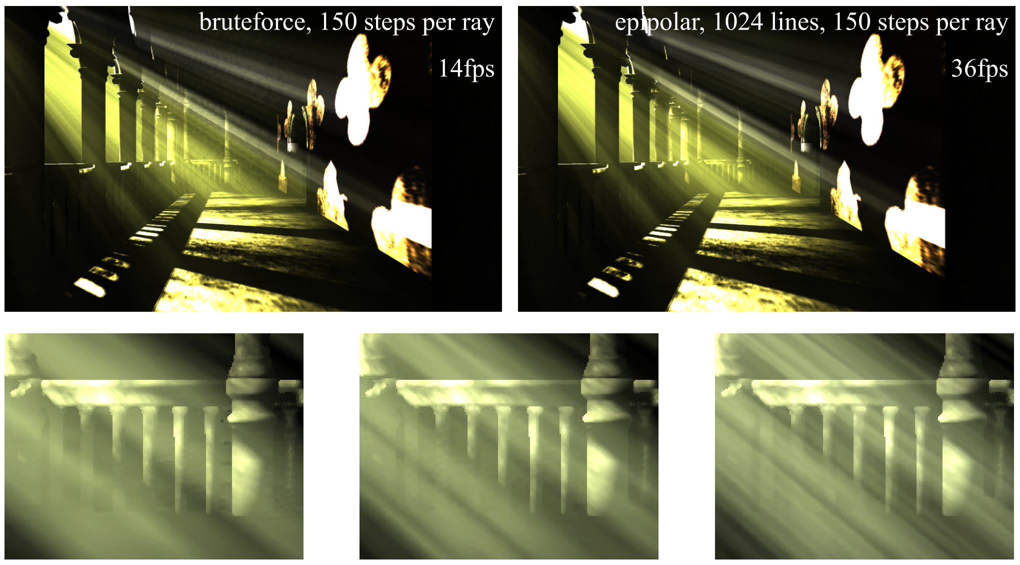

bruteforce, 150 steps per ray
14fps
epipolar, 1024 lines, 150 steps per ray
36fps
256 lines, 200k samples, 54fps
512 lines, 273k samples, 48fps
1024 lines, 443k samples, 36fps

Figure 9.11: *Top left: brute force rendering with 150 ray marching steps per pixel. 150 steps are still too few in this case (please zoom in to see the artefacts). Top right: our method yields smoother results with the same number of steps, due to the interpolation along the epipolar lines. Bottom line: close ups with increased contrast of renderings with 256, 512, and 1024 epipolar lines, and the respective number of ray marching events.Illustration: Engelhardt and Dachsbacher [2010].*

These methods approximate multiple scattering successively accumulating the single scattering contribution from virtual point light sources (cf. Chapter 10). Therefore our algorithm can be directly applied in a successive manner. Heterogeneous participating media pose a challenge as an extension is difficult. This is because distributing ray-marching samples sparsely in image space easily misses important features of the varying transmittance.

CHAPTER 10

Approximate Bias Compensation

In the previous chapter, we have shown that scattering effects contribute immensely to the realism of a scene, but single scattering is not sufficient for most scenes. Instant radiosity [Keller, 1997] (IR) approximates indirect illumination and multiple scattering [Raab et al., 2008] using a distribution of virtual point light sources (VPLs). The combined direct illumination and single scattering from primary and secondary (VPL) light sources yields full global illumination with multiple scattering. This makes the approach ideal for GPU acceleration because (1) preprocessing costs are minimal, (2) volumetric shadow mapping [Salvi et al.] can be used for visibility, (3) it is highly scalable, and (4) does not require complicated data structures.

Despite these striking advantages, rendering with VPLs suffers from luminous splotches in the image because VPLs have a high contribution to nearby shading points (Figure 10.1b). To avoid these artefacts, the contribution of a VPL is artificially bounded, which also eliminates short distance light transport. Bias compensation counters this effect and recovers the missing energy (Figure 10.1c) but requires costly ray tracing and thereby ruins the GPU-friendliness of IR.

In this chapter[1] we develop a GPU-friendly *approximate bias compensation* algorithm for efficient rendering of multiple scattering in scenes with heterogeneous media, and in the next chapter we extent our discussion to surface bias compensation. To derive our algorithm, we investigate different sampling strategies for the compensation energy to reduce noise in the final image and analyse various simplifications that greatly speed up the algorithm and make GPU implementations feasible. Our results show that the proposed simplifications yield results visually indistinguishable from ground truth.

[1]This chapter is based on our work [Engelhardt et al., 2012] presented at *Pacific Graphics 2012*

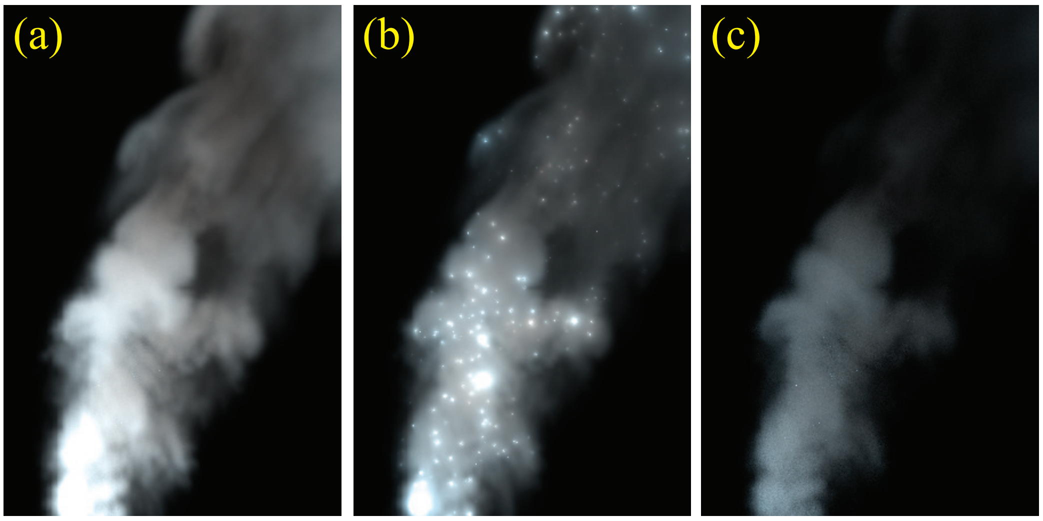


Figure 10.1: *(a) Unbiased rendering of heterogeneous smoke. (b) Rendering multiple scattering with VPLs causes characteristic splotches. These can be avoided by clamping, however, this removes a significant amount of energy (c). Illustration: Engelhardt et al. [2012].*

10.1 Instant Radiosity with Participating Media

The volumetric measurement equation was introduced in Chapter 4 as an integration over light carrying paths, concisely describing light transport in the presence of participating media. This equation computes the response

$$I_j^* = \sum_{k=1}^{\infty} \int_{\mathscr{P}_k} W_e^j(\mathbf{x}_0 \leftarrow \mathbf{x}_1) G^*(\mathbf{x}_0 \leftrightarrow \mathbf{x}_1) V^*(\mathbf{x}_0 \leftrightarrow \mathbf{x}_1) \mathbf{T}^*(\overline{\mathbf{x}}_k) L_e^*(\mathbf{x}_k \rightarrow \mathbf{x}_{k-1}) d\mu(\overline{\mathbf{x}}_k). \tag{10.1}$$

of a virtual sensor to incident light, and serves as theoretical foundation for many light transport algorithms, such as instant radiosity [Keller, 1997] among others. Instant radiosity (IR) approximates indirect illumination by a collection of virtual point lights (VPLs), and accumulating the illumination from primary light sources and VPLs yields full global illumination [Keller, 1997]. This method can easily be extended to participating media [Raab et al., 2008], where the combined single scattering from primary light sources and VPLs yields full volumetric illumination.

This can be easily derived from the measurement equation for *indirect illumination* and *multiple scattering* (for path length $k \geq 3$):

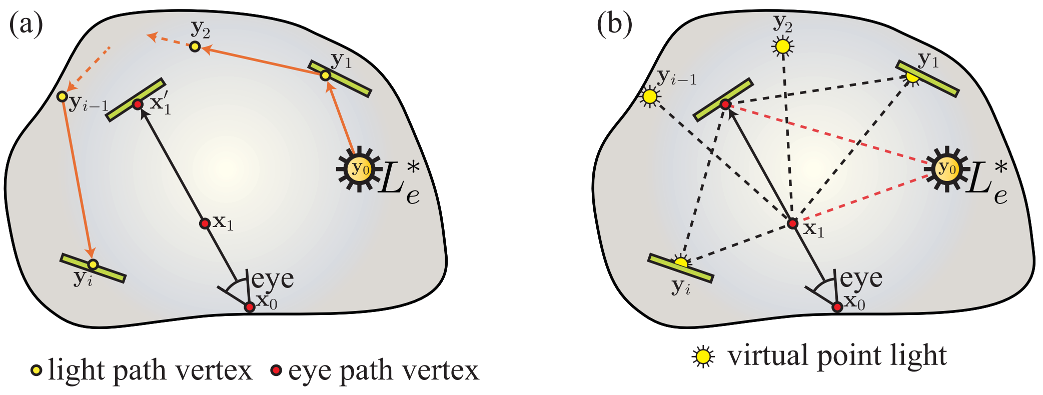


Figure 10.2: *(a) VPL-methods split light paths into light sub-paths and eye sub-paths. Path vertices either lie on a surface or in the volume. (b) Light path vertices are replaced by VPLs. Direct illumination and single scattering from primary light sources and VPLs (connection to eye path vertices) yields full global illumination with multiple scattering.*

$$\begin{aligned}\hat{I}^*_j &= \sum_{k=3}^{\infty}\int_{\mathscr{P}_k} W_e(\mathbf{x}_0\leftarrow\mathbf{x}_1)G^*_v(\mathbf{x}_0\leftrightarrow\mathbf{x}_1)\mathbf{T}^*(\bar{\mathbf{x}}_i)L^*_e(\mathbf{x}_k\rightarrow\mathbf{x}_{k-1})d\mu(\bar{\mathbf{x}})\\ &= \sum_{i=1}^{\infty}\int_{\mathscr{P}_1}\int_{\mathscr{P}_i} \underbrace{W_e(\mathbf{x}_0\leftarrow\mathbf{x}_1)G^*_v(\mathbf{x}_0\leftrightarrow\mathbf{x}_1)f^*_r(\mathbf{x}_0\rightarrow\mathbf{x}_1\rightarrow\mathbf{y}_0)}_{\text{eye subpaths}}\underbrace{G^*_v(\mathbf{x}_1\leftrightarrow\mathbf{y}_0)}_{\text{connection}}\\ &\cdot\underbrace{f^*(\mathbf{x}_1\leftarrow\mathbf{y}_0\leftarrow\mathbf{y}_1)\mathbf{T}(\bar{\mathbf{y}}_i)L^*_e(\mathbf{y}_i\rightarrow\mathbf{y}_{i-1})}_{\text{light subpaths}}d\mu(\bar{\mathbf{y}})d\mu(\bar{\mathbf{x}}_1).\end{aligned} \qquad (10.2)$$

The integral is split into eye paths and light paths (Figure 10.2a). Factoring out the light paths, we obtain the function

$$\hat{L}^*(\mathbf{x}_1\leftarrow\mathbf{y}_0) = \sum_{k=1}^{\infty}\underbrace{\int_P\cdots\int_P}_{\text{k-times}} f^*(\mathbf{x}_1\leftarrow\mathbf{y}_0\leftarrow\mathbf{y}_1)\mathbf{T}^*(\bar{\mathbf{y}}_i)L^*_e(\mathbf{y}_i\rightarrow\mathbf{y}_{i-1})d\mathbf{y}_1\cdots d\mathbf{y}_k. \qquad (10.3)$$

Here P denotes the set of all surface and volume points. This function can be approximated in a preprocessing step using Monte-Carlo random walks, which will be described in Section 10.1.1. If the generalized scattering function $f^*(.)$ is a constant, i.e. either resolves to a diffuse BRDF or an isotropic phase function, the directional dependency on $\mathbf{x}_1$ vanishes, and it can be included in the precomputation. This also means $\hat{L}^*(\mathbf{x}_1 \leftarrow \mathbf{y}_0)$ is constant and consequently can be represented by a (virtual) point light.

Replacing light subpaths in Equation 10.2 by the previous definition of a VPL (Equation 10.3) yields the IR formulation of indirect illumination and multiple scattering:

$$\hat{I}_j^* = \int_{\mathscr{P}_1 \times P} W_e(\mathbf{x}_0 \leftarrow \mathbf{x}_1) G_v^*(\mathbf{x}_0 \leftrightarrow \mathbf{x}_1) f^*(\mathbf{x}_0 \leftarrow \mathbf{x}_1 \leftarrow \mathbf{y}_0) G_v^*(\mathbf{x}_1 \leftrightarrow \mathbf{y}_0) \hat{L}^*(\mathbf{x}_1 \leftarrow \mathbf{y}_0) d\mu(\overline{\mathbf{x}}_1) d\mathbf{y}_0.$$

Rearranging terms and resolving generalized functions, we observe that in-scattering from VPLs in the medium is computed as follows:

$$\hat{L}_{\mathrm{M}}(\mathbf{x}_0 \leftarrow \mathbf{x}_1) = \int_P \sigma_s(\mathbf{x}_1) f_p(\mathbf{x}_0 \leftarrow \mathbf{x}_1 \leftarrow \mathbf{y}_0) G_v^*(\mathbf{x}_1 \leftrightarrow \mathbf{y}_0) \hat{L}^*(\mathbf{x}_1 \leftarrow \mathbf{y}_0) d\mathbf{y}_0. \quad (10.4)$$

Similarly, the reflected surface radiance is computed as:

$$\hat{L}_{\mathrm{S}}(\mathbf{x}_0 \leftarrow \mathbf{x}_1) = \int_P f_r(\mathbf{x}_0 \leftarrow \mathbf{x}_1 \leftarrow \mathbf{y}_0) G_v^*(\mathbf{x}_1 \leftrightarrow \mathbf{y}_0) \hat{L}^*(\mathbf{x}_1 \leftarrow \mathbf{y}_0) d\mathbf{y}_0. \quad (10.5)$$

Practical applications compute the radiance arriving at a pixel along eye rays parametrized by the image plane and the camera's origin $\mathbf{x}_0$, i.e. the measurement equation is reformulated as an integration over direction. Assuming $W_e(\mathbf{x}_0, \omega_0) = 1$ we obtain the radiance $\hat{L}^*(\mathbf{x}_0, \omega_0)$ at $\mathbf{x}_0$ arriving along an eye ray into direction ω_0 (Figure 10.2b).

$$\hat{L}^*(\mathbf{x}_0, \omega) = \underbrace{\int_0^t \tau(\mathbf{x}_0 \leftrightarrow \mathbf{x}_1) \hat{L}_{\mathrm{M}}(\mathbf{x}_0 \leftarrow \mathbf{x}_1) ds}_{\text{multiple scattering}} + \underbrace{\tau(\mathbf{x}_0 \leftarrow \mathbf{x}_1') \hat{L}_{\mathrm{S}}(\mathbf{x}_0 \leftarrow \mathbf{x}_1')}_{\text{indirect illum.}}, \quad (10.6)$$

where $\mathbf{x}_1'$ is determined using the ray casting function, i.e. $\mathbf{x}_1' = h(\mathbf{x}_0, \omega)$. The position $\mathbf{x}_1 = \mathbf{x}_0 + s\omega$ lies on the eye ray between the camera origin $\mathbf{x}_0$ and the visible surface point $\mathbf{x}_1'$ (Figure 10.2b).

Direct illumination and single scattering: Our derivation excluded direct illumination and single scattering, which can be trivially added again:

$$L^d(\mathbf{x}_0, \omega) = \underbrace{\int_0^t \tau(\mathbf{x}_0 \leftrightarrow \mathbf{x}_1) L_{\mathrm{M}}^d(\mathbf{x}_0 \leftarrow \mathbf{x}_1) ds}_{\text{single scattering}} + \underbrace{\tau(\mathbf{x}_0 \leftarrow \mathbf{x}_1') L_{\mathrm{S}}^d(\mathbf{x}_0 \leftarrow \mathbf{x}_1')}_{\text{direct illum.}}, \quad (10.7)$$

with in-scattering L^d_{M} from direct light sources:

$$L^d_{\mathrm{M}}(\mathbf{x}_0 \leftarrow \mathbf{x}_1) = \int_P \sigma_s(\mathbf{x}_1) f_p(\mathbf{x}_0 \leftarrow \mathbf{x}_1 \leftarrow \mathbf{y}_0) G^*_v(\mathbf{x}_1 \leftrightarrow \mathbf{y}_0) L^*_e(\mathbf{x}_1 \leftarrow \mathbf{y}_0) d\mathbf{y}_0.$$

Direct illumination is computed using the reflection integral (cf. Chapter 3).

$$L^d_{\mathrm{S}}(\mathbf{x}_0 \leftarrow \mathbf{x}_1) = \int_P f_r(\mathbf{x}_0 \leftarrow \mathbf{x}_1 \leftarrow \mathbf{y}_0) G^*_v(\mathbf{x}_1 \leftrightarrow \mathbf{y}_0) L^*_e(\mathbf{x}_1 \leftarrow \mathbf{y}_0) d\mathbf{y}_0.$$

Full light transport: With the previously defined terms, full light transport is simply the sum of indirect illumination and multiple scattering, and direct illumination and single scattering:

$$L(\mathbf{x}_0, \omega_0) = L^d(\mathbf{x}_0, \omega_0) + \hat{L}^*(\mathbf{x}_0, \omega_0). \tag{10.8}$$

10.1.1 Generating VPLs

As stated earlier, the function $\hat{L}^*(\mathbf{x}_1 \leftarrow \mathbf{y}_0)$ (Equation 10.3) can be precomputed and represented by VPLs. Practically, this is done using a particle tracing algorithm [Arvo and Kirk, 1990]. At each location a particle is reflected (surface) or scattered (volume), the algorithm creates a VPL. This process is described in detail by Pharr and Humphreys [2010] assuming vacuum, but the extension to participating media is trivial. For completeness it is described in the following:

1. First, a starting position from which to shoot a particle is sampled on the light source with probability $p(\mathbf{y}_0)$. Second, a shooting direction is sampled with probability $p(\omega_0)$. In the presence of participating media, a particle does not necessarily reach the point returned by the ray-casting function $h(\mathbf{y}_0, \omega)$ because it may be absorbed or scattered by the medium. Hence an additional free distance $t_k \leq \|\mathbf{y_0} - h(\mathbf{y}_0, \omega_0)\|$ along the tracing direction is sampled with probability $p(t_0)$. In homogeneous media, analytic sampling formulas exist [Lafortune and Willems, 1996]. In media with non-uniform density, rejection sampling techniques are used [Woodcock et al., 1965; Raab et al., 2008; Yue et al., 2010]. After determining these three factors, a particle

$$\alpha_1 = \frac{\tau(\mathbf{y}_0 \leftrightarrow \mathbf{y}_1)}{p(\mathbf{y}_0) p(\omega_0) p(t_0)}$$

is created at the position $\mathbf{y}_1 = \mathbf{y}_0 + t_0\omega_0$. Subscripts denote the number of bounces and scattering events that occurred so far. This particle is then turned into a VPL:

$$\hat{L}^*(\mathbf{x}_1 \leftarrow \mathbf{y}_1) = f_r^*(\mathbf{x}_1 \leftarrow \mathbf{y}_1 \leftarrow \mathbf{y}_0)\alpha_0 L_e^*(\mathbf{y}_0 \leftarrow \mathbf{y}_1).$$

As previously mentioned, the generalized scattering function $f^*(.)$ can be premultiplied, if it is constant. Otherwise this function can be explicitly stored together with the VPL for later evaluation.

2. To continue particle tracing, a new tracing direction ω_i and the free distance t_i ($t_i \leq \|\mathbf{y}_i - h(\mathbf{y}_i, \omega_i)\|$) are sampled with probabilities $p(t_i)$ and $p(\omega_i)$. Then a new particle

$$\alpha_{i+1} = \frac{1}{q_{i+1}} \frac{\tau(\mathbf{y}_i \leftrightarrow \mathbf{y}_{i+1}) f_r^*(\mathbf{y}_{i-1} \rightarrow \mathbf{y}_i \rightarrow \mathbf{y}_{i+1}) C^*(\mathbf{y}_i \rightarrow \mathbf{y}_{i+1})}{p(t_i)p(\omega_i)} \alpha_i$$

at $\mathbf{y}_{i+1} = \mathbf{y}_i + t_i\omega_i$ is created. The factor q_{i+1}^{-1} is introduced because of Russian roulette [Arvo and Kirk, 1990] which terminates particle tracing. Therefore an acceptance probability q_{i+1} is defined and a random number $\xi \in [0; 1]$ is drawn. If $\xi > q_{i+1}$, the particle α_{i+1} is discarded and particle tracing terminates. Otherwise it is accepted and a VPL is created:

$$\hat{L}^*(\mathbf{x}_1 \leftarrow \mathbf{y}_{i+1}) = f_r^*(\mathbf{x}_1 \leftarrow \mathbf{y}_{i+1} \leftarrow \mathbf{y}_i)\alpha_{i+1} L_e(\mathbf{y}_0 \rightarrow \mathbf{y}_1).$$

3. After N particles have been traced (and $M \geq N$ VPLs have been created), each VPL is reweighted:

$$\hat{L}^*(\mathbf{x}_1 \leftarrow \mathbf{y}_i) = \frac{\hat{L}^*(\mathbf{x}_1 \leftarrow \mathbf{y}_i)}{N}.$$

10.1.2 Rendering with VPLs

After particle tracing, in-scattered light from VPLs must be accumulated along the eye ray and (attenuated) indirect illumination must be computed (Equation 10.6).

Indirect illumination: Indirect illumination (Equation 10.5) is approximated accumulating and attenuating the contribution of all VPLs:

$$\hat{L}^*(\mathbf{x}_0 \leftarrow \mathbf{x}_1) \approx \tau(\mathbf{x}_0 \leftrightarrow \mathbf{x}_1) \sum_{j=1}^{M} f_r(\mathbf{x}_0 \leftarrow \mathbf{x}_1 \leftarrow \mathbf{y}_j) G^*(\mathbf{x}_1 \leftrightarrow \mathbf{y}_j) V^*(\mathbf{x}_1 \leftrightarrow \mathbf{y}_j) \hat{L}^*(\mathbf{x}_1 \leftarrow \mathbf{y}_j)$$

Here $\mathbf{y}_j$ is the position of the j-th VPL. The contribution of indirect illumination is attenuated through the medium (Equation 10.6), and we must compute transmittance $\tau(\mathbf{x}_0 \longleftrightarrow \mathbf{x}_1)$ to account for the fraction of light that reaches $\mathbf{x}_0$. In a homogeneous medium, we can compute this factor analytically, whereas we require numerical approaches in media with non-uniform density. Therefore we can either use ray-marching [Perlin and Hoffert, 1989] or Woodcock tracking [Jarosz et al., 2011].

Multiple scattering: Inside the volume the illumination from all VPLs must be integrated along the eye rays. For this, we use Monte-Carlo integration with importance sampling. We could employ the method of Kulla and Fajardo [2011] and sample according to the (unbounded) geometry term, which is analytic and thus efficient. Although originally developed for computing single scattering from area light sources, this method is also suitable to accumulate the contribution of a VPL along an eye ray. Sampling according to the unbounded geometry term places samples on the eye ray, where a VPL has high contribution. However, as we clamp the geometry term (cf. Section 10.1.3), this sampling strategy would oversample the region in which clamping occurs. Instead, we sample according to the transmittance along the ray (i.e. either analytically or using Woodcock tracking), as this can significantly contribute to reducing the variance of the Monte-Carlo estimator:

$$L(\mathbf{x}_0, \omega_0) \approx \sum_{s=0}^{S} \sum_{j=0}^{M} \frac{\tau(\mathbf{x}_0 \longleftrightarrow \mathbf{x}_1) f_p(\mathbf{x}_0 \longleftrightarrow \mathbf{x}_1 \longleftrightarrow \mathbf{y}_j) G_v^*(\mathbf{x}_1 \longleftrightarrow \mathbf{y}_j) \hat{L}^*(\mathbf{x}_1 \longleftarrow \mathbf{y}_j)}{S\ p(t_1)} \quad (10.9)$$

Here $\mathbf{x}_1 = \mathbf{x}_0 + t_0 \omega_0$ denotes a sample on the eye ray, and S denotes the total number of samples that are used for each of the M VPLs. $\hat{L}^*(\mathbf{x}_1 \longleftarrow \mathbf{y}_j)$ is the emitted radiance of the j-th VPL at position $\mathbf{y}_j$ towards the sample $\mathbf{x}_1$ on the eye ray.

10.1.3 Bias Compensation

Lighting as described in Section 10.1.2 leads to unbiased solutions, but suffers from bright splotches in the rendering due to a singularity in the generalized geometry term. In order to avoid these artefacts, the geometry term is bounded:

$$G_b^*(\mathbf{x} \longleftrightarrow \mathbf{y}) = \min(G^*(\mathbf{x} \longleftrightarrow \mathbf{y}), b),$$

using some bound $b > 0$, which defines the spherical *bounding region* in the volume with radius $r = G^{\frac{1}{2}}$. This bound depends on the size of the scene and the number of VPLs that have been created [Kollig and Keller, 2006]. Unfortunately, it also introduces bias because it eliminates a certain amount of transport energy with in this region. Fortunately, there exists a closed formula to account for the energy loss, i.e. the bias [Raab et al., 2008]:

$$L_r^*(\mathbf{x}\leftarrow\mathbf{y}) = \int_P f^*(\mathbf{x}\leftarrow\mathbf{y}\leftarrow\mathbf{z})\max(0, G^*(\mathbf{y}\leftrightarrow\mathbf{z}))V^*(\mathbf{y}\leftrightarrow\mathbf{z})L(\mathbf{y}\leftarrow\mathbf{z})d\mathbf{z}. \tag{10.10}$$

Equation 10.10 is an integration over area and volume, i.e. the singularity is still present. However, it can be avoided reformulating Equation 10.10 as an integration over direction. Then with $\mathbf{z} = \mathbf{y} + t\omega$ it becomes:

$$L_r^*(\mathbf{x}\leftarrow\mathbf{y}) = \int_0^u\int_\Omega f^*(\mathbf{x}\leftarrow\mathbf{y}\leftarrow\mathbf{z})B^*(\mathbf{y}\leftrightarrow\mathbf{z})\tau(\mathbf{y}\leftrightarrow\mathbf{z})C^*(\mathbf{y}\leftrightarrow\mathbf{z})L(\mathbf{y}\leftarrow\mathbf{z})dt\,d\omega. \tag{10.11}$$

Here $u = \|\mathbf{y} - h(\mathbf{y}, \omega)\|$ is the closest intersection along the ray with origin $\mathbf{y}$ and direction ω. Raab et al. [2008] showed that this reformulation introduces the weighting factor

$$B^*(\mathbf{y}\leftrightarrow\mathbf{z}) = \frac{\max(0, G^*(\mathbf{y}\leftrightarrow\mathbf{z}) - b)}{G^*(\mathbf{y}\leftrightarrow\mathbf{z})}$$

which accounts for the compensated energy. By the means of this correction term, unbiased global illumination is the sum of direct illumination and single scattering $L^d(\mathbf{x}\leftarrow\mathbf{y})$, bounded indirect illumination and multiple scattering from VPLs $L_b^*(\mathbf{x}\leftarrow\mathbf{y})$, and the compensation term. Hence it becomes:

$$L(\mathbf{x}\leftarrow\mathbf{y}) = L^d(\mathbf{x}\leftarrow\mathbf{y}) + L_b^*(\mathbf{x}\leftarrow\mathbf{y}) + L_r^*(\mathbf{x}\leftarrow\mathbf{y}) \tag{10.12}$$

The definition of the correction term (Equation 10.11) depeonds on the unbiased solution $L(\mathbf{y}\leftarrow\mathbf{z})$ computed according to Equation 10.12 which implies that compensation is evaluated recursively and the entire algorithm starting at a shading point $\mathbf{x}_0$ becomes [Kollig and Keller, 2006; Raab et al., 2008]:

1. Compute illumination from primary light sources and bounded illumination from all VPLs at point $\mathbf{x}_0$.

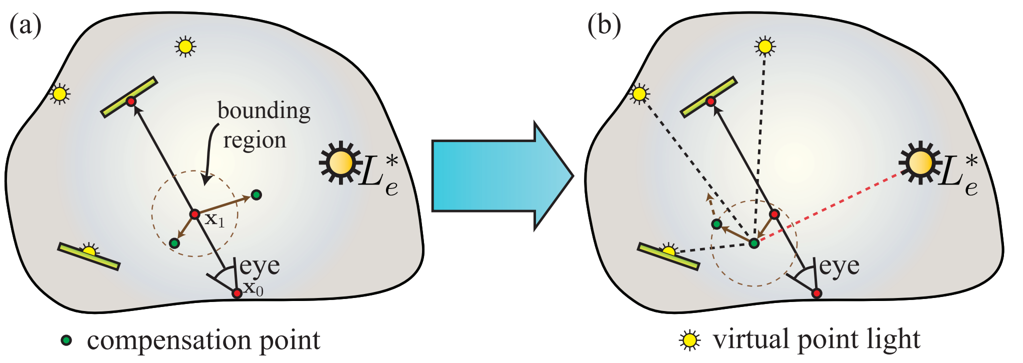


Figure 10.3: *(a) Bias compensation creates compensation points for each shading point. (b) Compensation is only computed for points within the bounding region. Therefore illumination from primary and secondary light sources are accumulated, and compensation continues recursively.*

2. Compute the bias numerically according to Equation 10.11. Therefore a ray is shot into a random direction ω and a free path with distance t is sampled to find the compensation point $\mathbf{x}_1 = \mathbf{x}_0 + t\omega$. Because compensation depends on an unbiased solution of light transport, it is computed repeating the algorithm starting with step 1 at $\mathbf{x}_1$. This recursion terminates when weighting factor $B(\mathbf{x}_i \longleftrightarrow \mathbf{x}_{i+1})$ evaluates to 0.

10.2 Approximate Bias Compensation

The bias compensation technique according to Raab et al. [2008] is very costly as it requires recursive ray casting to find compensation points, access to all VPLs at every compensation point, and in the worst case, it degenerates to (bidirectional) path tracing [Davidovič et al., 2010].

In this section we analyse the energy recovery due to bias compensation and derive several optimisations by analysing Equation 10.11, different sampling strategies, and simplifying assumptions. In this way we can derive a more efficient, *approximate bias compensation* (ABC) technique that yields results visually indistinguishable from ground truth. We illustrate and motivate our proposed simplifications for ABC using plots of the brightness of the scanline highlighted in Figure 10.4 and renderings with different settings to assess the image quality (Figures 10.5, 10.6, and 10.8).

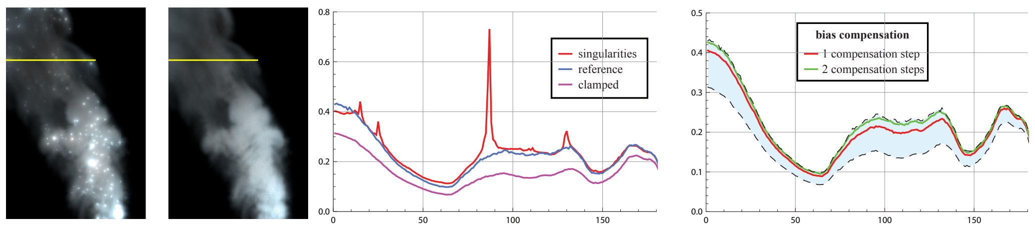


Figure 10.4: *Plots show the highlighted scanline. As can be seen from the centre plot, unbounded accumulation of VPLs causes high intensity peaks. Clamping, however, removes energy which darkens the image significantly. The left plot shows that after two compensation steps most energy is recovered and the result is indistinguishable from the reference (computed with full bias compensation). Illustration: Engelhardt et al. [2012].*

10.2.1 Limiting Recursion Depth

Bias compensation is a recursive process because the bounded contribution of VPLs is gathered at each compensation point. However, the compensation integral convolves the gathered radiance with the generalised scattering function f^*, thus removing high frequencies. More importantly, the contribution drops exponentially with each additional compensation step. Figure 10.4 shows that already after one compensation step most of the eliminated energy has been recovered and matches the behaviour of the ground truth curve quite well. It can also be seen that after two compensation steps the result is virtually indistinguishable from the ground truth solution. The conclusions we draw from this observation is that we limit recursion depth in our bias compensation algorithm to three steps at most.

10.2.2 Path Generation and Locally Homogeneous Media

To create new compensation points Raab et al. [2008] choose a random ray with a direction ω and sample a distance along it using free path sampling, e.g. Woodcock tracking [Woodcock et al., 1965]. This strategy creates compensation points with possibly zero contribution because they might be created outside the clamping region, i.e. $B(.) = 0$. This in turn leads to high variance in the compensation estimator. The variance can be reduced creating substantially more compensation points, but it also implies a substantial sampling overhead as there is no possibility in *heterogeneous* participating media to restrict sampling according to the transmittance within a given distance. We avoid this issue by introducing the assumption that the medium is *locally homogeneous* around the compensation point.

One key ingredient to our ABC is that we generate the new compensation points always inside the bounding region where clamping occurs. The radius d of the spherical bounding region can be derived from the bound b as $d = b^{-\frac{1}{2}}$. Assuming a *locally homogeneous* medium with extinction coefficient $\overline{\sigma}_t$, the probability density function for sampling a distance t (for the new compensation point $\mathbf{x}_{i+1}$) from $\mathbf{x}_i$ within the bounding region reads:

$$p(t) = \frac{\overline{\sigma}_t e^{-\overline{\sigma}_t t}}{1 - e^{-\overline{\sigma}_t d}}. \tag{10.13}$$

Using the inversion method we compute the distance as:

$$t = \frac{\ln(1 - \xi(1 - e^{-\overline{\sigma}_t d}))}{\overline{\sigma}_t}, \tag{10.14}$$

with a uniform random number $\xi \in [0;1)$. For $\overline{\sigma}_t$ we use the average extinction coefficient within the bounding region, which can be efficiently obtained from a downsampled version of the medium, i.e. if stored in a 3D texture. Note that if there is a surface intersection occurring closer to $\mathbf{y}$ than t, this intersection becomes the new compensation point. The assumption of local homogeneity does not compromise the results. In fact, it only affects the placement of the new vertex $\mathbf{x}_{i+1}$, and the computation remains unbiased as long as we correctly evaluate the transmittance $\tau(\mathbf{x}_i \longleftrightarrow \mathbf{x}_{i+1})$. Nevertheless, we take the assumption one step further and also approximate the transmittance $\overline{\tau}(\mathbf{x}_i \longleftrightarrow \mathbf{x}_{i+1}) = \exp(-\overline{\sigma}_t \left\|\mathbf{y} - \mathbf{y}'\right\|)$, effectively avoiding costly evaluation (e.g. by Woodcock tracking [Jarosz et al., 2011]).

In all our test scenes, this yielded results visually indistinguishable from ground truth (see Fig. 10.5), although the assumption can theoretically fail at locations with strongly varying extinction. Note that these locations could be easily detected if necessary using the gradient of $\sigma_t(\mathbf{y})$.

10.2.3 Integration Strategies

In this section we analyze different sampling strategies for bias compensation. Each of the strategies focuses the computation on different parts of the compensation. In Fig. 10.6 we compare the results at roughly equal rendering time. A corresponding plot of the RMSE induced by the different strategies is shown in Fig. 10.7.

1-to-N: Raab et al. [2008] propose to connect each compensation vertex to all VPLs. This has two implications. First, because of free path sampling along the compensation ray, the compensation point may be sampled

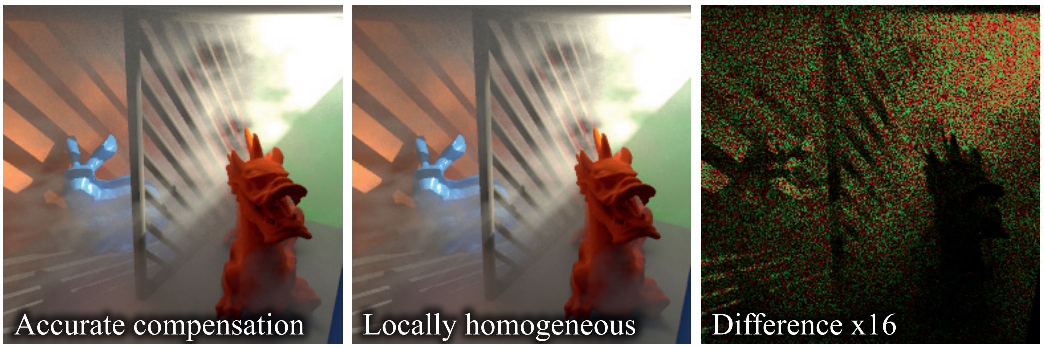


Figure 10.5: *Bias compensation with accurate transmittance, using our locally homogeneous assumption, and the* 16× *difference (green means too dark, red means too bright). Illustration: Engelhardt et al. [2012].*

outside the bounding region, and compensation is entirely discarded. This approach can lead to high variance, since the compensation integral (Equation 10.11) is severely under-sampled (Figure 10.6 1-to-N). Second, each compensation step requires access to *all* VPLs, making the approach GPU-unfriendly because a shadow map must be stored for each VPL.

N-to-1: We found that noise can be substantially reduced by creating more compensation vertices but connecting each of them to only one VPL (Figure 10.6 N-to-1). Unfortunately, this increases the number of rays that need to be traced thereby increasing run-time significantly. To ensure a fair comparison, we adjusted the number of eye ray samples to achieve roughly equal rendering time for all compensation strategies. However, as shown in the equal-time comparison in Fig. 10.6, the N-to-1 approach still exhibits lower variance than [Raab et al., 2008].

1-to-1: By always creating only a single compensation vertex connected to the VPL that is currently evaluated along the primary ray, we significantly reduce the cost of the compensation, and thus we can take more samples along the eye ray and further reduce the noise due to transmittance. This approach is also GPU-friendly because VPLs can be processed independently: we can compute the contribution of a single VPL to all pixels (including BC), accumulating the results progressively over time. Consequently, we need to store only a single shadow map in memory at a time.

1-to-1 locally homogeneous: The 1-to-1 approach discards many compensation rays because of free path sampling. Ideally we would like to

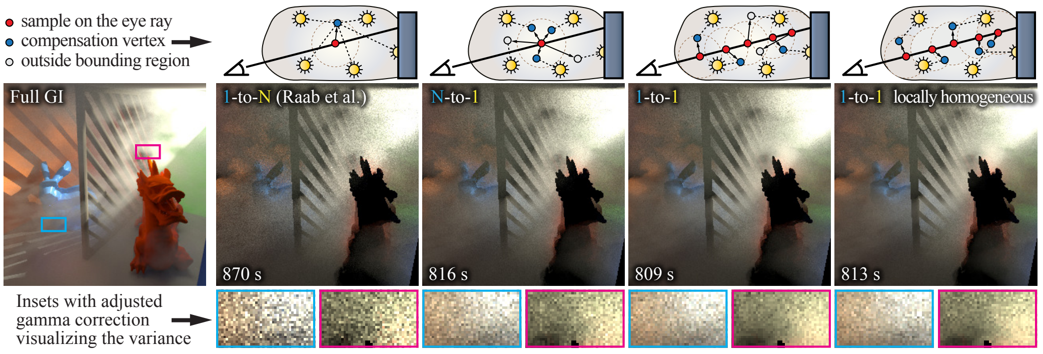


Figure 10.6: *Different compensation strategies: Raab et al. connect all VPLs to a single compensation vertex (1-to-N). Lower variance can be achieved by generating more vertices while connecting each of them to a single VPL (N-to-1). A similar, but GPU-friendly approach is to generate only one vertex connected to a single VPL (1-to-1). By assuming a locally homogeneous medium we avoid expensive evaluation of the transmittance and ensure that vertices are always created within the bounding region, thus minimizing divergent code paths on the GPU. Rendering time is controlled by adjusting the number of samples along the eye ray: 3, 1, 115, and 78 for 1-to-N, N-to-1, 1-to-1, and 1-to-1 locally homogeneous, respectively. The 1-to-1 discards roughly half of the compensation vertices compared to the loc. hom. approach in this case. Illustration: Engelhardt et al. [2012].*

create all compensation vertices within the bounding region. This can be assured using our locally homogeneous assumption. Additionally, it ensures that parallel execution paths do not become divergent, which is favourable for GPU implementation.

10.2.4 Omitting Visibility and Local Visibility

Computing visibility between a point **x**, that requires compensation, and a compensation vertex $\mathbf{x}'$ is crucial for overall performance. Obviously, the path **x** to $\mathbf{x}'$ can only be occluded when **x** is close to a surface, but not in "free space".

Figure 10.8a illustrates the case when not computing visibility causes artefacts. To asses how often these artefacts appear and their influence on the resulting images, we set up a series of experiments. Interestingly, it was not easy to produce visible artefacts at all – this can be explained by considering the circumstances that have to coincide to cause artefacts (Figure 10.8a): (1) **x** must be close to a thin opaque object and (2) the

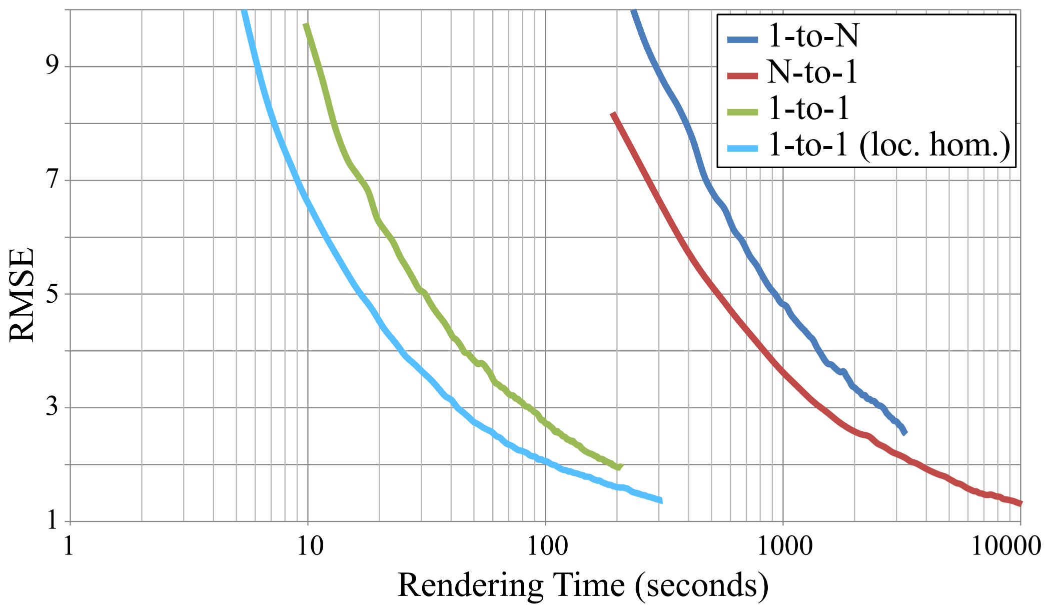


Figure 10.7: *RMSE plot for different compensation strategies (see Sec. 10.2.3) used in Figure 10.6, computed against a reference solution (converged* 1-to-N *strategy proposed by Raab et al. [2008]). The* 1-to-1 *strategy clearly outperforms other strategies. Illustration: Engelhardt et al. [2012].*

medium must not be too dense otherwise sampling a distance through the opaque object is unlikely. Note that if $\mathbf{x}$ and $\mathbf{x}'$ are further apart (yet within radius r), the quadratic decrease of the compensation term with the distance also suppresses light bleeding. Figure 10.8b shows one of our test scenes; artefacts become visible only after scaling the brightness by at least 2 orders of magnitude. Note that a somewhat similar assumption (ignoring visibility on short distances) has also been used for global illumination on surfaces [Arikan et al., 2005; Davidovič et al., 2010].

10.3 Implementation Details

We integrated our approximate bias compensation technique in a custom offline renderer to evaluate our assumptions (Figures 10.4, 10.5, 10.6, 10.8). For further acceleration we implemented a progressive GPU renderer using Direct3D 11 (Figures 10.9, 10.10, 10.11, 10.12, 10.13).

Random walks for creating VPLs are always carried out on the CPU using ray tracing. For acceleration, we use a kD-tree built with the surface area heuristic.

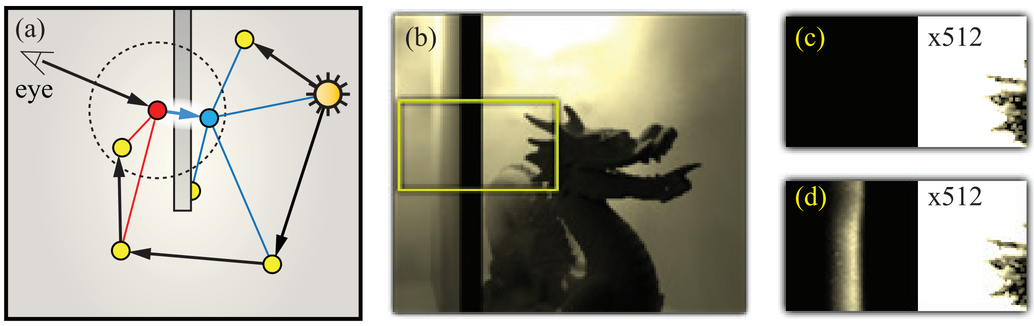


Figure 10.8: *Compared to the accurate computation (c) ignoring visibility to compensation vertices causes artefacts (d) only revealed with high scaling (×512). Illustration: Engelhardt et al. [2012].*

The CPU implementation of our method is shown as pseudo code in Section10.7. In this section, we restrict ourselves to the peculiarities of the GPU implementation, which is outlined in Figure 10.9. We split the computations into evaluating contributions from primary light sources and from secondary light sources, i.e. VPLs. First we render a geometry buffer filled with all relevant information such as BRDFs, positions, and normals. Afterwards we evaluate single scattering and direct illumination with visibility computed using shadow maps (with resolutions of 512^2 up to 4096^2). Transmittance towards the light sources is evaluated analytically in case of homogeneous media and numerically in heterogeneous media using ray marching (the offline renderer uses slower but unbiased Woodcock tracking).

VPL lighting: After the contributions from primary light sources have been computed, our renderer iterates over all VPLs and accumulates their contribution, one per iteration. Apart from conventional shadow maps, we also compute a variant of adaptive volumetric shadow maps [Salvi et al.]. Since lookups into regular AVSMs involve a costly search for nearby transmittance samples, we resample the AVSM at fixed intervals. Although this approach might miss prominent features in the transmittance function, we have not noticed any artefacts arising from this approximation. Looking up the transmittance then boils down to a single access to an array of fixed size. Finally, we accumulate the contribution of a VPL to the surfaces and the volume. The latter we evaluate using adaptive ray marching, i.e. the number of samples depends on the ray segment that intersects the volume.

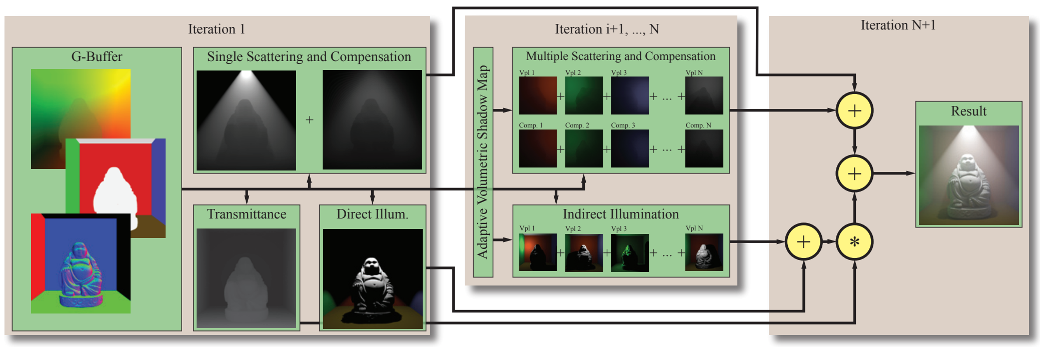


Figure 10.9: *Data flow in our progressive GPU renderer. The first iteration computes the geometry buffer, single scattering with bias compensation from primary light sources, transmittance and direct illumination. In each subsequent iteration the contribution of one VPL to multiple scattering, bias compensation and indirect illumination is accumulated. The final step composes all individual results to the final image. Illustration: Engelhardt et al. [2012].*

Bias compensation: We split the compensation integral into two terms: one for compensating from the primary light sources and the second gathering compensation energy from VPLs. This allows us to evaluate compensation from the primary lights directly in the single scattering shader, which is executed only once in the first iteration. For that we generate a buffer that contains a set of random directions and additional random numbers used to sample a distance along the compensation ray. Our assumption of locally homogeneous medium and neglecting visibility allow us to sample the new compensation vertices *always* within the bounding region. This, in contrast to the original BC Kollig and Keller [2006], avoids branching and divergent execution paths, substantially accelerating the GPU implementation. Bias compensation gathering illumination from VPLs is handled in a similar spirit: for each VPL and each sample point along the ray we create a single vertex within the bounding region and compute its contribution to the correction term.

10.4 Results

We evaluated our method using several test scenes with homogeneous and heterogeneous participating media. All timings have been recorded using an Intel Core i7 6-core hyperthreading system with 3.2GHz and a GeForce GTX 580 GPU.

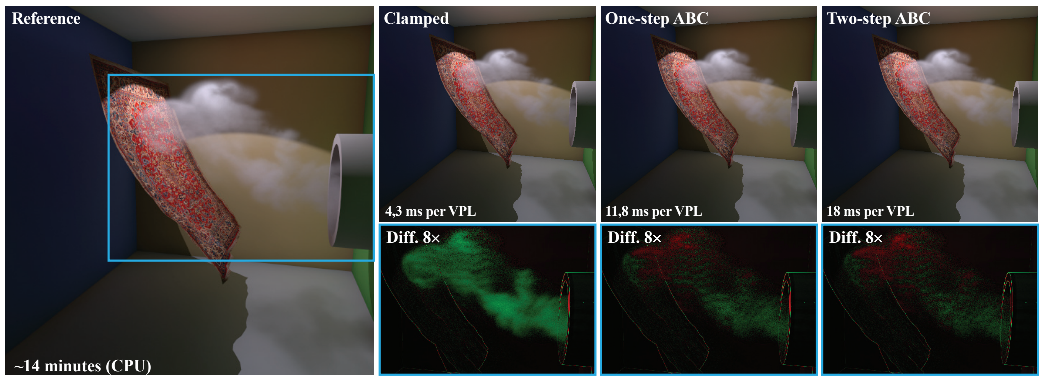


Figure 10.10: *Room with a heterogeneous smoke ($\sigma_s = 0.9$, $\sigma_a = 0.001$) rendered with 6800 VPLs. Clamping removes a remarkable amount of energy, which is almost completely recovered using just one ABC step. The insets visualise lost energy (green), and overcompensation (red). Differences on the edges are due to different sampling of primary visibility on the CPU and GPU. Illustration: Engelhardt et al. [2012].*

Accuracy: Figure 10.10 shows a visual comparison of our approximate bias compensation algorithm. The reference image with 512^2 pixels has been computed with full bias compensation [Raab et al., 2008] in the participating medium and 6800 VPLs in the entire scene. As can be seen, our approximate bias compensation recovers most of the lost energy already after the first compensation step.

Comparison to photon mapping: In Figure 10.11 we show an equal-time comparison between our bias compensation algorithm executed on the GPU and photon mapping using the beam radiance estimate [Jarosz et al., 2008] for images with 1024^2 pixels. Photon mapping and the beam radiance estimate (BRE) require hierarchical data structures which do not map well to the GPU, and we used our CPU implementation. The reference image was computed using full bias compensation. For the equal-time comparison we fixed the number of VPL paths to 1536 yielding 7887 VPLs, and that of the volume photons to 1 million. In order to match rendering times, we controlled the performance of the photon mapper by setting the maximum number of radiance estimation photons to be searched for (using adaptive nearest neighbour search) to 117. Shooting photons and building the *k*D-tree took 3 seconds. The BRE requires an additional search data structure which is constructed in 22 seconds. Rendering using beam queries took 110 seconds for 12 rendering threads. The ABC method finished in 125 seconds.

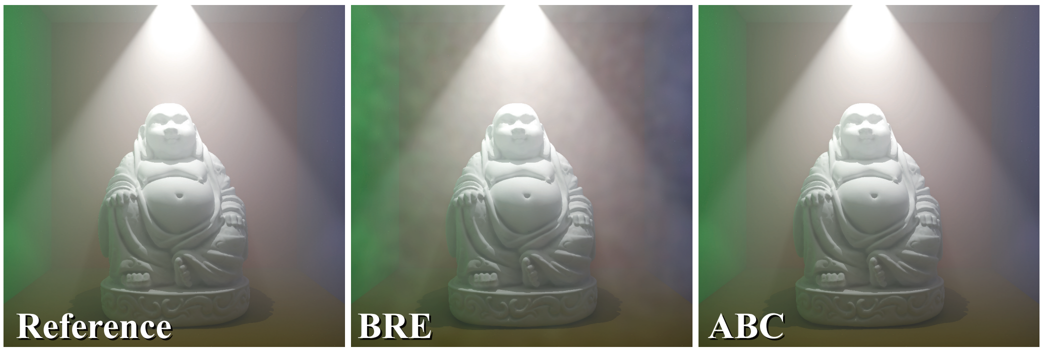


Figure 10.11: *Equal-time comparison for volumetric illumination. Left: reference CPU solution with full bias compensation. Middle: beam radiance estimate (BRE) Jarosz et al. [2008] with 1 million volume photons. Right: GPU-accelerated ABC with 7887 VPLs and 2 approximate compensation steps. Illustration: Engelhardt et al. [2012].*

Photon mapping still shows the typical artefacts that arise from an insufficient number of photons in the volume or used in the estimate, while ABC, in contrast, is nearly indistinguishable from the reference solution.

Anisotropic media: Our algorithm also supports anisotropic scattering media, as shown in Figure 10.12. This scene was rendered at 1024^2 with 4320 VPLs and two bounces of compensation. The average shading time per VPL was 16 ms.

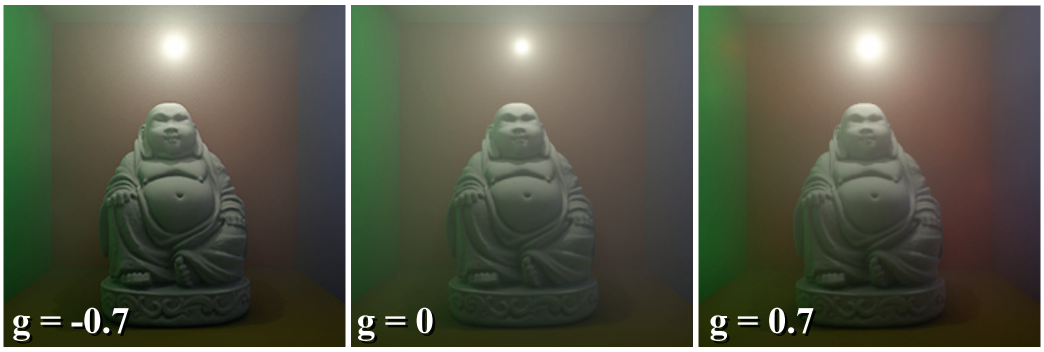


Figure 10.12: *The Buddha in a homogeneous medium (*$\sigma_s = 0.075$*,* $\sigma_a = 0.001$*) with varying g parameter of the Henyey-Greenstein phase function. From left to right, images are rendered with backward, isotropic, and forward scattering medium. Illustration: Engelhardt et al. [2012].*

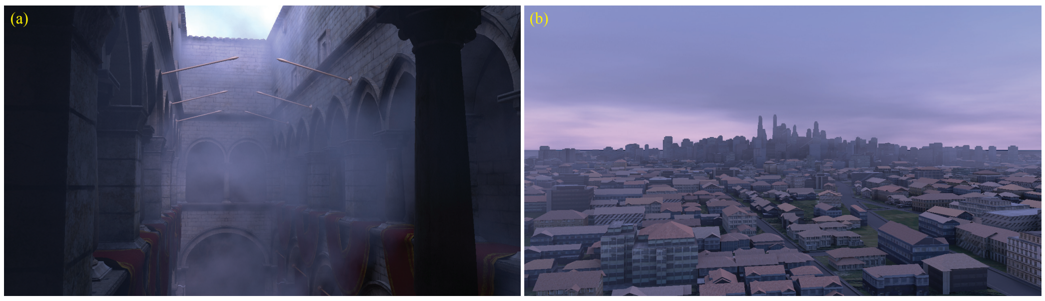


Figure 10.13: *ABC supports full multiple scattering in complex environments with image-based illumination. (a) Indirect illumination and multiple scattering in Crytek Sponza (262k triangles) rendered with 118k VPLs, two-bounce compensation. (b) City scene (823k triangles) with image-based illumination. Geometry and clouds are procedurally generated and both accounted for in light transport. Illustration: Engelhardt et al. [2012].*

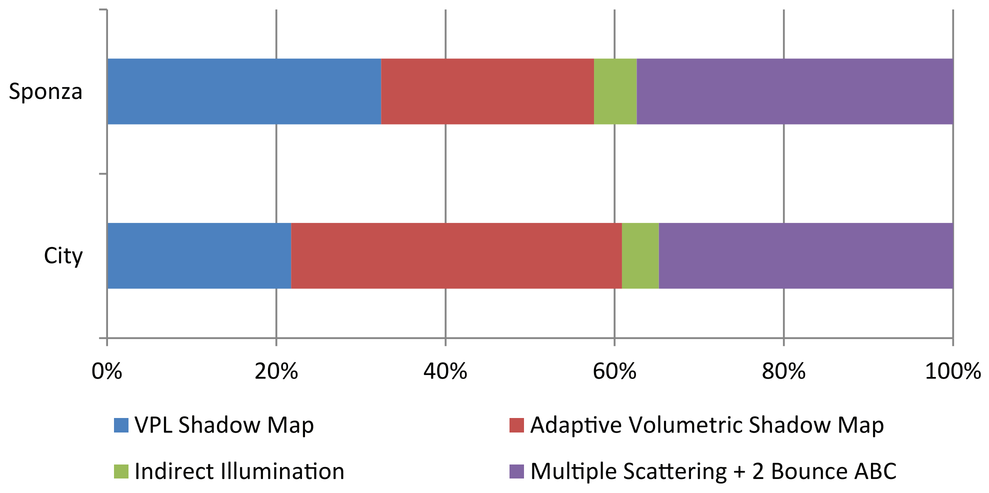


Figure 10.14: *Shading time per VPL for the city and sponza scenes: Indirect illumination is negligible. Computing (volumetric) shadow maps is costly and accounts for 60% in the city scene. Multiple scattering with two bounces of approximate bias compensation accounts for about 35 — 38% of the entire cost. Illustration: Engelhardt et al. [2012].*

Complex scenes: We also tested our algorithm with complex scenes and image-based lighting (Figure 10.13) for high quality rendering. Both the Sponza scene (262k triangles, 118k VPLs) and the city scene (823k triangles, 108k VPLs) were rendered with 2 bounces of ABC. The shading costs per VPL vary with the complexity of the scene (number of triangles) and are mainly due different resolutions of the 3D density textures for the heterogeneous participating medium. A detailed analysis of the per-VPL shading costs is shown in Figure 10.14.

10.5 Discussion

In this section we describe findings from experimenting with our method, which we believe are important to assess its strengths and limitations and discuss its robustness and use in complex scenes.

IR and participating media: Similar to IR for surfaces we *sub-sample the path space*. The result images are typically close to ground truth because single scattering and transmittance are computed at high precision using ray marching, which preserves crisp features. The quality of the “global” light transport heavily depends on the number of VPLs. We observe that more VPLs are required for dense and heterogeneous than for thin or homogeneous media. For surfaces, the link between the scene geometry and materials and the number of required VPLs has been extensively studied by Křivánek et al. [2010]. Deriving similar dependency on the parameters of the medium is an interesting future work.

Complex scenes: Scenes with large extent benefit from bi-directional VPL generation [Segovia et al., 2006a] to create enough VPLs that actually contribute to the image. In our implementation we use a variant of the method of Georgiev and Slusallek [2010]: we create a larger number of paths and keep only those VPLs that contribute the most to the image. This is evaluated by simply considering whether they are close to the camera. This simple approach is surprisingly well-suited for scenes with participating media, such as those in Fig. 10.13.

Phase functions: Our method supports *anisotropic phase functions*, but strong backward or forward scattering causes problems similar to glossy materials with IR. This is because sub-sampling the path space assumes smooth illumination, which is only valid for isotropic and moderately anisotropic scattering. To achieve artefact-free renderings of highly anisotropic media, a large number of VPLs is required.

Animated scenes: Temporal changes to primary light sources or scene geometry inherently change the distribution of VPLs in each frame. This becomes noticeable as flickering if an insufficient number of VPLs is used. Since our method directly depends on the VPL distribution, this effect is carried over. To reduce the temporal artefacts, a sufficient number of VPLs is necessary (several thousand for small scenes). To further improve temporal coherence, the VPLs can be distributed using the same sequence of random numbers in every frame.

10.6 Conclusions and Future Work

We have presented a novel method for rendering global illumination including multiple scattering in heterogeneous media, which is based on instant radiosity, thus requiring no precomputation. Key to our method is the approximate bias compensation technique thaxt enables rendering images close to ground truth. While the visual impact of all our approximations is indistinguishable from the original BC, our technique is more efficient and also amenable to GPU acceleration.

Unfortunately, due to the high number of VPLs required for good results, our technique is not interactive. This is mainly because typical low-resolution rendering techniques are not applicable to volumetric rendering. For future work we like to research these techniques and devise new methods, that make upsampling and interpolation of volumetric illumination in screen space possible, which is an important step towards interactivity. Then our approximate bias compensation could be easily combined with our GPU-friendly scree-space bias compensation for compensation on surfaces (cf. Chapter 11) to obtain full global illumination including short distance light transport between surfaces and media.

10.7 ABC Pseudo Code

The following pseudo code outlines the computation of compensation energy at an arbitrary point. For brevity we do not include surface compensation, which is thoroughly described in Kollig and Keller [2006]. We focus on the new N-to-1 and 1-to-1 strategies.

```
Color compensate(Point p, VPL vpl, int rec) {

  if(isSurface(p))
    return compensateSurface(p,vpl,rec);

  Direction w = samplePhaseFunction(p);
  Surface   s = infinity;
  Distance  d = 0;
  Ray     ray = Ray(p,w);

  // Limiting recursion depth (Section 4.1)
  if(rec > 3)
    return 0;

  // Ignore Visibility (Section 4.4)
  if(!ignoreVisibility)
    s = intersect(ray);

  // Local homogeneous assumption (Section 4.2)
  d       = sampleDistance(ray, isLocallyHomogeneous);
  Color T = transmittance(ray, d, isLocallyHomogeneous);

  // Phase Function
  Color P = phaseFunction(p,w);

  Point next_p = p + d * w;
  if(isOutsideBoundingRegion(next_p,p))
     return 0;

  // Compensation weight (Raab et al.)
  Real G = 1;
  if(isSurface(next_p,s))
    G = dot(NormalAt(next_p),-w);

  G /= (d*d);
  G  = max(0,G-bound) / G;

  if(1-to-1)
  {
     // 1-to-1 (Section 4.3)
     return G * P * T * (evalVplAt(next_p) + compensate(next_p, vpl, rec+1))
       / (pdf(d) * pdf(w));
  }
  else
  {
    // N-to-1 (Section 4.3)
    Color C = none;
    foreach(VPL v)
      c += evalVplAt(next_p, v) + compensate(next_p, v, rec+1);

    return G * P * T * C / (pdf(d) * pdf(w) * N_vpls);
  }
}
```

CHAPTER 11

Screen-Space Bias Compensation

IN the previous chapter we have exploited the amenity of instant radiosity [Keller, 1997] (IR) to derive a highly efficient, GPU-friendly algorithm for rendering multiple scattering in the presence of (heterogeneous) participating media. As pointed out, volume illumination with VPLs suffers from singularities due to the illumination with virtual point lights (VPLs). The same is true for lighting nearby surfaces: lighting from VPLs causes bright splotches due to the singularity in the illumination from point lights. Those artefacts are typically suppressed by bounding (or clamping) the contribution of a VPL to shading points in its proximity, eventually trading one artefact for another. Bounding removes energy from the light transport, which on surfaces becomes visible especially near edges, corners, and creases, leading to incorrect darkening in such regions.

In Chapter 10 we developed reasonable approximations to efficiently correct the bias for volume illumination based on the generalized compensation term [Raab et al., 2008]. This approach cannot be directly applied to surface compensation because it depends on light reflected from other surfaces, i.e. visibility cannot be ignored. Then, bias compensation requires Monte-Carlo ray tracing which is infeasible for efficient solutions without sophisticated spatial index structures.

In this chapter we present a GPU-friendly algorithm for bias compensation in screen space that recovers energy lost due to clamping from nearby surfaces in screen space.[1] This becomes possible reformulating the rendering equation which allows us to cast the bias compensation onto a simple post-process. This can be accelerated on modern GPUs, and we show results nearly indistinguishable from ground truth rendered at interactive speed.

[1]This chapter is based on our joint work [Novák et al., 2011] presented at the *Symposium on Interactive 3D Graphics and Games 2011 (I3D'11)*.

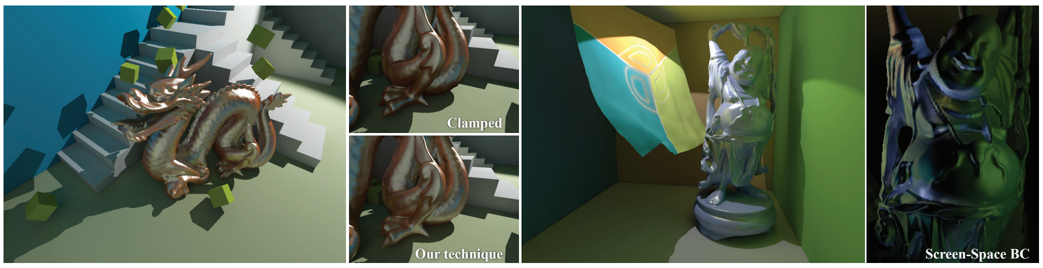


Figure 11.1: *Our SSBC algorithm improves the quality for VPL methods, which typically clamp the contributions of VPLs to avoid artefacts. SSBC recovers the thereby lost short distance illumination operating only from illumination present in screen space. This correction takes only about 27 ms in both scenes, which render at interactive speed including all lighting effects (Dragon scene: 10 fps, Happy Buddha: 7.2 fps). Both results are nearly indistinguishable from offline renderers which are magnitudes slower. Illustration: Novák et al. [2011].*

11.1 Global Illumination with Instant Radiosity

Recall from Chapter 3.2 that light transport can be formulated as a Neumann series

$$L = \sum_{n=0}^{\infty} \mathbf{T}^n L_e.$$

For our purposes, the transport operator is the shading integral defined over area using the three point notation [Veach, 1997]:

$$(\mathbf{T}L)(\mathbf{x}{\leftarrow}\mathbf{y}) = \int_A f_r(\mathbf{z}{\to}\mathbf{y}{\to}\mathbf{x}) V(\mathbf{y}{\longleftrightarrow}\mathbf{z}) G(\mathbf{y}{\longleftrightarrow}\mathbf{z}) L(\mathbf{z}{\to}\mathbf{y}) d\mathbf{z}.$$

This operator depends on the BRDF $f_r(\mathbf{z}{\to}\mathbf{y}{\to}\mathbf{x})$, the binary visibility term $V(\mathbf{y}{\longleftrightarrow}\mathbf{z})$, and the geometry term

$$G(\mathbf{y}{\longleftrightarrow}\mathbf{z}) = \frac{\cos^+\theta_{\mathbf{y}\to\mathbf{z}} \cos^+\theta_{\mathbf{z}\to\mathbf{y}}}{\|\mathbf{y}-\mathbf{z}\|^2}.$$

11.1.1 Instant Radiosity and Bias Compensation

Instant radiosity [Keller, 1997] computes global illumination in two passes. In the first step, IR distributes VPLs by the means of Monte-Carlo particle

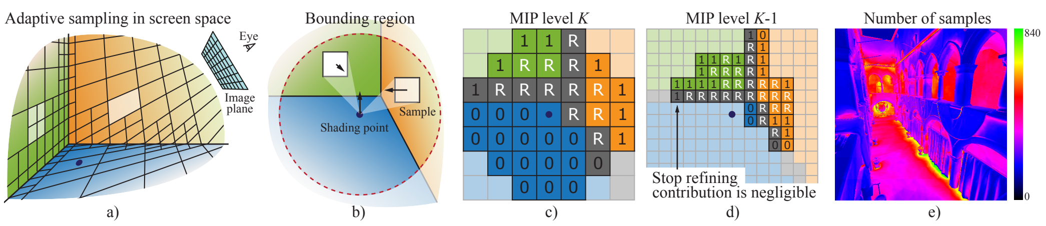


Figure 11.2: *Bias compensation is computed in screen space (a). Therefore we first determine the bounding region in which all surfaces contribute to the compensation (b) and use a hierarchical integration scheme traversing a mip-map hierarchy of the G-buffers (c,d). Samples spanning discontinuities or subtending a large projected solid angle are refined (R), until their contribution can be estimated accurately (1), or drops to zero (0). The total number of processed samples for each pixel for the Crytek Sponza scene is shown in (e). Illustration: Novák et al. [2011].*

tracing, creating VPLs at each location of particle impingement (cf. Chapter 4.5). The second pass lights the scene with the VPLs and thereby approximates indirect illumination. The full solution of light transport is then the sum of emitted light, direct illumination from primary light sources, and indirect illumination. The latter is approximated as direct illumination from all VPLs ($\hat{L}$), or in the terms of the Neumann-series

$$L = \underbrace{L_e}_{\text{emission}} + \underbrace{\mathbf{T}L_e}_{\text{direct illum.}} + \underbrace{\mathbf{T}\hat{L}}_{\text{indirect illum.}}. \tag{11.1}$$

As stated earlier, lighting with VPLs exhibits artefacts that become visible as bright splotches and high intensity peaks if VPLs are close to shading points. This is induced by the inverse of the squared distance between the shading point and the VPL in the *unbounded* geometry term $G(\mathbf{x}\longleftrightarrow\mathbf{y})$. A typically followed approach to avoid those artefacts is to introduce a *bounded* geometry term $G_b(\mathbf{x}\longleftrightarrow\mathbf{y}) = \min(b, G(\mathbf{x}\longleftrightarrow\mathbf{y}))$ and the corresponding *bounded transport operator* $\mathbf{T}_b$, that only differs in the geometry term from the *unbounded transport operator* $\mathbf{T}$.

11.1.2 Efficient Compensation using Residual Transport

In the following, we will derive a new transport operator that enables us to formulate bias compensation as a post-process in image-space. First we shall compute the *residual transport operator* as the difference between the bounded and the unbounded transport operators:

$$\mathbf{T}_r = \mathbf{T} - \mathbf{T}_b.$$

Similarly to $\mathbf{T}_b$, the residual operator $\mathbf{T}_r$ differs from $\mathbf{T}$ only in the geometry term: $G_r(\mathbf{x}\longleftrightarrow\mathbf{y}) = \max(G(\mathbf{x}\longleftrightarrow\mathbf{y}) - b, 0)$. Now we can express the unbounded transport operator as the sum of the residual and the bounded operators ($\mathbf{T} = \mathbf{T}_r + \mathbf{T}_b$). Plugging this expression into Equation 11.1 yields:

$$L = L_e + \mathbf{T}L_e + \mathbf{T}_b\hat{L} + \mathbf{T}_r\hat{L}.$$

Note that the residual operator is also prone to singularities when applied to illumination from VPLs ($\hat{L}$) directly. This is because of the definition of the residual geometry term G_r (see above). We observe that instead of using VPL illumination $\hat{L}$, we can replace it with general reflected illumination ($L - L_e$) which does not depend on the point light sources, removing the source of VPL typical artefacts. Then the full unbiased illumination becomes

$$L = L_e + \mathbf{T}L_e + \mathbf{T}_b\hat{L} + \mathbf{T}_r(L - L_e). \tag{11.2}$$

Expanding this equation recursively, yields the full unbiased rendering equation for instant radiosity:

$$L = L_e + \sum_{n=0}^{\infty} \mathbf{T}_r^n(\mathbf{T}L_e + \mathbf{T}_b\hat{L}). \tag{11.3}$$

The key of this reformulation is that direct illumination and indirect illumination from VPLs with clamping has to be computed only once for all surface points ($n = 0$), and then the residual operator can be applied recursively for an infinite number of times to obtain an unbiased solution. Therefore no re-evaluation of VPLs is required for the n-th addend as the previously computed illumination obtained by the $(n-1)$-th addend can simply be reused.

Because the bounded geometry term occurs in each addend, an unbiased solution is only obtained at infinity. This, however, is not practical and we only evaluate a finite number of iterations. Consequently an error ϵ^N is introduced. It can be computed considering only the first N iterations, and it is the sum over all omitted higher order terms:

$$\epsilon^N = |\mathbf{T}_r^{N+1}(\mathbf{T}L_e + \mathbf{T}\hat{L})|.$$

An important observation is that the energy gain due to compensation is $(N+1)$-times convolved with the BRDF, and therefore it is dropping exponentially with increasing N. This is an important observation for practical applications. Because of this rapid convergence rate, the bias decreases quickly and all *visible* bias is removed after 1 to 3 iterations. In our examples this yielded nearly indistinguishable results from our unbiased reference solutions. Based on our novel reformulation of the IR rendering equation, we can derive an algorithm with bias compensation that operates in two steps:

First, we compute global illumination using direct illumination from primary light sources ($\mathbf{T}L_e$) and bounded indirect illumination from secondary light sources ($\mathbf{T}_b\hat{L}$). Second, we correct the bias in the image applying our residual transport operator to the reflected illumination in the scene repeatedly ($\mathbf{T}_r^N(L-L_e)$) until all visible bias is removed.

This would require storing shading points over all scene surfaces, which is cumbersome. In the following section we outline how to simplify the problem using only the visible surfaces in screen space for compensation.

11.2 Screen Space Bias Compensation

A screen space algorithm has one noteworthy advantage: All information about visible surfaces is present in G-buffers [Saito and Takahashi, 1990] and no complicated data structures are required. This is also true for bias compensation and beyond that, surfaces that are close to a shading point in 3D world space, thereby contributing to the compensation, are also close in 2D screen space. This allows us to implement SSBC as a simple post-process behaving like a convolution with filter of limited support.

11.2.1 Integration in Screen Space

The transport operators, including $\mathbf{T}_r$, are actually integrals defined over surfaces, but are not analytically solvable. Hence we approximate these integrals numerically, computing a weighted sum over a finite number of pixels (we discuss how to handle the inherent limitations of screen space approaches in Section 11.2.3).

Each pixel corresponds to a world space finite element, or surface patch, P_i with position $\mathbf{z}_i$, normal $\mathbf{n}_i$, and surface area A_i stored in the G-buffer. The area is computed using screen space derivatives. Similar to radiosity we have to sum up the contribution integrated over each of the M finite elements in the image:

$$\mathbf{T}_r\hat{L} = \sum_i^M \int_{P_i} f_r(\mathbf{z}_i{\to}\mathbf{y}{\to}\mathbf{x})G_r(\mathbf{y}{\leftrightarrow}\mathbf{z}_i)V(\mathbf{y}{\leftrightarrow}\mathbf{z}_i)L(\mathbf{z}_i{\to}\mathbf{y})d\mathbf{z}_\mathrm{i}. \qquad (11.4)$$

Assuming the area A_i of the finite elements is sufficiently small, we can further assume the integrand to be constant which yields the approximation

$$\mathbf{T}_r\hat{L} \approx \sum_{i=1}^{M} f_r(\mathbf{z}_i{\rightarrow}\mathbf{y}{\rightarrow}\mathbf{x})G_r(\mathbf{y}{\longleftrightarrow}\mathbf{z}_i)V(\mathbf{y}{\longleftrightarrow}\mathbf{z}_i)L(\mathbf{z}_i{\rightarrow}\mathbf{y})A_i. \tag{11.5}$$

As previously mentioned, only surfaces nearby $\mathbf{y}$ contribute to the compensation. This follows directly from the definition of the residual geometry term $G_r(\mathbf{y}{\longleftrightarrow}\mathbf{z}_i)$, which evaluates to zero for surfaces beyond a certain distance r. Recall from chapter 4.5 that this distance in *3D world space* can be conservatively estimated as

$$r = \frac{1}{\sqrt{b}}.$$

This follows directly from the definition of the bounded geometry term, assuming the contained cosine-terms evaluate to 1. This assumption is necessary because detecting the maximum cosines for tight bounds is not feasible as it would require costly searches over all surfaces nearby. The fact that the residual geometry term is non-zero only for nearby surfaces allows us to exploit an observation made by Arikan et al. [2005]: nearby surfaces are rarely occluded and thus we omit the visibility function.

So far, we have only estimated the bounding radius in 3D world space, but for a screen-space method we require the bounding radius r_s under perspective projection with a field of view δ:

$$r_s = \frac{r}{\left(\tan(0.5\cdot\delta)\,\left\|\mathbf{x}-\mathbf{y}\right\|\right)}.$$

This implies that r_s varies with the distance between the camera at $\mathbf{x}$ and the point $\mathbf{y}$. The shorter the distance, the bigger the radius r_s, and hence more pixels contribute to the compensation. Due to this dependency it may occur that the screen-space bounding regions span several thousands of pixels (see Figure 11.3b). This high number deteriorates interactive performance and in the following we derive a hierarchical screen space integration scheme that helps to maintain efficiency.

11.2.2 Hierarchical Integration

As stated earlier, it is infeasible to evaluate thousands of pixels per compensation while simultaneously maintaining interactive frame rates. For this reason, we opt to reduce the number of arithmetic operations and memory accesses during computation and develop a hierarchical integration scheme that is inspired by hierarchical radiosity [Hanrahan et al., 1991]. Hierarchical radiosity makes use of a surface patch hierarchy, which is used to refine patches if the form factor between two patches exceeds a user defined threshold.

Similarly, we adopt this simple but powerful approach. First, we build a hierarchy on the pixels (each corresponding to a finite element) constructing a mip-map chain for the G-buffer which contains positions, normals, pixel area, material properties, and illumination computed from the bounded light transport, i.e. the colours of the pixels. Since the mip-map averages over all properties, and we want to avoid integrating over sharp edges and depth discontinuities, we additionally construct a discontinuity buffer (and a mip-map chain thereof), similarly to Nichols and Wyman [2009].

Second, we use this hierarchy for our integration scheme. Integration begins with the coarsest level in the mip hierarchy (typically a resolution of 64^2 for an image resolution of 1024^2), and we decide whether it is sufficiently accurate or if the current level must be refined further, proceeding to the next mip level. In order to avoid spatial and temporal artefacts in dynamic scenes, we refine the integration if at least one of the following criteria is met:

1. The projected solid angle of a surface corresponding to a sample exceeds a given threshold (0.08 sr in our examples) because it is too large or too close to the shading point $\mathbf{y}$.

2. The current sample spans a discontinuity. Then the position, normal, and area were averaged and do not represent the original geometry accurately.

In those cases, integration proceeds with the information on the next finer mip-level. Unfortunately, the second criterion may cause unnecessary refinement for surfaces with negligible contribution (e.g. distant surfaces). Thus we specify an additional criterion and only refine at discontinuities only if the projected solid angle is higher than a user specified threshold (0.04 sr in our examples). We have not noticed any artefacts stemming thereof, as only samples with low contribution are affected.

11.2.3 Avoiding Screen-Space Integration Artefacts

There are inherent problems that are common to all screen space approaches. On the one hand, a G-buffer stores only information about the front-most,

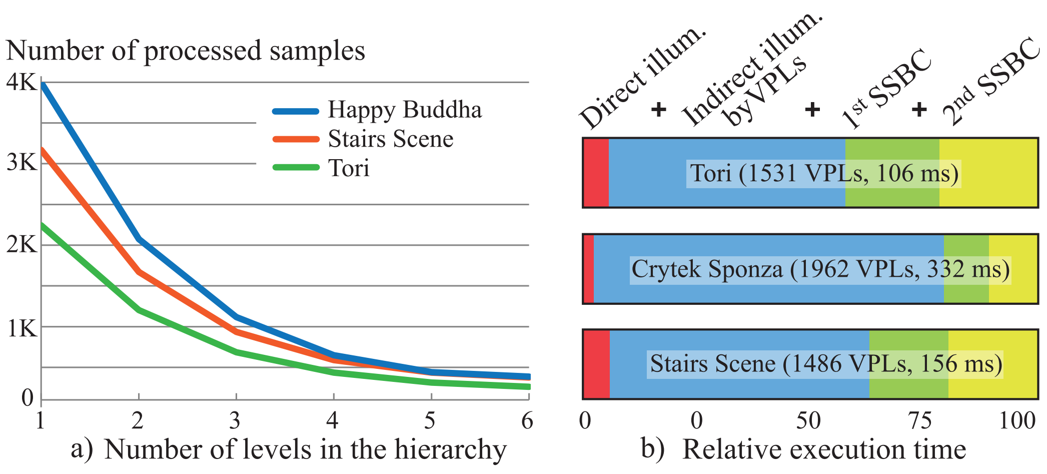


Figure 11.3: *a) The average number of samples per pixel decreases rapidly with each additional level that is used for hierarchical integration. b) The relative time spent for 2 compensation steps is only moderate compared to the evaluation of indirect illumination from all VPLs per pixel. Illustration: Novák et al. [2011].*

visible surfaces, but hidden surfaces potentially contribute to the compensation, too. They can possibly be included in the compensation using depth peeling or multi-fragment rendering. However, this would make the approach more complicated and less efficient. Other issues in our screen-space bias compensation can arise from pixels that represent surfaces which are nearly perpendicular to the viewing direction. In those cases, the area such a pixel represents is very large and the approximate residual operator (Equation 11.5) is no longer valid, and we should rather fall back to its exact variant (Equation 11.4). This, however, involves numerical integration, which is infeasible for our purposes. Thus we rely on a simple but efficient alternative. Overestimating a patch's contribution leads to brightening, which would sometimes be even more distracting than the non-clamped contribution of a nearby VPL, therefore, we decay the contribution of pixels with the angle between their normals and the viewing direction greater than 80 degrees by a quadratic falloff.

11.3 Implementation Details

We implemented our screen-space bias compensation algorithm entirely on the GPU. The only exception is VPL generation, which is executed entirely on the CPU using ray tracing in conjunction with a pre-built bounding vol-

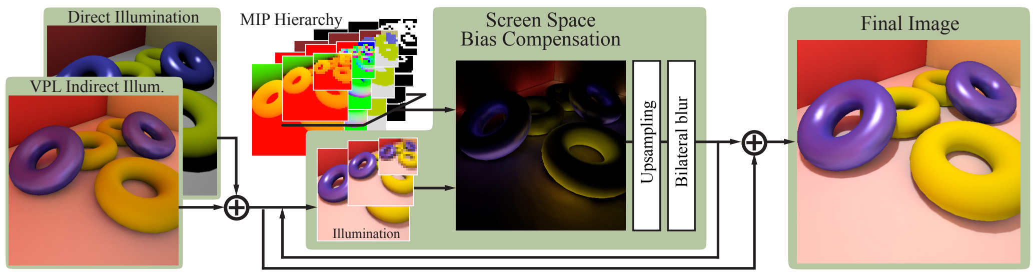


Figure 11.4: *For hierarchical integration, SSBC requires a hierarchical G-buffer and multi-resolution images of direct and indirect illumination. The integration result is upsampled and optionally blurred, and finally added to direct and clamped illumination to compute the corrected image. Illustration: Novák et al. [2011].*

ume hierarchy for static parts of the scene. For dynamic objects we use a separate BVH that is rebuild each frame. This in fact never proofed to be the bottleneck even for up to a few thousand of VPLs.

On the GPU our method first creates the G-buffers, shadow maps for primary light sources, and imperfect shadow maps [Ritschel et al., 2008] for VPLs. Then direct illumination and indirect illumination from VPLs is computed and stored in temporary textures. Before we step into bias compensation, the mip-map hierarchy and discontinuity buffers are built. Finally bias compensation is executed in a compute shader because it allows us to handle hierarchical refinement efficiently maintaining a stack placed in group-shared memory. Optionally, we apply an edge-preserving bilateral Gaussian blur to the compensation result to avoid block artefacts that can appear when using the hierarchical approach with very few samples. If we perform multiple compensation steps, we have to use the illumination buffer from the previous step and therefore must also update its mip map. These steps are also detailed in Figure 11.4.

For further performance improvements, we can exploit the fact that indirect illumination varies smoothly over surfaces and compute indirect illumination at half resolution in each image dimension. Afterwards a full resolution image is constructed by the means of bilateral upsampling [Sloan et al., 2007] to avoid blurring over sharp edges and depth discontinuities. Finally we execute our bias compensation shader once more for defective pixels, i.e. those for which all upsampling weights are (nearly) zero.

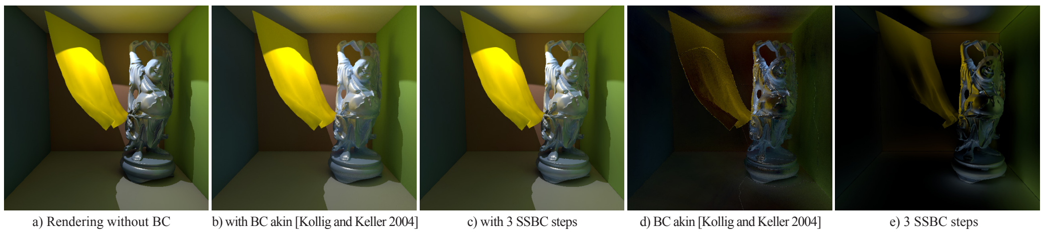

a) Rendering without BC
b) with BC akin [Kollig and Keller 2004]
c) with 3 SSBC steps
d) BC akin [Kollig and Keller 2004]
e) 3 SSBC steps

Figure 11.5: *We compare SSBC to the ground-truth offline BC solution [Kollig and Keller, 2006]. (a) Rendering without BC takes about 20 minutes for our offline renderer without any compensation and additional 10.9 hours with BC (b). A visually indistinguishable image can be obtained with 3 steps SSBC running entirely on the GPU (c). The energy recovered from both compensation methods is shown in (d) and (e), respectively. Illustration: Novák et al. [2011].*

11.4 Results

In this section we show the benefits of our screen-space bias compensation algorithm. We analyse it regarding rendering performance and quality and assess the importance of bias compensation. All renderings have been computed using an ATI Radeon HD 5870 running on an Intel Core i7 860 with 2.8 GHz with 8 GB of RAM.

Performance: Figure 11.3a demonstrates the benefit of hierarchical integration. Without hierarchical integration, the compensation requires up to four thousand samples for every pixel. When enabling hierarchical integration, the number of pixels that need to be processed decreases quickly with each additional level. Figure 11.3b reports the relative time spent on the individual parts of the pipeline. Here we always used hierarchies with 6 levels computing the compensation for 1024×768 images and performing two compensation steps (green and yellow).

Quality: In Figure 11.5 we assess the quality of SSBC. For a fair comparison we replace GPU shadow maps with ray traced shadows to rule out any potential artefacts stemming from insufficient visibility sampling. The ground-truth solution, rendered with offline bias compensation [Kollig and Keller, 2006], took about 16.5 hours to render results with an acceptable level of noise. Screen-space bias compensation with three compensation steps achieves results of comparable quality at interactive frame rates (3 SSBC steps at highest quality settings take 550 ms).

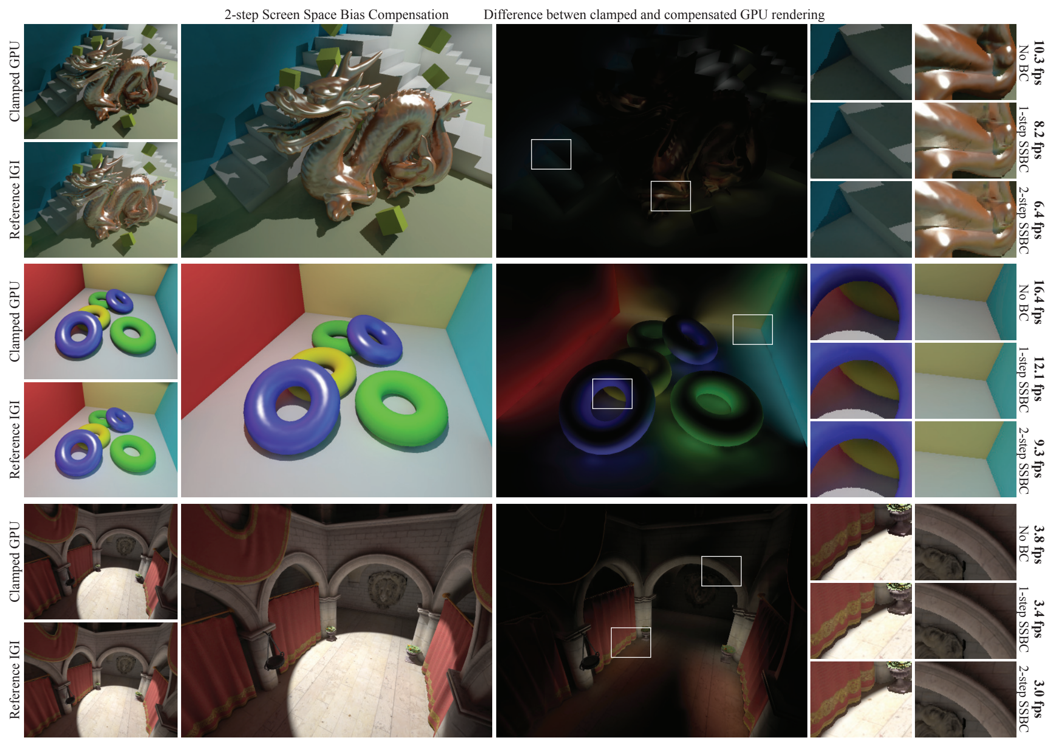


Figure 11.6: *Our test scenes for which we show clamped GPU and reference CPU solution (left column). Our technique with 2 compensation steps and full renderings are shown in the middle columns. The right columns show insets for which our compensation contributes significantly. For diffuse surfaces, a single compensation step typically suffices, whereas glossy surfaces benefit from more steps. Each compensation step takes approximately 27 ms for an image resolution of* 1024×768. *Illustration: Novák et al. [2011].*

Importance: Figure 11.6 demonstrates the importance of bias compensation. Clamping removes a significant amount of energy from short-distance transport and darkening of edges becomes clearly visible. As can be observed, recovery of lost energy is mostly sufficient after only one SSBC step for diffuse surfaces. Glossy surfaces, in contrast, greatly benefit from additional compensation steps.

11.5 Conclusions

In this chapter we presented a novel algorithm for rendering high-quality global illumination with virtual point light methods. We have shown how bias compensation can be cast into a simple post-process working entirely on a generated image, neither requiring access to all VPLs during the compensation, nor requiring any GPU-unfriendly ray tracing. Despite the assumptions and simplifications necessary in screen space, we have demonstrated, that our method computes results that are nearly indistinguishable from the ground-truth solution, but are generated orders of magnitudes faster. This makes our approach also suitable for offline renderers. The efficiency of our method is due to the fact that it can be implemented as a simple post-process that can be accelerated on the GPU, and, because we made use of an involved hierarchical integration scheme, it allows us to reduce the overall arithmetic cost.

Beyond that, we have discussed strategies to avoid artefacts that are stemming from the screen-space approach. For future work, we opt to reveal hidden surfaces in screen space using depth peeling or multi-fragment rendering.

CHAPTER 12

Conclusions

In this thesis we have presented several contributions that target specific light transport phenomena such as one-bounce indirect illumination or single scattering. Our algorithms are highly efficient to compute, which is because of two major aspects: (1) efficient computation and exploitation of visibility and (2) being amenable for highly parallel architectures, such as GPUs. Within these limits we heavily focused on many light or VPL-based methods as the foundation of our contributions. One of the key advantages is that they map exceptionally well onto GPUs. Illumination from point lights is trivially computed for all pixels in parallel using shadow maps in order to resolve visibility. Further, they support a wide variety of illumination effects, such as soft shadows from area light sources, diffuse and moderately glossy indirect illumination, colour bleeding, and even multiple scattering.

(Hemi-)spherical visibility: With VPL methods in mind we investigated a parametrization for (hemi-)spherical visibility from which we derived our octahedron environment maps (Chapter 7). These are applicable in many different scenarios, e.g. image-based lighting. But besides storing the light field at an arbitrary (surface) point, they are also a viable means for storing omni-directional shadow maps for (virtual) point lights. We have demonstrated two projection schemes that guarantee optimal usage of texture space, i.e. no space in the omni-directional shadow map is wasted. Although direct rendering of octahedron environment maps using rasterization proved to be challenging, they can be created efficiently using point-based algorithms [Ritschel et al., 2008].

The efficient computation of hemispherical visibility is also key to our micro-rendering algorithm (Chapter 8) for high quality final gathering at interactive framerates. Although this method has been primarily designed to compute only one-bounce indirect illumination, we have shown that successive applications of the method make it possible to compute an arbitrary number of light bounces. Beyond that, we have shown that various global illumination algorithms, such as photon mapping [Jensen, 1996] or instant radiosity [Keller, 1997], benefit from our approach. Final gathering diminishes the typical artefacts these algorithms suffer from to a point where low frequency noise from photon mapping or bright splotches from virtual point lights become invisible.

Approximate visibility: Artefacts from VPLs arise due to the inverse square distance in the illumination term at nearby surfaces. A more efficient way than final gathering to avoid these artefacts completely is to bound their contribution artificially, which on the other hand introduces bias. Fortunately, this bias can be computed but requires a procedure that is akin to final gathering [Kollig and Keller, 2006]. Although micro-rendering could be used for this purpose as well, it still depends on a hierarchical data structure of the entire scene which limits its complexity and requires sophisticated treatment of dynamic objects.

A key insight in bias compensation is that only light reflected from surfaces in the close proximity of a shading point contributes to the compensation term. We exploited this fact and developed a screen-space bias compensation algorithm (Chapter 11). This method reconstructs an approximation of the visible surfaces at a shading point solely from the image the point is contained in. Thereby our screen-space method does not depend on sophisticated hierarchical data structures, is output sensitive, and supports fully dynamic scenes at the same time. Despite the fact that the accuracy of this reconstruction is limited by the amount of information available in screen-space, we have demonstrated that we can render fully dynamic scenes at interactive speed with full indirect illumination from thousands of VPLs.

VPLs can not only be used to simulate indirect illumination. In the presence of participating media, multiple scattering becomes possible accumulating the single scattering contribution of each VPL. This approach suffers from the same VPL-typical artefacts as surface illumination. As before, it can be remedied bounding the contribution and recovering the short distance light transport through bias compensation at each volume shading point [Raab et al., 2008]. There it becomes exceedingly more expensive because visibility is no longer binary and evaluated hundred times over compared to the surface-only case. In Chapter 10 we have introduced several approximations and improvements that require less visibility evaluations and employ appropriate approximations to make bias compensation more efficient without introducing noticeable errors in the final image. Additionally, our method is easily parallelized on GPUs and generates high quality images of multiple scattering effects in a matter of seconds.

Visibility coherence: These aforementioned approximations only benefit the compensation step, but single scattering from (virtual) point light sources is also costly because in-scattering is computed and integrated at multiple samples along each eye ray. In Chapter 9 we have demonstrated how this process can be accelerated exploiting visibility coherence along light rays in order to derive an optimised subsampling algorithm. Our method generates epipolar lines in screen space and exploits the fact that all eye rays intersecting a line share the same visibility with respect to the light source. Consequently in-scattered light observed along an epipolar line varies only slowly and which allowed us to replace most expensive operations by screen-space interpolation.

Culling: Finally, we have not only investigated visibility with respect to light sources or the environment but also presented an algorithm that accelerates the image generation process with respect to the camera. This becomes important for algorithms that heavily rely on screen-space representations of the scene, e.g. screen-space bias compensation (Chapter 11). Our granular occlusion queries (Chapter 6) make it possible to cull invisible geometry from the rendering pipeline of arbitrary granularity within a single draw call. We have demonstrated that rendering techniques that heavily rely on geometry amplification in the geometry shader particularly benefit from our approach.

Summary: In summary we have presented several new algorithms, that tackle the visibility problem in scenes with participating media from different angles. Our studies focused on increasing the efficiency of virtual point light methods for highly parallel hardware architectures as found on GPUs. In the process we have addressed visibility for (1) image generation (Chapter 6), (2) efficient storage and computation of (hemi-)spherical visibility for indirect illumination (Chapters 7 and 8), (3) exploiting visibility coherence for improved subsampling of single scattering from (virtual) point lights (Chapter 9), and (4) efficient visibility approximations for fast bias compensation on GPUs for multiple scattering (Chapter 10) and indirect illumination (Chapter 11).

Bibliography

Timo Aila and Samuli Laine. Understanding the Efficiency of Ray Traversal on GPUs. In *Proceedings of the Conference on High Performance Graphics*, HPG '09, pages 145–149, 2009.

Tomas Akenine-Möller, Eric Haines, and Natty Hoffman. *Real-Time Rendering 3rd Edition*. A. K. Peters, Ltd., Natick, MA, USA, 2008. ISBN 987-1-56881-424-7.

Arthur Appel. Some Techniques for Shading Machine Renderings of Solids. In *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, AFIPS '68 (Spring), pages 37–45, 1968.

Okan Arikan, David A. Forsyth, and James F. O'Brien. Fast and Detailed Approximate Global Illumination by Irradiance Decomposition. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2005)*, 24(3):1108–1114, July 2005.

James Arvo and David Kirk. Particle Transport and Image Synthesis. *Computer Graphics (Proceedings of SIGGRAPH 1990)*, 24(4):63–66, September 1990.

Ulf Assarsson, Michael Dougherty, Michael Mounier, and Tomas Akenine-Möller. An Optimized Soft Shadow Volume Algorithm with Real-Time Performance. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, HWWS '03, pages 33–40, 2003.

Ilya Baran, Jiawen Chen, Jonathan Ragan-Kelley, Frédo Durand, and Jaakko Lehtinen. A Hierarchical Volumetric Shadow Algorithm for Single Scattering. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia 2010)*, 29(6):178:1–178:10, December 2010.

Louis Bavoil, Miguel Sainz, and Rouslan Dimitrov. Image-space horizon-based ambient occlusion. In *SIGGRAPH '08: ACM SIGGRAPH 2008 talks*, 2008.

Neeta Bhate and Tokuta A. Photorealistic Volume Rendering of Media with Directional Scattering. In *Proceedings of the Eurographics Workshop on Rendering*, EGWR '92, pages 227–245, 1992.

Jiří Bittner, Michael Wimmer, Harald Piringer, and Werner Purgathofer. Coherent Hierarchical Culling: Hardware Occlusion Queries Made Useful. *Computer Graphics Forum (Proceedings of Eurographics 2004)*, 23(3):615–624, September 2004.

Jiří Bittner and Peter Wonka. Visibility in Computer Graphics. *Environment and Planning B: Planning and Design*, 30(5):729–755, 2003.

James F. Blinn. Light Reflection Functions for Simulation of Clouds and Dusty Surfaces. *Computer Graphics (Proceedings of SIGGRAPH 1982)*, 16 (3):21–29, July 1982.

James F. Blinn and Martin E. Newell. Texture and Reflection in Computer Generated Images. *Communications of the ACM*, 19(10):542–547, October 1976.

Edwin Earl Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. PhD thesis, The University of Utah, 1974. AAI7504786.

Eva Cerezo, Frederic Perez-Cazorla, Xavier Pueyo, Francisco Seron, and François X. Sillion. A Survey on Participating Media Rendering Techniques. *The Visual Computer*, 2005.

Subrahmanyan Chandrasekhar. *Radiative Transfer*. Dover Publications Inc., 1960. ISBN 978-0486605906.

Jiawen Chen, Ilya Baran, Frédo Durand, and Wojciech Jarosz. Real-Time Volumetric Shadows using 1D Min-Max Mipmaps. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, I3D '11, pages 39–46, 2011.

Per H. Christensen. Point-Based Approximate Color Bleeding. Technical Report 08-01, Pixar, Emeryville, California, July 2008.

Michael F. Cohen and Donald P. Greenberg. The Hemi-Cube: A Radiosity Solution for Complex Environments. *Computer Graphics (Proceedings of SIGGRAPH 1985)*, 19(3):31–40, July 1985.

Michael F. Cohen, Shenchang Eric Chen, John R. Wallace, and Donald P. Greenberg. A Progressive Refinement Approach to Fast Radiosity Image Generation. *Computer Graphics (Proceedings of SIGGRAPH 1988)*, 22(4): 75–84, June 1988.

Michael F. Cohen, John Wallace, and Pat Hanrahan. *Radiosity and Realistic Image Synthesis*. Academic Press Professional, Inc., San Diego, CA, USA, 1993. ISBN 978-0121782702.

Daniel Cohen-Or, Yiorgos L. Chrysanthou, Cláudio T. Silva, and Frédo Durand. A Survey of Visibility for Walkthrough Applications. *IEEE Transactions on Visualization and Computer Graphics*, 9(3):412–431, July 2003.

W. Coleman. Mathematical Verification of a Certain Monte Carlo Sampling Technique and Applications of the Technique to Radiation Transport Problems. *Nuclear Science and Engineering*, 32:76–81, 1968.

Robert. L. Cook and Kenneth. E. Torrance. A Reflectance Model for Computer Graphics. *ACM Transactions on Graphics*, 1(1):7–24, January 1982.

Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed Ray Tracing. *Computer Graphics (Proceedings of SIGGRAPH 1984)*, 18(3):137–145, January 1984.

Greg Coombe, Mark J. Harris, and Anselmo Lastra. Radiosity on Graphics Hardware. In *Proceedings of Graphics Interface 2004*, GI '04, pages 161–168, 2004.

Franklin C. Crow. Shadow Algorithms for Computer Graphics. *Computer Graphics (Proceedings of SIGGRAPH 1977)*, 11(2):242–248, July 1977.

Carsten Dachsbacher and Marc Stamminger. Translucent Shadow Maps. In *Rendering Techniques 2003 (Proceedings of the Eurographics Symposium on Rendering)*, EGSR '03, pages 197–201, 2003.

Carsten Dachsbacher and Marc Stamminger. Reflective Shadow Maps. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, I3D '05, pages 203–231, 2005.

Carsten Dachsbacher and Marc Stamminger. Splatting Indirect Illumination. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, I3D '06, pages 93–100, 2006.

Carsten Dachsbacher, Christian Vogelgsang, and Marc Stamminger. Sequential Point Trees. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2003)*, 22(3):657–662, July 2003.

Carsten Dachsbacher, Marc Stamminger, George Drettakis, and Frédo Durand. Implicit Visibility and Antiradiance for Interactive Global Illumination. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007)*, 26 (3), July 2007.

Tomáš Davidovič, Jaroslav Křivánek, Miloš Hašan, Philipp Slusallek, and Kavita Bala. Combining Global and Local Virtual Lights for Detailed Glossy Illumination. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia 2010)*, 29(6):143:1–143:8, December 2010.

Tomáš Davidovič, Thomas Engelhardt, Iliyan Georgiev, Philipp Slusallek, and Carsten Dachsbacher. 3D Rasterization: A Bridge between Rasterization and Ray Casting. *Proceedings of Graphics Interface 2012*, pages 201 – 208, 2012.

Paul Debevec. Rendering Synthetic Objects into Real Scenes: Bridging Traditional and Image-Based Graphics with Global Illumination and High Dynamic Range Photography. In *Proceedings of SIGGRAPH 1998*, Annual Conference Series, pages 189–198, 1998.

Paul E. Debevec and Jitendra Malik. Recovering High Dynamic Range Radiance Maps from Photographs. In *Proceedings of SIGGRAPH 1997*, Annual Conference Series, pages 369–378, 1997.

Xavier Décoret. N-Buffers for Efficient Depth Map Query. *Computer Graphics Forum*, 24(3), 2005.

Yoshinori Dobashi, Tsuyoshi Yamamoto, and Tomoyuki Nishita. Interactive Rendering of Atmospheric Scattering Effects using Graphics Hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, HWWS ’02, pages 99–107, 2002.

George Drettakis and François X. Sillion. Interactive Update of Global Illumination Using a Line-Space Hierarchy. In *Proceedings of SIGGRAPH 1997*, Annual Conference Series, pages 57–64, 1997.

George Drettakis, Nicolas Bonneel, Carsten Dachsbacher, Sylvain Lefebvre, Michael Schwarz, and Isabelle Viaud-Delmon. An Interactive Perceptual Rendering Pipeline using Contrast and Spatial Masking. In *Rendering*

Techniques 2007 (Proceedings of the Eurographics Symposium on Rendering), EGSR '07, June 2007.

Bryan Dudash. Skinned Instancing. NVIDIA Direct3D SDK 10 Samples Whitepaper, February 2007.

Philip Dutré, Kavita Bala, and Philippe Bekaert. *Advanced Global Illumination*. A K Peters, Ltd., 888 Worcester Stree, Suite 230, Wellesley, MA 02482, 2006. ISBN 978-1568813073.

Elmar Eisemann and Xavier Décoret. Fast Scene Voxelization and Applications. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, I3D '06, pages 71–78, 2006.

Elmar Eisemann, Michael Schwarz, Ulf Assarsson, and Michael Wimmer. *Real-Time Shadows*. A K Peters/CRC Press, Boca Raton, FL, USA, 2011. ISBN 978-1-56881-438-4.

Thomas Engelhardt and Carsten Dachsbacher. Octahedron Environment Maps. In *Proceedings of the Vision, Modeling, and Visualization Workshop*, VMV '08, October 2008.

Thomas Engelhardt and Carsten Dachsbacher. Granular Visibility Queries on the GPU. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, I3D '09, pages 161–167, 2009.

Thomas Engelhardt and Carsten Dachsbacher. Epipolar Sampling for Shadows and Crepuscular Rays in Participating Media with Single Scattering. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, I3D '10, pages 119–125, 2010.

Thomas Engelhardt, Jan Novák, Thorsten-Walter Schmidt, and Dachsbacher Carsten. Approximate Bias Compensation for Rendering Scenes with Heterogeneous Participating Media. *Computer Graphics Forum (Proceedings of Pacific Graphics 2012)*, 31(7), September 2012.

Manfred Ernst, Frank Firsching, and Roberto Grosso. Entkerner: A System for Removal of Globally Invisible Triangles from Large Meshes. In *IMR*, pages 449–458, 2004.

Raanan Fattal. Participating Media Illumination using Light Propagation Maps. *ACM Transactions on Graphics*, 28(1):7:1–7:11, February 2009.

James D. Foley, Andries Van Dam, and Steven K. Feiner. *Computer Graphics: Principles and Practice in C*. Addison-Wesley Longman Publishing Co. Inc, 1990.

Akira. Fujimoto, Takayuki Tanaka, and Kansei Iwata. ARTS: Accelerated Ray-Tracing System. *IEEE Computer Graphics and Applications*, 6(4):16 –26, April 1986.

Thomas A. Funkhouser, Seth J. Teller, and Delnaz Khorramabadi. The UC Berkeley System for Interactive Visualization of Large Architectural Models. *Presence*, 5(1):13–44, 1996.

Pascal Gautron, Jaroslav Křivánek, Kadi Bouatouch, and Sumanta Pattanaik. Radiance Cache Splatting: A GPU-Friendly Global Illumination Algorithm. In *Rendering Techniques 2005 (Proceedings of the Eurographics Symposium on Rendering)*, EGSR ’05, pages 55–64, 2005.

Pascal Gautron, Jean-Eudes Marvie, and Guillaume François. Volumetric shadow mapping. In *ACM SIGGRAPH 2009: Sketches*, SIGGRAPH ’09, pages 49:1–49:1, 2009.

Iliyan Georgiev and Philipp Slusallek. Simple and Robust Iterative Importance Sampling of Virtual Point Lights. In *Eurographics 2010 Short Papers*, pages 57–60, 2010.

Andrew S. Glassner. Space Subdivision for Fast Ray Tracing. *IEEE Computer Graphics and Applications*, 4(10):15–22, October 1984.

Jeffrey Goldsmith and John Salmon. Automatic Creation of Object Hierarchies for Ray Tracing. *IEEE Computer Graphics and Applications*, 7(5):14 – 20, May 1987.

Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, and Bennett Battaile. Modeling the Interaction of Light Between Diffuse Surfaces. *Computer Graphics (Proceedings of SIGGRAPH 1984)*, 18(3):213–222, January 1984.

Steven J. Gortler, Peter Schröder, Michael F. Cohen, and Pat Hanrahan. Wavelet Radiosity. In *Proceedings of SIGGRAPH 1993*, Annual Conference Series, pages 221–230, 1993.

Donald P. Greenberg, James Arvo, Eric Lafortune, Kenneth E. Torrance, James A. Ferwerda, Bruce Walter, Ben Trumbore, Peter Shirley, Sumanta

Pattanaik, and Sing-Choong Foo. A Framework for Realistic Image Synthesis. In *Proceedings of SIGGRAPH 1997*, Annual Conference Series, pages 44–53, 1997.

Ned Greene. Environment Mapping and Other Applications of World Projections. *IEEE Computer Graphics and Applications*, 6(11):21–29, November 1986.

Ned Greene. Occlusion Culling with Optimized Hierarchical Buffering. In *ACM SIGGRAPH 99 Conference Abstracts and Applications*, SIGGRAPH '99, pages 261–, 1999.

Ned Greene, Michael Kass, and Gavin Miller. Hierarchical Z-Buffer Visibility. In *Proceedings of SIGGRAPH 1993*, Annual Conference Series, pages 231–238, 1993.

Gene Greger, Peter Shirley, Philip M. Hubbard, and Donald P. Greenberg. The Irradiance Volume. *IEEE Computer Graphics and Applications*, 18(2): 32–43, March 1998.

Michael Guthe, Ákos Balázs, and Reinhard Klein. Near Optimal Hierarchical Culling: Performance Driven Use of Hardware Occlusion Queries. In *Rendering Techniques 2006 (Proceedings of the Eurographics Symposium on Rendering)*, EGSR '06, June 2006.

Toshiya Hachisuka, Shinji Ogaki, and Henrik Wann Jensen. Progressive Photon Mapping. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia 2008)*, 27(5):130:1–130:8, December 2008.

Pat Hanrahan, David Salzman, and Larry Aupperle. A Rapid Hierarchical Radiosity Algorithm. *Computer Graphics (Proceedings of SIGGRAPH 1991)*, 25(4):197–206, July 1991.

Richard I. Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition, 2004. ISBN 0521540518.

Vlastimil Havran and Jiři Bittner. On Improving KD-Trees for Ray Shooting. *Journal of WSCG, University of West Bohemia*, 10(1):209–217, February 2002.

Miloš Hašan, Fabio Pellacini, and Kavita Bala. Matrix Row-Column Sampling for the Many-Light Problem. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007)*, 26(3), July 2007.

Miloš Hašan, Jaroslav Křivánek, Bruce Walter, and Kavita Bala. Virtual Spherical Lights for Many-Light Rendering of Glossy Scenes. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia 2009)*, 28(5):143:1–143:6, December 2009.

Paul S. Heckbert. Fundamentals of Texture Mapping and Image Warping. Technical report, University of California at Berkeley, Berkeley, CA, USA, 1989.

Paul S. Heckbert. Adaptive Radiosity Textures for Bidirectional Ray Tracing. *Computer Graphics (Proceedings of SIGGRAPH 1990)*, 24(4):145–154, September 1990.

Paul S. Heckbert. Discontinuity Meshing for Radiosity. In *Proceedings of the Eurographics Workshop on Rendering*, EGWR '92, pages 203–226, May 1992.

Hans-Christian Hege, Tobias Höllerer, and Detlev Stalling. Volume Rendering - Mathematical Models and Algorithmic Aspects. Technical Report TR 93-7, Konrad-Zuse-Zentrum Berlin, June 1993.

Wolfgang Heidrich and Hans-Peter Seidel. View-Independent Environment Maps. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, HWWS '98, New York, NY, USA, 1998.

Justin Hensley, Thorsten Scheuermann, Greg Coombe, Montek Singh, and Anselmo Lastra. Fast Summed-Area Table Generation and its Applications. *Computer Graphics Forum (Proceedings of Eurographics 2005)*, 24 (3):547–555, 2005.

Naty Hoffman and Arcot J. Preetham. Real-Time Light-Atmosphere Interactions for Outdoor Scenes. In *Graphics Programming Methods*, pages 337–352. Charles River Media, Inc., Rockland, MA, USA, 2003. ISBN 1-58450-299-1.

Matthias Holländer, Tobias Ritschel, Elmar Eisemann, and Tamy Boubekeur. ManyLoDs: Parallel Many-View Level-of-Detail Selection for Real-Time Global Illumination. *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering 2011)*, 30(4):1233–1240, 2011.

Daniel Reiter Horn, Jeremy Sugerman, Mike Houston, and Pat Hanrahan. Interactive k-D Tree GPU Raytracing. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, I3D '07, pages 167–174, 2007.

Warren Hunt and William R. Mark. Ray-Specialized Acceleration Structures for Ray Tracing. In *Proceedings of the IEEE/EUROGRAPHICS Symposium on Interactive Ray Tracing 2008*, pages 3–10, August 2008.

Takashi Imagire, Henry Johan, Naoki Tamura, and Tomoyuki Nishita. Anti-Aliased and Real-Time Rendering of Scenes with Light Scattering Effects. *The Visual Computer*, 23(9):935–944, 2007.

David S. Immel, Michael F. Cohen, and Donald P. Greenberg. A Radiosity Method for Non-Diffuse Environments. *Computer Graphics (Proceedings of SIGGRAPH 1986)*, 20(4):133–142, August 1986.

Kei Iwaski, Yoshinori Dobashi, Fujiichi Yoshimoto, and Tomoyuki Nishita. Precomputed Radiance Transfer for Dynamic Scenes Taking into Account Light Interreflection. In Jan Kautz and Sumanta Pattanaik, editors, *Rendering Techniques 2007 (Proceedings of the Eurographics Symposium on Rendering)*, EGSR '07, pages 35–44, 2007.

Robert James. True Volumetric Shadows. In *Graphics Programming Methods*, pages 353–366. Charles River Media, Inc., Rockland, MA, USA, 2003. ISBN 1-58450-299-1.

Wojciech Jarosz, Matthias Zwicker, and Henrik Wann Jensen. The Beam Radiance Estimate for Volumetric Photon Mapping. *Computer Graphics Forum (Proceedings of Eurographics 2008)*, 27(2):557–566, April 2008.

Wojciech Jarosz, Derek Nowrouzezahrai, Robert Thomas, Peter-Pike Sloan, and Matthias Zwicker. Progressive Photon Beams. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia 2011)*, 30(6), December 2011.

Henrik Wann Jensen. Importance Driven Path Tracing using the Photon Map. In *Rendering Techniques 1995 (Proceedings of the Eurographics Workshop on Rendering)*, EGWR '95, pages 326–335, 1995.

Henrik Wann Jensen. *The Photon Map in Global Illumination*. PhD thesis, Technical University of Denmark, September 1996.

Henrik Wann Jensen and Per H. Christensen. Efficient Simulation of Light Transport in Scences with Participating Media using Photon Maps. In *Proceedings of SIGGRAPH 1998*, Annual Conference Series, pages 311–320, 1998.

David Jevans and Brian Wyvill. Adaptive Voxel Subdivision for Ray Tracing. In *Graphics Interface '89*, pages 164–172, 1989.

James T. Kajiya. The Rendering Equation. *Computer Graphics (Proceedings of SIGGRAPH 1986)*, 20(4):143–150, August 1986.

James T. Kajiya and Brian P. Von Herzen. Ray Tracing Volume Densities. *Computer Graphics (Proceedings of SIGGRAPH 1984)*, 18(3):165–174, January 1984.

Michael R. Kaplan. The Uses of Spatial Coherence in Ray Tracing. SIGGRAPH '85 Course Notes 11, 1985.

Alexander Keller. Instant Radiosity. In *Proceedings of SIGGRAPH 1997*, Annual Conference Series, pages 49–56, 1997.

Alexander Keller and Wolfgang Heidrich. Interleaved Sampling. In *Rendering Techniques 2001 (Proceedings of the Eurographics Workshop on Rendering)*, EGWR '01, June 2001.

Claude Knaus and Matthias Zwicker. Progressive Photon Mapping: A Probabilistic Approach. *ACM Transactions on Graphics*, 30(3):25:1–25:13, May 2011.

Thomas Kollig and Alexander Keller. Illumination in the Presence of Weak Singularities. In Harald Niederreiter and Denis Talay, editors, *Monte Carlo and Quasi-Monte Carlo Methods 2004*, pages 245–257. Springer Berlin Heidelberg, 2006. ISBN 978-3-540-31186-7.

Christopher Kulla and Marcos Fajardo. Importance Sampling of Area Lights in Participating Media. In *ACM SIGGRAPH 2011 Talks*, SIGGRAPH '11, pages 55:1–55:1, 2011.

Jaroslav Křivánek, Pascal Gautron, Sumanta Pattanaik, and Kadi Bouatouch. Radiance Caching for Efficient Global Illumination Computation. *IEEE Transactions on Visualization and Computer Graphics*, 11(5):550 – 561, September 2005.

Jaroslav Křivánek, James A. Ferwerda, and Kavita Bala. Effects of Global Illumination Approximations on Material Appearance. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2010)*, 29(4):112:1–112:10, July 2010.

Eric P. F. Lafortune. *Mathematical Models and Monte Carlo Algorithms for Physically Based Rendering*. PhD thesis, Cornell University, February 1996.

Eric P. F. Lafortune and Yves D. Willems. Rendering Participating Media with Bidirectional Path Tracing. In *Rendering Techniques 1996 (Proceedings of the Eurographics Workshop on Rendering)*, EGWR '96, pages 91–100, 1996.

Eric P. F. Lafortune, Sing-Choong Foo, Kenneth E. Torrance, and Donald P. Greenberg. Non-Linear Approximation of Reflectance Functions. In *Proceedings of SIGGRAPH 1997*, Annual Conference Series, pages 117–126, 1997.

Samuli Laine, Hannu Saransaari, Janne Kontkanen, Jaakko Lehtinen, and Timo Aila. Incremental Instant Radiosity for Real-Time Indirect Illumination. In *Rendering Techniques 2007 (Proceedings of the Eurographics Symposium on Rendering)*, EGSR '07, pages 277–286, 2007.

Eric Languénou, Kadi Bouatouch, and Michael Chelle. Global Illumination in the Presence of Participating Media with General Properties. In *Photorealistic Rendering Techniques (Proceedings of the Eurographics Workshop on Rendering)*, EGWR '94, pages 69–85, 1994.

Christian Lauterbach, Michael Garland, Shubhabrata Sengupta, David Luebke, and Dinesh Manocha. Fast BVH Construction on GPUs. *Computer Graphics Forum (Proceedings of Eurographics 2009)*, 28(2):375–384, 2009.

Jason Lawrence, Szymon Rusinkiewicz, and Ravi Ramamoorthi. Efficient BRDF Importance Sampling using a Factored Representation. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2004)*, 23(3):496–505, August 2004.

Pascal Lecocq, Andras Kemeny, Sylvain Michelin, and Didier Arquès. Mathematical Approximation for Real-Time Lighting Rendering Through Participating Media. In *Proceedings of the 8th Pacific Conference on Computer Graphics and Applications*, PG '00, pages 400–401, 2000.

Jaakko Lehtinen, Matthias Zwicker, Emmanuel Turquin, Janne Kontkanen, Frédo Durand, François X. Sillion, and Timo Aila. A Meshless Hierarchical Representation for Light Transport. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2008)*, 27(3):37:1–37:9, August 2008.

Tommer Leyvand, Olga Sorkine, and Daniel Cohen-Or. Ray Space Factorization for From-Region Visibility. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2003)*, 22(3):595–604, July 2003.

Dani Lischinski, Filippo Tampieri, and Donald P. Greenberg. Combining Hierarchical Radiosity and Discontinuity Meshing. In *Proceedings of SIGGRAPH 1993*, Annual Conference Series, pages 199–208, 1993.

Brandon Lloyd, Jeremy Wendt, Naga Govindaraju, and Dinesh Manocha. CC shadow volumes. In *ACM SIGGRAPH 2004 Sketches*, SIGGRAPH '04, pages 146–, 2004.

Bradford James Loos and Peter-Pike Sloan. Volumetric obscurance. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, I3D '10, pages 151–156, 2010.

David Luebke and Chris Georges. Portals and Mirrors: Simple, Fast Evaluation of Potentially Visible Sets. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, I3D '95, pages 105–ff., 1995.

David J. MacDonald and Kellogg S. Booth. Heuristics for ray tracing using space subdivision. *The Visual Computer*, 6(3):153–166, May 1990.

Oliver Mattausch, Jiří Bittner, and Michael Wimmer. CHC++: Coherent Hierarchical Culling Revisited. *Computer Graphics Forum (Proceedings of Eurographics 2008)*, 27(2):221–230, 2008.

Nelson Max. Atmospheric Illumination and Shadows. *Computer Graphics (Proceedings of SIGGRAPH 1986)*, 20(4):117–124, August 1986.

Rodomír Mech. Hardware-Accelerated Real-Time Rendering of Gaseous Phenomena. *Journal of Graphics Tools*, 6(3):1–16, September 2002.

Quirin Meyer, Christian Eisenacher, Marc Stamminger, and Carsten Dachsbacher. Data-Parallel, Hierarchical Link Creation. In *Proceedings of the Eurographics Symposium on Parallel Graphics and Visualization*, pages 65–70, 2009.

Microsoft Corporation. Windows DirectX Graphics Documentation, June 2010.

Gene S. Miller and C. Robert Hoffman. Illumination and Reflection Maps: Simulated Objects in Simulated and Real Environments. In *SIGGRAPH Course Notes for Advanced Computer Graphics Animation*, 1984.

Jason Mitchell. Light Shafts: Rendering Shadows in Participating Media. Game Developers Conference 2004 Presentations, 2004.

Kenny Mitchell. Volumetric Light Scattering as a Post-Process. In *GPU Gems 3*, pages 275–285. Addison-Wesley, 2007. ISBN 978-0321515261.

Martin Mittring. Finding next gen: CryEngine 2. In *ACM SIGGRAPH 2007 Courses*, SIGGRAPH '07, pages 97–121, 2007.

Steven S. Muchnick. *Advanced Compiler Design and Implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997. ISBN 1-55860-320-4.

Greg Nichols and Chris Wyman. Multiresolution Splatting for Indirect Illumination. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, I3D '09, pages 83–90, 2009.

Greg Nichols, Jeremy Shopf, and Chris Wyman. Hierarchical Image-Space Radiosity for Interactive Global Illumination. *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering 2009)*, 28(4): 1141–1149, 2009.

F. E. Nicodemus, J. C. Richmond, J. J. Hsia, I. W. Ginsberg, and T. Limperis. Geometric Considerations and Nomenclature for Reflectance. In *Monograph #161. National Bureau of Standards*, 1977.

Tomoyuki Nishita, Yasuhiro Miyawaki, and Eihachiro Nakamae. A Shading Model for Atmospheric Scattering Considering Luminous Intensity Distribution of Light Sources. *Computer Graphics (Proceedings of SIGGRAPH 1987)*, 21(4):303–310, August 1987.

Jan Novák, Vlastimil Havran, and Carsten Dachsbacher. Path regeneration for interactive path tracing. In *Short Papers (Eurographics 2010)*, pages 61–64, 2010.

Jan Novák, Thomas Engelhardt, and Carsten Dachsbacher. Screen-Space Bias Compensation for Interactive High-Quality Global Illumination with Virtual Point Lights. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, I3D '11, pages 119–124, 2011.

NVIDIA Corporation. NVIDIA CUDA C Programming Guide, November 2011.

Marc Olano and Trey Greer. Triangle Scan Conversion Using 2D Homogeneous Coordinates. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, HWWS '97, pages 89–95, 1997.

Steven G. Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, and Martin Stich. OptiX: A General Purpose Ray Tracing Engine. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2010)*, 29(4):66:1–66:13, July 2010.

Mark Pauly, Thomas Kollig, and Alexander Keller. Metropolis Light Transport for Participating Media. In *Rendering Techniques 2000 (Proceedings of the Eurographics Workshop on Rendering)*, EGWR '00, pages 11–22, 2000.

Vincent Pegoraro. *Efficient Physically-Based Simulation of Light Transport in Participating Media*. PhD thesis, School of Computing, University of Utah, 2009.

Vincent Pegoraro, Mathias Schott, and Steven G. Parker. A Closed-Form Solution to Single Scattering for General Phase Functions and Light Distributions. *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering 2010)*, 29(4):1365–1374, 2010.

Ken Perlin and Eric M. Hoffert. Hypertexture. *Computer Graphics (Proceedings of SIGGRAPH 1989)*, 23(3):253–262, July 1989.

Matt Pharr and Greg Humphreys. *Physically Based Rendering*. Morgan Kaufmann Publishers Inc., 30 Corporate Drive, Suite 400, Burlington, MA 01803, USA, 2010. ISBN 978-0-12-375079-2.

Bui Tuong Phong. Illumination for Computer Generated Pictures. *Communications of the ACM*, 18(6):311–317, June 1975.

Juan Pineda. A Parallel Algorithm for Polygon Rasterization. *Computer Graphics (Proceedings of SIGGRAPH 1988)*, 22(4):17–20, June 1988.

Stefan Popov, Johannes Günther, Hans-Peter Seidel, and Philipp Slusallek. Stackless KD-Tree Traversal for High Performance GPU Ray Tracing. *Computer Graphics Forum (Proceedings of Eurographics 2007)*, 26(3):415–424, September 2007.

Emil Praun and Hugues Hoppe. Spherical Parametrization and Remeshing. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2003)*, 22(3): 340–349, July 2003.

Simon Premoze, Michael Ashikhmin, Jerry Tessendorf, Ravi Ramamoorthi, and Shree Nayar. Practical Rendering of Multiple Scattering Effects in Participating Media. In Alexander Keller and Henrik Wann Jensen, editors, *Rendering Techniques 2004 (Proceedings of the Eurographics Symposium on Rendering)*, EGSR '04, pages 363–374, 2004.

Timothy J. Purcell, Ian Buck, William R. Mark, and Pat Hanrahan. Ray Tracing on Programmable Graphics Hardware. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2002)*, 21(3):703–712, July 2002.

Matthias Raab, Daniel Seibert, and Alexander Keller. Unbiased Global Illumination with Participating Media. In *Monte Carlo and Quasi-Monte Carlo Methods 2006*, pages 591–606, 2008.

Zhong Ren, Rui Wang, John Snyder, Kun Zhou, Xinguo Liu, Bo Sun, Peter-Pike Sloan, Hujun Bao, Qunsheng Peng, and Baining Guo. Real-Time Soft Shadows in Dynamic Scenes using Spherical Harmonic Exponentiation. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2006)*, 25(3): 977–986, July 2006.

Tobias Ritschel, Thorsten Grosch, Min H. Kim, Hans-Peter Seidel, Carsten Dachsbacher, and Jan Kautz. Imperfect Shadow Maps for Efficient Computation of Indirect Illumination. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia 2008)*, 27(5):129:1–129:8, December 2008.

Tobias Ritschel, Thomas Engelhardt, Thorsten Grosch, Hans-Peter Seidel, Jan Kautz, and Carsten Dachsbacher. Micro-Rendering for Scalable, Parallel Final Gathering. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia 2009)*, 28(5):132:1–132:8, December 2009a.

Tobias Ritschel, Thorsten Grosch, and Hans-Peter Seidel. Approximating Dynamic Global Illumination in Image Space. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, I3D '09, pages 75–82, 2009b.

Tobias Ritschel, Elmar Eisemann, Inwoo Ha, James D. K. Kim, and Hans-Peter Seidel. Making Imperfect Shadow Maps View-Adaptive: High-Quality Global Illumination in Large Dynamic Scenes. *Computer Graphics Forum*, 30(8):2258–2269, 2011.

Steven M. Rubin and Turner Whitted. A 3-Dimensional Representation for Fast Rendering of Complex Scenes. *Computer Graphics (Proceedings of SIGGRAPH 1980)*, 14(3):110–116, July 1980.

Holly E. Rushmeier and Kenneth E. Torrance. The Zonal Method for Calculating Light Intensities in the Presence of a Participating Medium. *Computer Graphics (Proceedings of SIGGRAPH 1987)*, 21(4):293–302, August 1987.

Szymon Rusinkiewicz and Marc Levoy. QSplat: A Multiresolution Point Rendering System for Large Meshes. In *Proceedings of SIGGRAPH 2000*, Annual Conference Series, pages 343–352, 2000.

Takafumi Saito and Tokiichiro Takahashi. Comprehensible Rendering of 3-D Shapes. *Computer Graphics (Proceedings of SIGGRAPH 1990)*, 24(4): 197–206, September 1990.

Marco Salvi, Kiril Vidimče, Andrew Lauritzen, and Aaron Lefohn. Adaptive Volumetric Shadow Maps. In *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering 2010)*.

Thorsten Scheuermann and Justin Hensley. Efficient Histogram Generation using Scattering on GPUs. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, I3D '07, pages 33–37, 2007.

Benjamin Segovia, Jean-Claude Iehl, Richard Mitanchey, and Bernard Péroche. Bidirectional Instant Radiosity. In *Rendering Techniques 2006 (Proceedings of the Eurographics Symposium on Rendering)*, EGSR '06, pages 389–397, 2006a.

Benjamin Segovia, Jean-Claude Iehl, Richard Mitanchey, and Bernard Péroche. Non-Interleaved Deferred Shading of Interleaved Sample Patterns. In *Proceedings of the 21st ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware*, pages 53–60, 2006b.

Benjamin Segovia, Jean-Claude Iehl, and Bernard Péroche. Metropolis Instant Radiosity. *Computer Graphics Forum (Proceedings of Eurographics 2007)*, 26(3):425–434, September 2007.

Maxim Shevtsov, Alexei Soupikov, and Alexander Kapustin. Highly Parallel Fast KD-tree Construction for Interactive Ray Tracing of Dynamic Scenes. *Computer Graphics Forum (Proceedings of Eurographics 2007)*, 26(3):395–404, 2007.

François X. Sillion and Claude Puech. *Radiosity & Global Illumination*. Morgan Kaufmann Publishers, Inc., 1994. ISBN 1-55860-277-1.

Peter-Pike Sloan, Jan Kautz, and John Snyder. Precomputed Radiance Transfer for Real-Time Rendering in Dynamic, Low-Frequency Lighting Environments. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2002)*, 21(3):527–536, July 2002.

Peter-Pike Sloan, Naga K. Govindaraju, Derek Nowrouzezahrai, and John Snyder. Image-Based Proxy Accumulation for Real-Time Soft Global Illumination. In *Proceedings of the 15th Pacific Conference on Computer Graphics and Applications*, pages 97–105, 2007.

Tiago Sousa. Crysis Next Gen Effects. Game Developers Conference 2008 Presentations, 2008.

Jos Stam. Multiple Scattering as a Diffusion Process. In P M Hanrahan and WEditors Purgathofer, editors, *Rendering Techniques 1995 (Proceedings of the Eurographics Workshop on Rendering)*, EGWR '95, pages 41–50, 1995.

Dirk Staneker, Dirk Bartz, and Michael Meissner. Improving Occlusion Query Efficiency with Occupancy Maps. In *Proceedings of the 2003 IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, PVG '03, pages 15–, 2003.

Bo Sun, Ravi Ramamoorthi, Srinivasa G. Narasimhan, and Shree K. Nayar. A Practical Analytic Single Scattering Model for Real-Time Rendering. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2005)*, 24(3): 1040–1049, July 2005.

Ivan Edward Sutherland. Sketchpad: A Man-Machine Graphical Communication System. Technical Report 296, MIT Lincoln Laboratory, 1963.

The Khronos Consortium. OpenGL 4.2 Specification. OpenGL SDK, August 2011.

Carlo Tomasi and Roberto Manduchi. Bilateral Filtering for Gray and Color Images. In *Proceedings of the Sixth International Conference on Computer Vision*, ICCV '98, pages 839–, 1998.

Balazs Tóth and Tamas Umenhoffer. Real-time Volumetric Lighting in Participating Media. In *EUROGRAPHICS 2009 Short Papers*, pages 57–60, 2009.

Eric Veach. *Robust Monte Carlo Methods for Light Transport Simulation*. PhD thesis, Standford University, December 1997.

Biri Venceslas, Arqués Didier, and Michelin Sylvain. Real-Time Rendering of Atmospheric Scattering and Volumetric Shadows. *Journal of WSCG '06*, 14(1):65–72, 2006.

Ingo Wald. *Real-Time Ray Tracing and Interactive Global Illumination*. PhD thesis, Universität des Saarlands, 2004.

Ingo Wald. On Fast Construction of SAH Based Bounding Volume Hierarchies. In *Proceedings of the IEEE/EUROGRAPHICS Symposium on Interactive Ray Tracing 2007*, 2007.

Ingo Wald and Vlastimil Havran. On Building Fast KD-Trees for Ray Tracing, and on Doing that in O(NlogN). In *Proceedings of the IEEE/EUROGRAPHICS Symposium on Interactive Ray Tracing 2006*, pages 61–69, 2006.

Ingo Wald, Thiago Ize, Andrew Kensler, Aaron Knoll, and Steven G. Parker. Ray Tracing Animated Scenes using Coherent Grid Traversal. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2006)*, 25(3):485–493, 2006.

Ingo Wald, Solomon Boulos, and Peter Shirley. Ray Tracing Deformable Scenes using Dynamic Bounding Volume Hierarchies. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007)*, 26(1), 2007a.

Ingo Wald, William R. Mark, Johannes Günther, Solomon Boulos, Thiago Iye, Warren Hunt, Steven G. Parker, and Peter Shirley. State of the Art in Ray Tracing Animated Scenes. In *STAR Proceedings of Eurographics*, pages 89–116, 2007b.

Bruce Walter, Sebastian Fernandez, Adam Arbree, Kavita Bala, Michael Donikian, and Donald P. Greenberg. Lightcuts: A Scalable Approach to Illumination. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2005)*, 24(3):1098–1107, July 2005.

Bruce Walter, Adam Arbree, Kavita Bala, and Donald P. Greenberg. Multidimensional Lightcuts. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2006)*, 25(3):1081–1088, July 2006.

Xi Wang, Xin Tong, Stephen Lin, Shi-Min Hu, Baining Guo, and Heung-Yeung Shum. Generalized displacement maps. In *Rendering Techniques 2004 (Proceedings of the Eurographics Symposium on Rendering)*, EGSR ’04, pages 227–234, July 2004.

Gregory J. Ward. Measuring and Modeling Anisotropic Reflection. *Computer Graphics (Proceedings of SIGGRAPH 1992)*, 26(2):265–272, July 1992.

Gregory J. Ward, Francis M. Rubinstein, and Robert D. Clear. A Ray Tracing Solution for Diffuse Interreflection. *Computer Graphics (Proceedings of SIGGRAPH 1988)*, 22(4):85–92, June 1988.

Hank Weghorst, Gary Hooper, and Donald P. Greenberg. Improved Computational Methods for Ray Tracing. *ACM Transactions on Graphics*, 3(1): 52–69, January 1984.

Turner Whitted. An Improved Illumination Model for Shaded Display. *Communications of the ACM*, 23(6):343–349, June 1980.

Lance Williams. Pyramidal Parametrics. *Computer Graphics (Proceedings of SIGGRAPH 1983)*, 17(3):1–11, July 1983.

E. R. Woodcock, T. Murphy, P. J. Hemmings, and T. C. Longworth. Techniques used in the GEM code for Monte Carlo neutronics calculations in reactors and other systems of complex geometry. In *Applications of Computing Methods to Reactor Problems*, pages 557–579. Argonne National Laboratory, 1965.

Chris Wyman and Shaun Ramsey. Interactive Volumetric Shadows in Participating Media with Single-Scattering. In *Proceedings of the IEEE/EUROGRAPHICS Symposium on Interactive Ray Tracing 2008*, pages 87–92, August 2008.

Chris Wyman, Greg Nichols, and Jeremy Shopf. Fast, Stencil-Based Multiresolution Splatting for Indirect Illumination. In *GPU Pro Advanced Rendering Techniques*, pages 199–213. A K Peters Ltd., 2010. ISBN 978-1-56881-472-9.

Sung-Eui Yoon, Brian Salomon, and Dinesh Manocha. Interactive View-Dependent Rendering with Conservative Occlusion Culling in Complex Environments. In *Proceedings of the 14th IEEE Visualization*, VIS '03, pages 22–, 2003.

Yonghao Yue, Kei Iwasaki, Bing-Yu Chen, Yoshinori Dobashi, and Tomoyuki Nishita. Unbiased, Adaptive Stochastic Sampling for Rendering Inhomogeneous Participating Media. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia 2010)*, 29(6):177:1–177:8, December 2010.

Harold R. Zatz. Galerkin Radiosity: A Higher Order Solution Method for Global Illumination. In *Proceedings of SIGGRAPH 1993*, Annual Conference Series, pages 213–220, 1993.

Hansong Zhang, Dinesh Manocha, Tom Hudson, and Kenneth E. Hoff, III. Visibility Culling using Hierarchical Occlusion Maps. In *Proceedings of SIGGRAPH 1997*, Annual Conference Series, pages 77–88, 1997.

Kun Zhou, Qiming Hou, Rui Wang, and Baining Guo. Real-Time KD-Tree Construction on Graphics Hardware. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia 2008)*, 27(5):126:1–126:11, December 2008.