Visually Pleasing Real-time Global Illumination Rendering for Fully-dynamic Scenes

Zhao Dong

Max-Planck-Institut Informatik
Saarbrücken, Germany

Dissertation zur Erlangung des Grades
Doktor der Ingenieurwissenschaften (Dr.–Ing.)
der Naturwissenschaftlich-Technischen Fakultäten
der Universität des Saarlandes

Eingereicht am 03. März 2011 in Saarbrücken.

Betreuender Hochschullehrer — Supervisor

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

Gutachter — Reviewers

Prof. Dr. Hans-Peter Seidel	MPI Informatik	Saarbrücken, Germany
Prof. Dr. Jan Kautz	University College London	London, UK
Prof. Dr. Thorsten Grosch	University of Magdeburg	Magdeburg, Germany

Dekan — Dean

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

Datum des Kolloquiums — Date of Defense

03. March 2011 in Saarbrücken

Prüfungsausschuss — Board of Examiners

Head of Colloquium	Prof. Dr. Joachim Weickert	Universität des Saarlandes Saarbrücken, Germany
Examiner	Prof. Dr. Hans-Peter Seidel	MPI Informatik Saarbrücken, Germany
Examiner	Prof. Dr. Thorsten Grosch	University of Magdeburg Magdeburg, Germany
Protocol	Dr. Carsten Stoll	MPI Informatik Saarbrücken, Germany

Zhao Dong
Max-Planck-Institut Informatik
Campus E1 4 (Room 117)
66123 Saarbrücken, Germany
dong@mpi-inf.mpg.de

Dedicated to the wonderful land and people of Deutschland.
Thank you for five amazing years!

Abstract

Global illumination (GI) rendering plays a crucial role in the photo-realistic rendering of virtual scenes. With the rapid development of graphics hardware, GI has become increasingly attractive even for real-time applications nowadays. However, the computation of physically-correct global illumination is time-consuming and cannot achieve real-time, or even interactive performance. Although the real-time GI is possible using a solution based on precomputation, such a solution cannot deal with fully-dynamic scenes. This dissertation focuses on solving these problems by introducing visually pleasing real-time global illumination rendering for fully-dynamic scenes.

To this end, we develop a set of novel algorithms and techniques for rendering global illumination effects using the graphics hardware. All these algorithms not only result in real-time or interactive performance, but also generate comparable quality to the previous works in off-line rendering. First, we present a novel implicit visibility technique to circumvent expensive visibility queries in hierarchical radiosity by evaluating the visibility implicitly. Thereafter, we focus on rendering visually plausible soft shadows, which is the most important GI effect caused by the visibility determination. Based on the pre-filtering shadow mapping theory, we successively propose two real-time soft shadow mapping methods: “convolution soft shadow mapping” (CSSM) and “variance soft shadow mapping” (VSSM). Furthermore, we successfully apply our CSSM method in computing the shadow effects for indirect lighting. Finally, to explore the GI rendering in participating media, we investigate a novel technique to interactively render volume caustics in the single-scattering participating media.

Kurzfassung

Das Rendern globaler Beleuchtung ist für die fotorealistische Darstellung virtueller Szenen von entscheidender Bedeutung. Dank der rapiden Entwicklung der Grafik-Hardware wird die globale Beleuchtung heutzutage sogar für Echtzeitanwendungen immer attraktiver. Trotz allem ist die Berechnung physikalisch korrekter globaler Beleuchtung zeitintensiv und interaktive Laufzeiten können mit "standard Hardware" noch nicht erzielt werden. Obwohl das Rendering auf der Grundlage von Vorberechnungen in Echtzeit möglich ist, kann ein solcher Ansatz nicht auf voll-dynamische Szenen angewendet werden.

Diese Dissertation zielt darauf ab, das Problem der globalen Beleuchtungsberechnung durch Einführung von neuen Techniken für voll-dynamische Szenen in Echtzeit zu lösen. Dazu stellen wir eine Reihe neuer Algorithmen vor, die die Effekte der globaler Beleuchtung auf der Grafik-Hardware berechnen. All diese Algorithmen erzielen nicht nur Echtzeit bzw. interaktive Laufzeiten sondern liefern auch eine Qualität, die mit bisherigen off-line Methoden vergleichbar ist. Zunächst präentieren wir eine neue Technik zur Berechnung impliziter Sichtbarkeit, die aufwändige Sichbarkeitstests in hierarchischen Radiosity-Datenstrukturen vermeidet. Anschliessend stellen wir eine Methode vor, die weiche Schatten, ein wichtiger Effekt für die globale Beleuchtung, in Echtzeit berechnet. Auf der Grundlage der Theorie über vorgefilterten Schattenwurf, zeigen wir nacheinander zwei Echtzeitmethoden zur Berechnung weicher Schattenwürfe: "Convolution Soft Shadow Mapping" (CSSM) und "Variance Soft Shadow Mapping" (VSSM). Darüber hinaus wenden wir unsere CSSM-Methode auch erfolgreich auf den Schatteneffekt in der indirekten Beleuchtung an. Abschliessend präsentieren wir eine neue Methode zum interaktiven Rendern von Volumen-Kaustiken in einfach streuenden, halbtransparenten Medien.

Summary

Global illumination (GI) rendering plays a crucial role in the photorealistic rendering of virtual scenes. It has long been applied in the special effect production in the film industry. With the rapid development of the graphics hardware, GI has become increasingly attractive for real-time applications, like video games, nowadays. However, the computation of the fully physically-correct global illumination is usually time-consuming and cannot achieve real-time, or even interactive performance. Although the real-time GI is possible using a solution based on precomputation, such a solution cannot deal with fully-dynamic scenes. This dissertation focuses on solving these problems by introducing the visually pleasing real-time GI rendering for fully-dynamic scenes. The visually pleasing GI rendering is motivated not only by improving the performance, but also by fulfilling the visual perception of human beings. Some research works already prove that in lots of scenarios, the fully physically-correct GI is not necessary for the human perception. Since the final goal of our renderings is to provide images for perception, we can derive some reasonable approximations from the fundamental theory of GI to achieve the visually pleasing real-time GI renderings.

To this end, we develop a set of novel algorithms and techniques for rendering visually pleasing GI effects using graphics hardware. All these algorithms not only result in real-time or interactive performance, but also generate comparable quality to the previous off-line rendering. First, we present a novel implicit visibility technique to circumvent expensive visibility queries in hierarchical radiosity by evaluating the visibility implicitly. Thereafter, we focus on rendering visually plausible soft shadows, which is the most important GI effect caused by the visibility determination. Based on the pre-filtering shadow mapping theory, we successively propose two real-time soft shadow mapping methods: “convolution soft shadow mapping” (CSSM) and “variance soft shadow mapping” (VSSM). Furthermore, we successfully apply our CSSM method in computing the shadow effects for indirect lighting. Finally, to explore the GI rendering in participating media, we investigate a novel technique to interactively render volume caustics in the single-scattering participating media. In brief, in Part.II and Part.III, we focus on how to approximately solve the visibility determination in GI which is usually the bottleneck of the whole GI algorithm. In Part.IV, we step further to deal with the GI effect: volume caustics in participating media. Before starting the detailed introductions of all these algorithms, in Part.I, we also lay out the general theoretical background materials that are needed to understand our novel algorithms and techniques.

Implicit Visibility Visibility determination usually dominates the performance of the GI algorithm. We start with circumventing the visibility queries in radiosity

methods. Usually, ray casting is utilized to explicitly determine the visibility between two elements in the scene. However, its performance is slow and prevents the algorithm to reach interactive or real-time performance. Compared with the explicit way, we propose to implicitly evaluate the visibility between individual scene elements in Chapter 4. Our method is inspired by the principles of hierarchical radiosity and tackles the visibility problem by implicitly evaluating the mutual visibility while constructing a hierarchical link structure between scene elements. Our novel method is able to reproduce a large variety of GI effects for moderately sized scenes at interactive rates, such as indirect lighting, soft shadows under environment map lighting, as well as area light sources.

Pre-filtering Soft Shadow Maps and their Applications Soft shadow is one of the most important global illumination effects and computing a soft shadow has long been an important topic in the rendering research. We successively present two kinds of visually plausible soft shadow mapping methods which are based on the pre-filtering shadow map theory and implemented in the percentage closer soft shadow (PCSS) [Fernando05a] framework. The first one is the so-called "convolution soft shadow mapping" (CSSM) which is based on the convolution in Fourier space to approximate the traditional shadow test function. The key contribution of CSSM is the convolution pre-filtering theory which can be applied in both the average blocker depth step and the soft shadow computation step of PCSS framework. One major limitation of CSSM is: achieving high-quality soft shadows increases the number of Fourier basis terms to be at least four, so that large amounts of texture memory are required to store Fourier basis terms, making it less practical. To overcome this problem, we present a second method called "variance soft shadow mapping" (VSSM). VSSM is based on a one-tailed version of Chebyshev's inequality and requires a much lower amount of texture memory. Both CSSM and VSSM can achieve visually plausible soft shadow rendering at the real-time performance. Especially for VSSM, more than 100 fps can be achieved for very complex scenes.

Moreover, motivated by the concept of clustered visibility, we extend the CSSM method to compute the shadow effects of indirect lighting. Since the perception of the indirect shadows is not sensitive, a reasonable approximated shadow result is usually sufficient. We propose a highly efficient method to compute indirect illumination by clustering virtual point lights (VPLs), which represent the indirect illumination, into virtual area lights (VALs). A single visibility value is shared for all VPLs in a cluster, which we compute with the CSSM method to avoid banding artifacts. Our method achieves both visually plausible quality and real-time frame rates for large and dynamic scenes.

Interactive Volume Caustics Furthermore, we investigate the GI effect: volume caustics rendering in the single-scattering participating media. Our method

is based on the observation that line rendering of illumination rays into the screen buffer establishes a direct light path between the viewer and the light source. This connection is introduced via a single scattering event for every pixel affected by the line primitive. Based on this connection, the radiance contributions of these light paths to each of the pixels can be computed and accumulated independently using the graphics hardware. Our method achieves high-quality results at real-time or interactive frame rates for large and dynamic scenes containing the homogeneous or inhomogeneous participating media.

Finally, we conclude the thesis and point out the future works in Chapter.9.

Zusammenfassung

Das Rendern globaler Beleuchtung ist für die fotorealistische Darstellung virtueller Szenen von entscheidender Bedeutung. Es findet seit langem Anwendung bei der Erzeugung von Spezialeffekten in der Filmindustrie. Dank des rapiden Fortschritts in der Grafik-Hardwareentwicklung wurde die globale Beleuchtung heutzutage sogar für Echtzeitanwendungen, wie z.B. Computer Spiele, attraktiv. Trotz allem ist die Berechnung gänzlich physikalisch korrekter globaler Beleuchtung zeitintensiv, und es können weder Echtzeit noch interaktive Laufzeiten erzielt werden. Obwohl das Rendering auf der Grundlage von Vorberechnungen in Echtzeit möglich ist, kann ein solcher Ansatz nicht mit dynamischen Szenen umgehen.

Diese Dissertation zielt darauf ab, diese Probleme mittels der Einführung von “visuell ansprechendem Rendering” globaler Beleuchtung für voll-dynamische Szenen in Echtzeit zu lösen. Dabei ist neben der Geschwindigkeitsverbesserung auch die menschliche visuelle Wahrnehmung von Wichtigkeit. Einige wissenschaftliche Arbeiten beweisen bereits, dass in vielen Szenarien, die physikalisch korrekte globale Beleuchtung für die menschliche Wahrnehmung nicht notwendig ist. Da es die Aufgabe unseres Renderns ist, realistische Bilder zur Betrachtung von Menschen zu erzeugen, können wir einige akzeptable Näherungen aus der Theorie der globalen Beleuchtung herleiten, um visuell ansprechendes Rendering globaler Beleuchtung zu erzielen.

Dazu entwickeln wir eine Reihe neuer Algorithmen und Techniken zum Rendern visuell ansprechender Effekte der globaler Beleuchtung auf der Grafik Hardware. Unsere Algorithmen erzielen nicht nur Echtzeit oder interaktive Laufzeit, sondern liefern auch eine Qualität, die mit bisherigen off-line Methoden vergleichbar ist. Zuerst stellen wir eine neue Technik zur Berechnung der Sichtbarkeit vor, um aufwändige Sichbarkeitsabfragen in hierarchischen Radiosity-Datenstrukturen zu umgehen, indem wir die Sichtbarkeit implizit auswerten. Anschliessend betrachten wir das Rendern visuell realistischer weicher Schatten, welches ein wichtiger Effekt für die globale Beleuchtung darstellt und durch die Auswertung der Sichtbarkeit hervorgerufen wird. Auf der Grundlage der Theorie über vorgefilterten Schattenwurf stellen wir nacheinander zwei EchtzeitMethoden zur Berechnung weicher Schattenwürfe vor: “Convolution Soft Shadow Mapping” (CSSM) und “Variance Soft Shadow Mapping” (VSSM). Darüber hinaus wenden wir unsere CSSM-Methode erfolgreich an, um den Schatteneffekt auch für die indirekte Beleuchtung zu berechnen. Abschliessend erforschen wir eine neue Technik zum interaktiven Rendern von Volumen-Kaustiken in einfach streuenden, teildurchlässigen Medien, um das Rendern globaler Beleuchtung in halbtransparenten Medien zu untersuchen. Kurz gefasst, in Teil. II und Teil. III konzentrieren wir uns darauf, wie man die Sichtbarkeit bei globaler Beleuchtung

näherungsweise bestimmen kann, was im Normalfall der Engpass des gesamten Algorithmus zur globalen Beleuchtung ist. In Teil. IV fahren wir mit der Behandlung des Effekts der Volumen-Kaustiken in teiltransparenten Medien fort.

Implizite Sichtbarkeit Die Bestimmung der Sichtbarkeit ist ein massgeblicher Faktor für die Geschwindigkeit des Algorithmus zur Berechnung der globalen Beleuchtung. Wir beginnen damit, die Sichtbarkeitstests mittels Schnittpunktberechnungen von Strahlen zu umgehen. Normalerweise wird ein Strahl verwendet, um explizit die Sichtbarkeit zwischen zwei Elementen einer Szene zu bestimmen. Allerdings mindert dieses Vorgehen die Geschwindigkeit des Algorithmus und behindert somit das Erreichen interaktiver Laufzeit oder Echtzeit. Im Vergleich zum expliziten Ansatz, beschreiben wir in Kapitel. 4 wie die Sichtbarkeit zwischen einzelnen Szene-Elementen implizit berechnet werden kann. Unsere Methode wurde inspiriert von dem Prinzip des hierarchischen Radiosity und widmet sich dem Sichtbarkeitsproblem durch implizite Auswertung mit Hilfe einer hierarchischen Verbindungsstruktur zwischen Szenen-Elementen. Unsere neue Methode ermöglicht es, viele Effekte der globaler Beleuchtung für mittelgrosse Szenen mit interaktiven Bildwiederholraten wiederzugeben; wie z.B. indirekte Beleuchtung und weiche Schatten sowohl unter Umgebungsbeleuchtung als auch unter Flächenlichtquellen.

Vorfiltern von weichen Schattenwürfen und deren Anwendung Weiche Schatten sind einer der wichtigsten Effekte der globalen Beleuchtung und standen lange Zeit im Fokus der Rendering-Forschung. Basierend auf der Theorie zum vorgefilterten Schattenwurf (implementiert im “Percentage Closer Soft Shadow” (PCSS) [Fernando05a] system), demonstrieren wir zwei Methoden zum visuell realistischen Rendern von weichem Schattenwurf. Die erste Methode ist das sogenannte “Convolution Soft Shadow Mapping” (CSSM), welches auf Faltung im Fourier-Raum basiert und womit die Schattenfunktion näherungsweise berechnet werden kann. Der Hauptbeitrag, den CSSM leistet, ist dass die Theorie vorgefilterter Schatten sowohl im Schritt zur Bestimmung der durschnittlichen Blockadetiefe als auch im Schritt zur Berechnung der weichen Schatten im wächst-system angewendet werden kann. Eine grundlegende Beschränkung von CSSM ist Folgendes: hochqualitative weiche Schatten erfordern mehrere Fourier-Basisterme (mehr als 4), sodass grosse Mengen an Texturspeicher benötigt werden. Dadurch ist diese Methode wenig praktikabel. Um dies zu umgehen, stellen wir eine zweite Methode, das sogenannte “Variance Soft Shadow Mapping” (VSSM), vor. VSSM basiert auf einer Version der Chebyshev-Ungleichung und benötigt viel weniger Texturspeicher. Sowohl CSSM als auch VSSM können in Echtzeit visuell realistische weiche Schatten rendern. Insbesondere kann VSSM selbst bei komplexen Szenen Bilder mit mehr als 100 Bildern pro Sekunde auf aktueller Hardware berechnen.

Darüberhinaus, motiviert durch das Verfahren der “geclusterten Sichtbarkeit”, erweitern wir die CSSM-Methode, um auch den Schatteneffekt der indirekten Beleuchtung zu berechnen. Da die visuelle Wahrnehmung indirekter Schatten nicht sehr genau ist, genügt im Normalfall ein hinreichend angenäherter Schatten. Wir führen eine hocheffiziente Methode ein, um die indirekte Beleuchtung zu berechnen. Dies geschieht durch Gruppierung virtueller Punktlichtquellen (Cluster) und Approximation durch virtuelle Flächenlichtquellen, die die indirekte Beleuchtung darstellen. Jeder Cluster von virtuellen Punktlichtquellen erhält einen gemeinsamen Sichtbarkeitswert, der durch die CSSM-Methode berechnet wird, um Artefakte zu vermeiden. Unser Ansatz erzielt sowohl visuell ansprechende Qualität als auch Wiederholraten in Echtzeit für große und dynamische Szenen.

Interactive Volumen-Kaustiken Desweiteren untersuchen wir den folgenden Effekt globaler Beleuchtung: Rendern von Volumen-Kaustiken in einfachstreuenden, teiltransparenten Medien. Unsere Methode basiert auf der Beobachtung, dass das Rendern von Beleuchtungstrahlen als Linien im Bildschirmpuffer einen direkten Lichtpfad zwischen dem Betrachter und der Lichtquelle erzeugt. Diese Verbindung wird für jeden Pixel erstellt, der von der Linie geschnitten wird, und zwar durch eine einzige Streuung. Basierend auf dieser Verbindung kann der Strahlungsdichteanteil dieser Lichtpfade unter Verwendung der Grafik-Hardware zu jedem der Pixel berechnet und unabhängig aufsummiert werden. Unser Verfahren erzielt hochqualitative Resultate bei interaktiven oder Echtzeit-Wiederholraten für große und dynamische Szenen mit homogenen oder inhomogenen teiltransparenten Medien.

Im letzten Kapitel. 9 folgt eine Zusammenfassung dieser Dissertation und ein Ausblick auf zuküftige Forschung und Verbesserungen unserer hier vorgestellten Verfahren.

Acknowledgements

This thesis would not have been possible without the help and support of many individuals. First of all, I would like to thank my supervisor Prof. Dr. Hans-Peter Seidel. I am extremely thankful to Prof. Seidel who gave me the opportunity to work in the truly remarkable research environment in the Computer Graphics group at the MPI and supported me throughout my thesis work. It was a privilege to work in one of the best Computer Graphics groups in the world.

I owe special thanks to Prof. Dr. Jan Kautz and Prof. Dr. Thorsten Grosch for being reviewers of my thesis. Both have guided me and shared their experiences with me during my Ph.D. studies. Over the past few years we have all along closely cooperated with each other.

I want to especially thank Prof. Dr. Christian Theobalt, who was my mentor in my first two years of Ph.D.. His support and kindness greatly helped me to find out my own interested research field. I also want to especially thank Dr. John Snyder who invited me to do an internship at the computer graphics group of Microsoft Research, Redmond. John is a wonderful researcher and mentor, who provided me so many advices and guidance for both my research and career. Also gracious thanks to the computer graphics group at Microsoft Research, especially Dr. Jim Blinn for inviting me as an intern.

Gracious thanks for my research collaborators: Tobias Ritschel, Sungkil Lee, Kaleigh Smith, Robert Strzodka, Thomas Annen and Elmar Eissemann. Their wonderful insights and support helped me a lot during the tough research process. They are not only my collaborators, but also my good friends. Also Robert Herzog and Michael Schwarz always supported me for discussions in depth when I got confused, and I appreciate their help very much.

I also own special thanks for my two officemates: Prof. Dr. Ivo Ihrke and Jens Kerber for their great support for me. I was really fortunate to have so many great times and memorable moments with them. I own special thanks to Naveed Ahmed for all his support in various ways during these years, and he truly is one of my best friends. Also gracious thanks for my Chinese colleagues: Kuangyu Shi and Tongbo Chen who supported and helped me so much during my Ph.D. studies.

Since I came from abroad to Germany, I usually have lots of administrative processing stuffs to bother Ms. Sabine Budde and Ms. Conny Liegl, who are the secretary of AG4. They are always kind and generous to support me with their professional works, and gracious thanks for their wonderful works.

Furthermore, I own thanks to all my colleagues of the computer graphics group at MPI. It is these colleagues who make MPI such a wonderful research place. I cannot name all of them, but I would like to especially thank the following people (alphabetical order in last name): Edioson de Aguiar, Tunc Ozan Aydin, Andreas

Baak, Lionel Baboud, Martin Bokeloh, Karol Myszkowski, Carsten Stoll, Martin Sunkel, Art Tevs.

I would like to further thank my previous supervisors Prof. Qunsheng Peng and Prof. Wei Chen from the State Key Lab of CAD&CG of Zhejiang University, who really convinced me that graphics is a wonderful topic to do research in, and hence I decided to pursue a Ph.D. in this field.

Finally I would like to thank my wife Yi Chai and my parents for their support and sacrifices throughout the years of this thesis.

Contents

Chapter 1

Introduction

1.1 Motivation

Computer graphics involves creating, or *rendering*, images of arbitrary virtual environments. A long sought research goal in computer graphics is to render images of these environments as realistically as possible to finally achieve images indistinguishable from photographs. Such an image synthesis process is usually called the photorealistic rendering. The probably most involved part in the photorealistic rendering is the global illumination (GI). GI is a general name for a group of algorithms used in 3D computer graphics that are meant to add more realistic lighting to 3D scenes. The general definition of the GI effects includes realistic soft shadow, indirect lighting (interreflection), caustics and so on. All these GI effects deliver important cues in the perception of 3D virtual scenes, which are important for lots of applications, such as material and architectural design, film special effects production, etc. While the fundamental theory behind the creation of the GI effects is well-studied and often considered as solved, its computing efficiency is not. Improving the rendering performance is therefore the main motivation of current research in the GI.

In the rendering field of computer graphics, lots of algorithms have been developed to simulate the GI effects accurately or approximately, such as ray tracing, radiosity, ambient occlusion [Zhukov98], precomputed radiance transfer (PRT) [Sloan02] and so on. Over the past several years, these methods have been widely applied in lots of fields. For example, in the field of film special effects production, the high-quality GI renderings create stunning visual effects, as can be seen in the movies like Shrek or Avatar. Yet the computation time of synthesizing such high-quality, realistic images is still very long and one single movie frame usually takes several hours to render. Although some methods, like PRT, can achieve real-time or interactive GI rendering relying on precomputation, they

usually restrict the input scene to be static or contains only rigid transformation.

With the rapid development of the graphics hardware, GI has become increasingly attractive even for the real-time applications, e.g. video games, nowadays. However, the real-time rendering performance is the basic requirement for the game rendering. The computation of fully physically-correct global illumination is usually time-consuming and cannot achieve real-time or even interactive performance. Another requirement for the game rendering is no restriction for the input geometry. The deformable animation or other dynamic models are very common in the game scenes. Therefore, the precomputation-based GI rendering method cannot be fully applied in such a scenario.

Recently, the perception researches in computer graphics reveal that in lots of scenarios the fully physically-correct GI rendering is not necessary based on the accuracy of human perceptions. [Ramasubramanian99, Myszkowski01, Yu09]. This motivates the research works to derive reasonable approximations from the fundamental theory of GI to achieve visually pleasing real-time GI renderings. For example, although the ambient occlusion [Zhukov98] is a crude approximation to full GI, it has the nice property of offering a better perception of the 3d shape of the displayed objects. Therefore, the ambient occlusion technique has been widely applied in real-world applications.

Motivated by aforementioned reasons, in this dissertation, we introduce a set of novel algorithms and techniques using the graphics hardware to achieve visually pleasing real-time rendering for GI effects. All these algorithms can achieve results in real-time or interactive performance and their rendering quality is comparable to the traditional offline rendering. Furthermore, all of our methods do not impose any restrictions for the input scenes and can be easily applied in the real scenarios.

1.2 Problem Statement

Creating real-time visually pleasing GI rendering has several important requirements:

- The rendering quality should be visually plausible and comparable to the ground truth reference generated by the offline rendering. The minor differences between the generated image and the reference image is acceptable, but the visually noticeable artifacts should be avoided.

- The rendering performance should achieve real-time or at least interactive rates for arbitrary cases. The algorithm itself should contain performance potential for the future improved graphics hardware.

- There should not be any kinds of scene-related precomputation involved, so that there is no restriction imposed on the input scenes.

Considering the GI effects: realistic soft shadow, indirect lighting (interreflection), caustics and so on, our research works focus on designing the reasonable approximate GI solutions, which fulfill the above requirements, for all the effects to achieve visually pleasing real-time rendering.

The visibility determination is always the bottleneck for the GI rendering. Therefore, we will treat how to reasonably approximate the visibility test and improve its performance as our major target in the research. In radiosity methods, ray casting is usually applied to explicitly compute the visibility between scene elements, which is in low efficiency. To solve this problem, in Chapter. 4 we introduce the *implicit visibility* scheme to attack this bottleneck, which achieves the GI renderings at the near-real-time frame rates for fully dynamic scenes.

Realistic soft shadow is probably the most important and difficult effect for the GI rendering. However, it is also limited by the performance in visibility determination. Soft shadow is a long-standing hard problem in rendering. Actually, for any existing soft shadow algorithm, a compromise between the quality and performance always exists. In Chapter. 5 and Chapter. 7 of this dissertation, we propose two different soft shadow mapping methods to deal with the generation of the visually plausible real-time soft shadows. Further, we extend our method in Chapter. 6 to render the soft shadows of indirect lighting in real-time and also receive promising results.

Recently, the GI effects in the participating medium start to gather more and more attentions. However, the interactions of light in a participating medium, like scattering, will increase the computation burden of the GI simulation and make it more difficult. Volume caustics are intricate illumination patterns formed by the light first interacting with a specular surface and subsequently being scattered inside a participating medium. Previous techniques [Jensen98] for generating volume caustics are time-consuming and are considered impossible to achieve in real-time before. In Chapter. 8, we introduce a novel volume caustics rendering method for the single-scattering participating media. Our method achieve high-quality results at real-time/interactive frame rates for complex dynamic scenes containing homogeneous/inhomogeneous participating media.

1.3 Main Contributions and Organization of the Thesis

This thesis is divided into 4 parts and contains 9 chapters. Apart from Part. I, which deals with the necessary theoretical and technical background and covers

the preliminaries, each subsequent part focuses on one specific GI effect. The algorithmic solutions described in part. II, III,and IV have been published before in a variety of peer-reviewed conference and journal articles [Dong07, Annen08a, Dong09, Yang10, Hu10]. We will conclude this thesis and discuss some future works in Chapter. 9. The major contributions of the thesis are briefly summarized in the following sections:

1.3.1 Part I - Background and Basic Definitions

This part covers the theoretical preliminaries required for the understanding of the rest of the thesis. In Chapter. 2, the foundation of our works is laid. We will introduce the general theoretical background materials that are needed to understand our new algorithms and techniques. In Chapter. 3, we briefly summarize and discuss the most related work in the GI rendering field. Later, we introduce several recent research works that follow our works in this thesis.

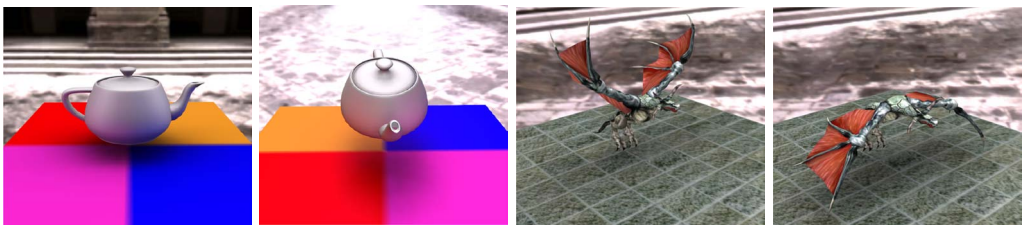


Figure 1.1: Interactive global illumination effects for fully dynamic scenes. The bin discretization is $6 \times 12 \times 12$ and the indirect lighting is one-bounce. The performance for teapot scene (3878 vertices) is 6.43fps and for dragon scene (2670 vertices) is 7.53fps.

1.3.2 Part II - Interactive Global Illumination Using Implicit Visibility

In order to improve the speed of visibility determination for radiosity-like methods, in Chapter 4 we propose to implicitly evaluate the visibility between individual scene elements. Our method is inspired by the principles of hierarchical radiosity [Hanrahan91] and tackles the visibility problem by implicitly evaluating mutual visibility while constructing a hierarchical link structure between scene elements. Both the construction of the hierarchy and the computation of the final lighting solution can be efficiently mapped onto the CPU and the GPU. By means of the same efficient and easy-to-implement framework, we are able to reproduce

a large variety of the GI effects for moderately sized scenes, such as the indirect lighting, soft shadows under environment map lighting as well as area light sources (as shown in Fig. 1.1).

1.3.3 Part III - Convolution Soft Shadow Maps and Its Applications

Based on the convolution theory of the pre-filtered shadow test [Annen07], we develop a fast, approximate pre-filtering soft shadow mapping method which is the so-called *convolution soft shadow maps* (CSSM) in Chapter. 5. CSSM is implemented in the soft shadow framework of *percentage closer soft shadow* (PCSS) [Fernando05a] and achieves several hundred frames per second for a single area light source. This allows for approximating environment illumination with a sparse collection of area light sources and yields real-time frame rates. Furthermore, we present this technique for rendering fully dynamic scenes under arbitrary environment illumination, which does not require any precomputation. The rendering results are shown in Fig. 1.2.

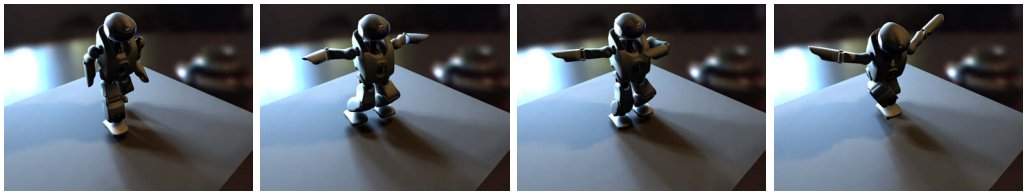


Figure 1.2: A fully dynamic animation of a dancing robot under environment map lighting rendered at 29.4 **fps without any precomputation. Incident radiance is approximated by 30 area light sources (**256×256 **shadow map resolution each).**

Visibility computation is also the bottleneck when rendering the indirect illumination. However, recent perception researches [Yu09] have demonstrated that the accurate visibility is unnecessary for the indirect illumination. To exploit this insight, we cluster a large number of *virtual point lights* (VPLs), which represent the indirect illumination when using the instant radiosity [Keller97], into a small number of *virtual area lights* (VALs). This allows us to approximate the soft shadow of the indirect lighting using our CSSM algorithm. Such an approximate and fractional from-area visibility is faster to compute and avoids banding when compared to the exact binary from-point visibility. In Chapter. 6, our experimental results demonstrate that the perceptual error of this approximation is negligible and the real-time frame rates for large and dynamic scenes can be achieved. The example rendering results are shown in Fig. 1.3.

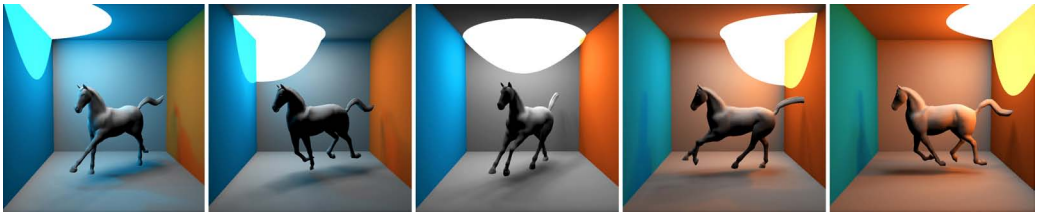


Figure 1.3: One-bounce diffuse global illumination rendered at 800×800 pixels for a scene with dynamic geometry (17 k faces) and dynamic lighting at 19.7 fps. Our method uses soft shadows from 30 area lights to efficiently compute the indirect visibility.

CSSM usually demands of large amounts of texture memory for storing Fourier basis terms. This limitation prevents CSSM to be practically implemented in the real-world interactive applications, like video games. In order to reduce the memory cost and further improve the rendering speed, in Chapter. 7, we develop *Variance Soft Shadow Maps* (VSSM) which extends the *variance Shadow Maps* (VSM) [Donnelly06a] for the soft shadow generation. VSSM is also based on the soft shadow framework of PCSS. To accelerate the average blocker depth evaluation, we derive a novel formulation to efficiently compute the average blocker depth based on pre-filtering. Furthermore, we avoid incorrectly lit pixels commonly encountered in VSM-based methods by appropriately subdividing the filter kernel. As shown in Fig. 1.4, VSSM can render high-quality soft shadows very efficiently (usually over 100 fps) for complex scene settings.

Figure 1.4: Different rendering results generated by our variance soft shadow mapping method without any precomputation. From left to right, the rendering performances and the faces numbers of different input scenes are: 131 fps (76k), 148 fps (141k), 110 fps (120k), 25 fps (9700k).

1.3.4 Part IV - Interactive Global Illumination in Participating Media

Motivated by the interactive applications, in Chapter. 8, we propose a novel volume caustics rendering method for the single-scattering participating media. Our

method is based on the observation that the line rendering of illumination rays into the screen buffer establishes a direct light path between the viewer and the light source. This connection is introduced via a single scattering event for every pixel affected by the line primitive. Since the GPU is a parallel processor, the radiance contributions of these light paths to each of the pixels can be computed and accumulated independently. The implementation of our method is straightforward and we show that it can be seamlessly integrated with existing methods for rendering participating media. As shown in Fig. 1.5, Our method achieves high-quality re-

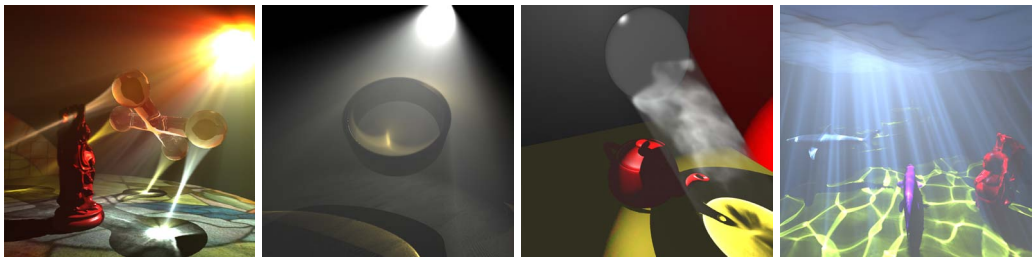


Figure 1.5: Different rendering results generated by our screen-based interactive volume caustics method. Both, specular and refractive volume caustics in homogeneous and inhomogeneous participating media are handled by our technique. From left to right, the rendering performances of different input scenes are: 31 fps, 28 fps, 11 fps, 12.5 fps.

sults at real-time/interactive frame rates for complex dynamic scenes containing homogeneous/inhomogeneous participating media.

Part I

Background and Related Works

Chapter 2

Background Knowledge

In this chapter, we will first begin with the review of the basic quantities and equations which form the foundation of computer graphics rendering. Thereafter, we will explain the soft shadow mapping techniques, especially the *percentage-closer soft shadows* (PCSS) [Fernando05a] which provides the basic rendering framework for our soft shadow mapping algorithms. Considering the performance of the implementation, the general GPU-based rendering pipeline and deferred shading will be further introduced, which has been exploited for all of our methods to improve rendering speed.

2.1 Radiometry

The Global Illumination (GI) effects are all formed by the interaction between the light and physical properties of input scene. Therefore, it is important for us to understand the nature of light and some of the underlying physical properties so as to understand how the light transportation is computed in computer graphics.

From the point of physics, the light is a kind of electromagnetic radiation and it can be interpreted either as an electromagnetic wave (*wave optics*) or as a flow of photons carrying energy (*particle optics*). In computer graphics, we adopt *geometrical optics* which models the light as independent rays which travel in space. Therefore, the interaction between the light and input scene can be modeled as a geometrical problem. For a more detailed description of optics, please refer to [Born64].

In this section, we will firstly explain the basic radiometric terms, then how to solve the general light transport problems.

2.1.1 Basic Terms

In following table, the symbols and corresponding units of basic radiometric terms are shown.

Radiometric term	Symbol	Unit
Radiant energy	Q	J
Radiant flux	Φ	W
Radiant intensity	I	W/sr
Irradiance (incident)	E	W/m^2
Radiosity (outgoing)	B	W/m^2
Radiance	L	W/m^2sr

Let us firstly introduce the basic unit of radiometry: *radiant energy*. Usually it is denoted as Q and its unit is *joule* $[J = W \cdot s]$. The radiant energy represents the sum of all the energy that are carried by n photons over all wavelengths .

Radiant flux Φ is defined to be the radiant energy Q flowing through a surface per unit time dt:

$$\Phi = \frac{dQ}{dt} \tag{2.1}$$

It is also often called *flux* and its unit is *Watt* $[W]$. Based on Φ, if considering the per differential area dA we can get the quantity definition of irradiance E and radiosity B:

$$E = B = \frac{d\Phi}{dA} \tag{2.2}$$

Note, although the quantity of *irradiance* E and *radiosity* B might be the same, the directions of their energy transport are inverse. *Irradiance* $E(x)$ represents the *flux* arriving at a surface point x, and *Radiosity* $B(x)$ represents the *flux* leaving from surface point x.

Compared with *irradiance* E and *radiosity* B, *intensity* I is defined as the ratio of *flux* with respect to solid angle Ω instead of surface area A:

$$I = \frac{d\Phi}{d\Omega} \tag{2.3}$$

Here, we should introduce the definition of solid angle Ω. Considering a surface point x and an arbitrary surface of projected size $\overline{A} = A \cdot cos\Theta$ on x's hemisphere (as shown in Fig. 2.1.1), the definition of solid angle is $\Omega = \frac{\overline{A}}{r^2}$ and its unit is *steradian*$[sr]$. Here, $\Theta$is the angle between the directional line Θ from x to surface A and the local normal $\mathbf{n}_x$ of A.

The last, but the most important radiometric term in computer graphics is *radiance*. It is defined as the radiant flux per unit solid angle per unit projected area

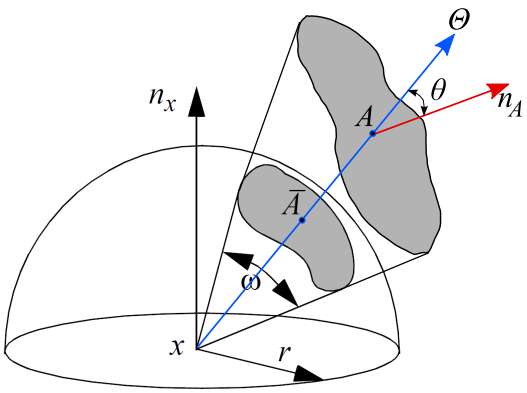


Figure 2.1: The definition of solid angle.

which is leaving from or arriving at a surface point x:

$$L(x) = \frac{d^2\Phi}{d\bar{A}d\Omega} = \frac{d^2\Phi}{\cos\Theta dAd\Omega} \tag{2.4}$$

The unit of radiance is $[W/m^2sr]$. Radiance can be interpreted as the number of photons arriving per time at a small area from a certain direction. Hence, we usually write radiance $L(x)$ for point x and direction Θ as $L(x \rightarrow \Theta)$. Radiance has the important property of being constant along a ray in empty space, therefore ti is commonly used by most rendering algorithms, such as ray tracing, GPU-based rasterization rendering, etc.

All the above mentioned quantities also generally vary with the wavelength of light. For example, the *spectral radiance* $L = dL/d$ and its unit correspondingly changes to $[W/m^3sr]$. In computer graphics, the spectral quantities are generally defined based on the basis functions decomposed form the spectrum. In most cases, the *RGB* color space and its decomposed *R*, *G*, or *B* channels are used. For example, the L_R, L_G and L_B are the spectral radiance values which correspond to *R*, *G* and *B* channels separately.

2.1.2 Bidirectional Reflectance Distribution Function

The bidirectional reflectance distribution function (BRDF) is used to describe how the incident light reflect into a continuum of directions after hitting the surface point. It is defined as the ratio of the differential reflected radiance L_r leaving current point x in direction Θ_r and the differential incident irradiance E_i arriving from direction Θ_i:

$$f_r(x_s, \Theta_i \rightarrow \Theta_r) = \frac{dL_r(x_s \rightarrow \Theta_r)}{dE_i(x_s \leftarrow \Theta_i)} = \frac{dL_r(x_s \rightarrow \Theta_r)}{L_i(x_s \leftarrow \Theta_i)\cos\Theta_i d\Omega_{\Theta_i}} \tag{2.5}$$

The geometry of BRDF is show in Fig. 2.1.2. Each direction Θ is itself pa-

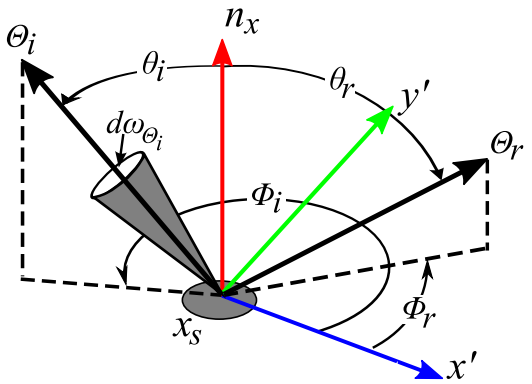


Figure 2.2: The geometry of BRDF. x', y' and n_x constitute the local coordinate system of surface position x_s.

rameterized by zenith angle Θ and azimuth angle Φ in polar coordinates. The incident direction $\Theta_i = (\Theta_i, \Phi_i)$ and reflected direction $\Theta_r = (\Theta_r, \Phi_r)$ vary over the unit hemisphere and x is the 2D position on the surface. The unit of BRDF is $[1/sr]$.

Considering the 2D position x_s, incident direction Θ_i and reflected direction Θ_r, the BRDF function $f_r(x_s, \Theta_i \rightarrow \Theta_r) = f_r(x_s, \Theta_i, \Phi_i, \Theta_r, \Phi_r)$ as a whole is 6-dimensional. If the BRDF varies for different surface position x_s, it is often called *spatially varying* BRDF. Otherwise, the current material and its BRDF are *shift-invariant* or *homogeneous*, and the current BRDF degenerates into 4-dimentional function $f_r(\Theta_i, \Phi_i, \Theta_r, \Phi_r)$. If for current position x_s the reflection changes when the incident direction is rotated around n_x, the BRDF of current material is *anisotropic*. Otherwise, the BRDF is *isotropic*, and becomes a 5-dimensional function $f_r(x_s, \Theta_i, \Theta_r, \Phi_r - \Phi_i)$.

BRDF describes the reflectance properties of surface material. Physically-correct BRDF models usually contain three important properties [Beckmann63]: (1) Positivity:

$$f_r(x_s, \Theta_i \rightarrow \Theta_r) \leq 0 \tag{2.6}$$

. Obviously, the BRDF value should not be negative. (2) Helmholtz reciprocity:

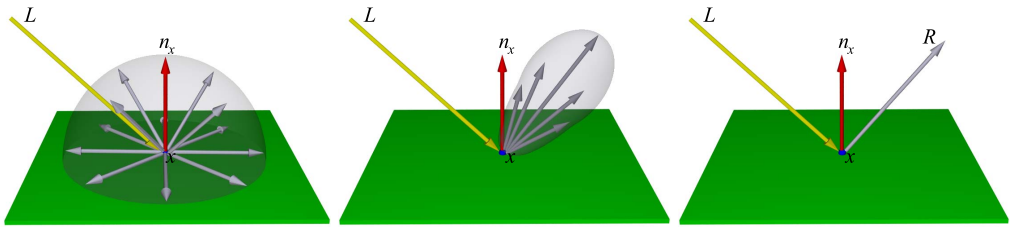
$$f_r(x_s, \Theta_i \rightarrow \Theta_r) = f_r(x_s, \Theta_r \rightarrow \Theta_i) \tag{2.7}$$

It means that the BRDF must be symmetric in Θ_i and Θ_r (3) Energy conserving:

$$\int_{\Omega^+} f_r(x_s, \Theta_i \rightarrow \Theta_r) \cos\Theta_i d\Omega_{\Theta_r} \leq 1 \quad \forall\ \Theta_i \in \Omega^+ \tag{2.8}$$

it describes the fact that real materials do not reflect more energy than what they receive. When integrating the reflected energy over the upper hemisphere at x_s the total amount of energy must be less or equal to the amount of incident energy.

Here, we would like to firstly introduce three most basic and popular BRDF models used in our rendering projects: diffuse, glossy and specular.



(a) diffuse BRDF (b) glossy BRDF (c) specular BRDF

Figure 2.3: Three most basic and popular BRDF models are illustrated. The yellow arrow represents the incident ray L, the red arrow is the local normal n_x of point x, and the grey arrow represents the reflected ray R.

Diffuse BRDF reflects incident energy evenly into all directions (Fig. 2.3 (a)). Therefore the reflected energy is the same in arbitrary direction, and there will be no highlight and view dependency for diffuse BRDF. The surface with this kind of material is usually called *diffuse* or *lambertian* surface. **Specular** BRDF reflects incident energy in single direction and works just like a mirror (Fig. 2.3 (c)). The reflected energy can only be observed in specular reflected direction, so specular BRDF contains the highlight effect and view dependency. The behavior of **Glossy** BRDF locates between diffuse and specular BRDF, and it reflect the incident energy into a range of directions (Fig.2.3 (b)). Glossy BRDF also has the highlight effect and the view dependency. These three basic BRDF models are usually the components of other advanced BRDF models, such as Blinn-Phong model [Blinn77], Lafortune model [Lafortune97], Ashikhmin-Shirley model [Ashikhmin00] and so on. Since the rendering of surface material is not the focus of this thesis, we only briefly introduce the BRDF concept and the basic BRDF models here. We refer the reader to the recent SIGGRAPH 2005 course [Lensch05] for a comprehensive and detailed introduction about the realistic materials in computer graphics.

2.1.3 Reflection and Refraction

When an incident light ray L is traveling through a medium with an index of refraction $_1$ to arrive at a perfectly-smooth planar interface with an index of refraction $_2$, the energy of this ray splits into two parts (as shown in Fig. 2.1.3).

The first part (R) is reflected off the surface and the other part (T) is refracted and transmitted into the other medium. The reflected angle Θ_1 and refracted angle Θ_2 between reflected/refracted rays and the surface normal $\mathbf{n}_x$ can be computed based on *Snell's law* [Born64]:

$$_1 \sin \Theta_1 = \ _2 \sin \Theta_2 \tag{2.9}$$

Usually the incident ray L is known. To compute the reflected ray R and the

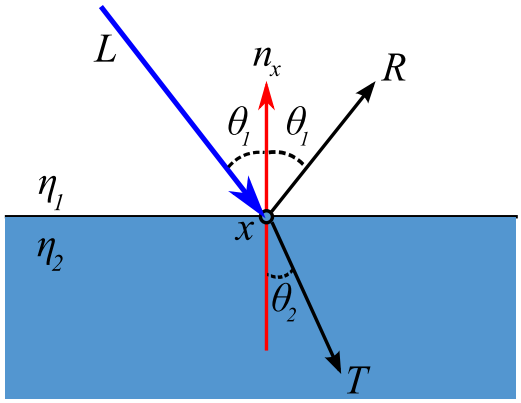


Figure 2.4: The geometry for the reflected and refracted rays at interface surface.

refracted ray T, we can derive the following formulas based on the Snell's law [Glassner95] and the geometry for ray reflection and refraction (Fig. 2.1.3).

$$R = 2\,(n_x \cdot L)\,n_x - L \tag{2.10}$$

$$T = -\frac{1}{2}(L - (n_x \cdot L)n_x) - \left(\sqrt{1 - \left(\frac{1}{2}\right)^2 (1 - (n_x \cdot L)^2)}\right) n_x \tag{2.11}$$

The value of the index of refraction can be found in most of the optics textbooks [Born64]. For example, the air is usually considered to be close to vacuum and has ≈ 1.0. The glass often has $\approx 1.5 - 1.7$.

2.1.4 Fresnel

We have already showed the formulas to compute the directions of reflected and refracted ray. Now we need to know how the energy of the incident ray splits between the two different rays.

Considering the distribution of incident energy at surface point x, physically there are three parts: absorbed by current material, reflected part and refracted part. We define the ratio of each of these three parts to incident energy to be:

absorptance (x), reflectance (x) and transmittance (x). In physically correct rendering, the sum of $(x) + (x) + (x)$ should be 1.0. In usual case, the absorption is ignored ($(x) = 0$) and we just need to consider reflectance (x) and transmittance (x).

Fresnel's equations are used to describe the reflectance and transmittance of electromagnetic waves at an interface. That is, they give the reflectance and transmittance for waves parallel and perpendicular to the plane of incidence [Born64]:

$$_{||}(x) = \frac{_2 \cos \Theta_1 - _1 \cos \Theta_2}{_2 \cos \Theta_1 + _1 \cos \Theta_2} \tag{2.12}$$

$$_{\perp}(x) = \frac{_1 \cos \Theta_1 - _2 \cos \Theta_2}{_1 \cos \Theta_1 + _2 \cos \Theta_2} \tag{2.13}$$

Here $_{||}(x)$ is for the reflected light with its wave parallel to the plane of incidence and $_{\perp}(x)$ is for reflected light with its wave perpendicular to the plane of incidence. If the incident light contains an equal mixture of parallel wave and perpendicular wave, which is the usual natural light around us, the reflectance can be computed by:

$$(x) = \frac{1}{2}\left(_{||}(x)^2 + _{\perp}(x)^2 \right) \tag{2.14}$$

the corresponding transmittance is $(x) = 1.0 - (x)$.

The effect of the Fresnel equations can be easily observed in reality, e.g. if you look straight down from above at a pool of water, you will not see very much reflected light on the surface of the pool, and can see down through the surface to the bottom of the pool. At a glancing angle (looking with your eye level with the water, from the edge of the water surface), you will see much more specularity and reflections on the water surface, and might not be able to see what's under the water. In such a case, the amount of reflectance you see on a surface depends on the viewing angle.

2.2 Rendering equation

Based on aforementioned basic concepts and terms, we now know how light received by a surface is reflected. When rendering a scene without participating media, the kernel problem is how to describe the complete light transportation from the light source to final view point. In 1986 Jim Kajiya [Kajiya86] and David Immel [Immel86] simultaneously introduced the Rendering Equation to computer graphics. The rendering equation is an integral equation describing the radiance equilibrium leaving a surface point x to reach view point v as the sum of emitted

and reflected radiance at that point:

$$L_v(x \to \Theta_o) = L_e(x \to \Theta_o) + \int_{\Omega_x^+} f_r(x, \Theta_i' \leftrightarrow \Theta_o') \cdot L(x \leftarrow \Theta_i) \cdot (\mathbf{n}_x \cdot \Theta_i) d\Omega_{\Theta_i} \tag{2.15}$$

In this equation, L_e is the emitted light, f_r is the BRDF, $\mathbf{n}_x$ is the normal at x, Θ_i and Θ_o are the global incident light and viewing directions, and Θ_i' and Θ_o' are light and view in local coordinates. This rendering equation describes, that the radiance leaving point x towards direction Θ_o (to view point v) equals the radiance emitted from the x in direction Θ_o, in case x is an emitter itself, plus all the reflected radiance which can be computed by integrating all the incident radiance $L(x \leftarrow \Theta_i)$ (over the hemisphere Ω_x^+ of x) scaled by the BRDF $f_r(x, \Theta_i' \leftrightarrow \Theta_o')$ and the cosine weighting term $\mathbf{n}_x \cdot \Theta_i$.

If we simplify eq.2.15 as following:

$$L_v = L_e + TL_{in} \tag{2.16}$$

Here, L_{in} represents the incident radiance $L(x \leftarrow \Theta_i)$ and T represents the integral operator in eq.2.15. Note, L_{in} can be from direct lighting of light source or indirect lighting bounced from scene elements. Considering the complete light transport in scene, eq.2.16 can be recursively expanded into:

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

This expansion is called *Neumann series* and sums all the light contributions through $0, 1, 2, \ldots$ times reflection. Such a kind of multiple-bounces light transportation is so-called global illumination.

The rendering equation forms the basis for computing the light transport in a scene model without participating media. We notice that eq.2.15 performs integration over differential solid angle $d\Omega_{\Theta_i}$. In real rendering problems, we usually parameterize the equation over differential surface instead for convenience:

$$L_v(x \to \Theta_o) = L_e(x \to \Theta_o) + \int_{x' \in \Omega^+} f_r(x, x' \leftrightarrow \Theta_o') \cdot L(x \leftarrow x') \cdot G(x, x') dA_{x'} \tag{2.18}$$

Compared with eq.2.15, here x' is the surface element in the hemisphere of x. The previous cosine weighting term is replaced by $G(x, x')$. It is the *geometric term* which is responsible for the geometric arrangement of both differential surfaces taking the distance between each other, the orientation of each surface, as well as mutual visibility between them into account:

$$G(x, x') = \frac{\cos\Theta_{x'} \cos\Theta_x}{||x - x'||} V(x, x') \tag{2.19}$$

The visibility function $V(x,x')$ represents whether there exists occlusion between x and x' and in other words, whether x and x' can see each other. $V(x,x')$ is a piecewise binary function defined as follows:

$$V(x,x') = \begin{cases} 1 & if \quad x \quad and \quad x' \quad is \quad mutually \quad visible \\ 0 & else \end{cases} \tag{2.20}$$

The visibility term in the rendering equation illustrates how the shadow is originated from. Shadow is usually the most important visual clue for the perception of rendering. However, the evaluation of visibility function is usually time-consuming, e.g. relying on ray casting, so how to improve the shadow generation has all along been an important topic in rendering research.

2.3 Rendering Techniques

All existing rendering techniques are trying to accurately or approximately solve the rendering equation. In this section, we will briefly introduce two existing accurate rendering methods: *ray tracing* and *radiosity*, and two approximate rendering methods: *precomputed radiance transfer* and *ambient occlusion*.

2.3.1 Ray Tracing methods

Ray tracing methods generate a final image by tracing light rays from each pixel on the image plane into a given scene. During the light transportation, the intersections between the rays and scene elements will be computed. At each intersection point, the exit radiance from light-surface interaction is computed based on the rendering equation. Actually, the ray tracing start inversely from the screen pixel, and therefore the whole process is implemented in recursive manner. That's why ray tracing is usually so-called *inverse rendering* method and also ray tracing is an image-space algorithm.

The common bottleneck of ray tracing methods is the intersection computation. The hierarchical data structures [Glassner91], such as kd-tree, bounding volume hierarchy (BVH), etc, are usually adopted to organize the input scene and hence accelerate the intersection query. To handle global illumination effects, such as soft shadow from area light source, diffuse/glossy inter-reflection, etc, the Monte Carlo algorithm is applied. It is because we need to randomly decide the a bunch of reflection/refraction ray directions so as to approach the correct result of rendering equation. Although conceptually ray tracing can accurately solve the rendering equation, the time-consuming for large amount of ray sampling usually limits the performance of ray tracing in reality. We will discuss more state-of-the-art ray tracing methods in Chapter. 3.

2.3.2 Radiosity methods

Radiosity [Goral84, Cohen93] is an application of the finite element method to accurately solve the rendering equation for scenes. It subdivide the input scene geometry into patches and treat the patches as either *emitters* or *receivers*. As stated before, the rendering equation describes that the illumination process is in equilibrium. Therefore, with the consideration of visibility, the energy exchange between emitter patches and receiver patches can continue until the solution reaches its convergence. The visibility determination in radiosity is usually relying on a simplified ray tracing method: ray casting [Roth82], which has the same performance limitation as ray tracing. Such an energy exchange process turns out to be equivalent to solving a system of linear equations. The final result is usually stored at each vertex of the scene, hence radiosity can be regarded as a kind of object-space algorithm.

Early radiosity methods only account for diffuse receivers but can be extended to support glossy receivers [Immel86, Cohen93]. The subdivision scheme for radiosity methods are usually tricky. Basic radiosity has trouble resolving sudden changes in visibility (e.g., hard-edged shadows) because coarse, regular discretization of input scene (into piecewise constant elements) corresponds to a low-pass box filter of the spatial domain. The discontinuity meshing method [Lischinski92] uses knowledge of visibility events to generate a more intelligent discretization. We will also discuss more about radiosity methods in the following Chapter. 3. Note, radiosity solve the rendering equation and get the result for each vertex. To finally display the rendering result, we still require another displaying algorithm to achieve it.

2.3.3 Precomputed Radiance Transfer

Precomputed Radiance Transfer (PRT) [Sloan02] is aiming at rendering a scene in real-time with complex light interactions being precomputed to save time. The light transport at scene point from incident radiance to output radiance is a linear transformation. In essence, PRT can compute the illumination of a scene point as a linear combination of incident lighting. Prior of that, in precomputation stage a set of efficient basis functions must be used to approximately encode this data, such as Spherical harmonics [Sloan02] or wavelets [Ng03]. Finally, the rendering equation can be solved by computing the linear combination of the coefficients in basis function space.

PRT can real-time handle environment and area lighting and glossy as well as diffuse objects. Further, its rendering quality is very good. However, since the projections into basis function usually only approximate original signal, the rendering result of PRT is usually an approximate solution of rendering equation.

The biggest problem of PRT comes from the precomputation. The precomputation limits the input scenes which PRT method can handle are static or only contains rigid transformations. If the scene start to deform, the whole precomputed info will be invalid. However, the fully-dynamic scene is very common in usual rendering applications, such as computer game, film special effects post-production, etc. To overcome this limitation and develop global illumination rendering for fully-dynamic scene is also another strong motivation of our research works in this thesis. More PRT related works will be discussed in Chapter. 3.

2.3.4 Ambient Occlusion

Ambient occlusion [Zhukov98] captures a subset of global illumination effects, by computing for each point of the surface the amount of incoming light from all directions and considering potential occlusion by neighboring geometry. Ambient occlusion is most often calculated by casting rays in every direction from the current surface point. Essentially, the result of ambient occlusion is just a ratio number which equals to the number of potential occluded rays divided by the number of overall out-shooting rays. Obviously, it is a very crude approximation to full solution of rendering equation.

Ambient occlusion is related to accessibility shading [Miller94], which determines appearance based on how easy it is for a surface to be touched by various elements (e.g., dirt, light, etc.). It always works under the environmental lighting (sky lighting) condition. Although it is a crude approximation, the ambient occlusion shading model has the nice property of offering a better perception of the 3d shape of the displayed objects. This was shown in [Langer00] where the authors report the results of perceptual experiments showing that depth discrimination under diffuse uniform sky lighting is superior to that predicted by a direct lighting model. Further, the computation cost of ambient occlusion is cheap compared with the full solution of complete global illumination. Due to its relative simplicity and efficiency, the ambient occlusion has been very popularized in film production animation and become a standard tool for computer graphics lighting in motion pictures.

2.3.5 Accurate vs. Approximate

To achieve complete solution of rendering equation for global illumination effects are very time-consuming and expensive. For the accurate rendering techniques, such ray tracing and radiosity, the huge amount of data samplings usually slow the performance of rendering and make the real-time target very difficult. However, from the perception side, the final target of our rendering is to generate an image which can fulfill the perception of human being. One interesting

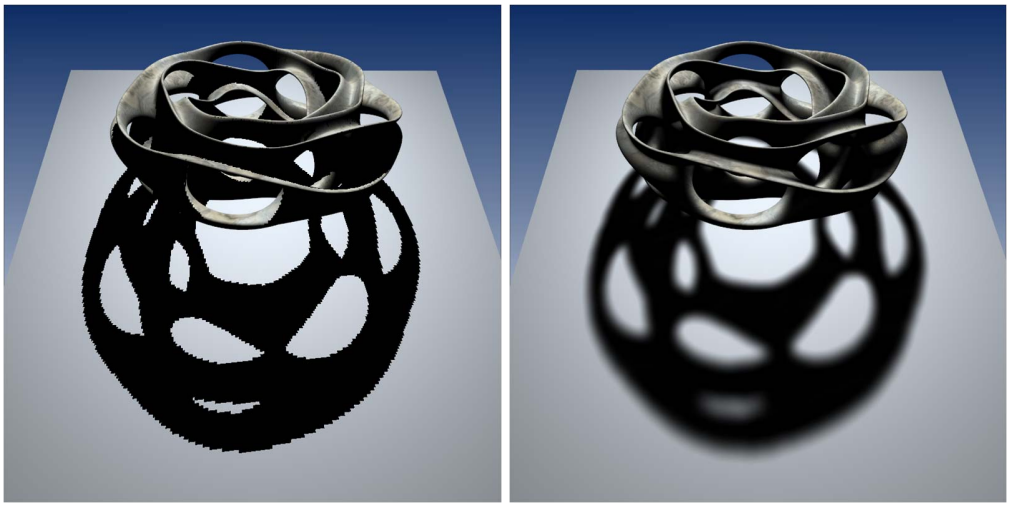
question is: whether the complete solution of rendering equation is necessary to convince people the global illumination effect is correct? Lots of recent research works [Ramasubramanian99, Myszkowski01]have proved that in lots of scenarios, the fully physically-correct global illumination is not necessary to convince human's perception. Usually the visually pleasing approximate global illumination rendering results would be enough. That's also why ambient occlusion becomes so successful in real application.

We have three targets in our global illumination effects rendering research works: (1) visually pleasing rendering quality (2) real-time rendering performance (2) handling fully-dynamic input scenes without any precomputation. Motivated by perception-guided global illumination rendering, we position the rendering results for our research are close to (not fully) physically-correct but visually very pleasing. The visibility computation is usually the bottleneck of global illumination. Therefore, in Chapter. 4 we introduce a kind of *implicit visibility* method to accelerate the visibility computation of hierarchical radiosity method and achieve interactive performance for fully-dynamic scenes. As stated before, global illumination effects include realistic soft shadow, indirect lighting (interreflection), caustics and so on. Among of them, the soft shadow caused by visibility determination plays very crucial role for the human perception of the 3D world [Hasenfratz03a]. In Chapter. 5,6,7, we sequentially introduce several visually pleasing soft shadow algorithms and their applications in global illumination rendering. Since all of our soft shadow algorithms are based on shadow mapping [Williams78], in following section we will introduce the background of shadow mapping and also the theory of percentage-closer soft shadow mapping.

2.4 Visually Pleasing Soft Shadow Mapping

The simplest scene setting for shadow casting will include one light source, one occluder and one receiver. As mentioned before, shadow casting is crucial for the human perception of the 3D world [Wanger92]: (1) Shadows help to understand relative object position and size in a scene. (2) Shadows can also help us understanding the geometry of a complex receiver. (3) Finally, shadows provide useful visual cues that help in understanding the geometry of a complex occluder. Based on the hard shadow of the complex occluder (Fig. 2.5 (a)), you can easily feel all the three above points about how shadow affects our perception of the scene.

There are two major techniques for generating shadow: object-based *shadow volume* [Crow77] and image-based *shadow map* [Williams78]. Both techniques can be accelerated using the evolving graphics hardware. Since all the soft shadow methods in this thesis are based on shadow mapping theory, we will not go further into shadow volume methods. For more details about shadow volume methods,



(a) Hard shadow (b) Soft-edged shadow

Figure 2.5: The illustration of hard shadow and soft-edged shadow.

we refer the reader to Woo et al. [Woo90], Hasenfratz et al. [Hasenfratz03b], and to a recent course [Eisemann09] for a detailed overview. In following subsection, we will firstly illustrate the basic theory of *shadow mapping*

2.4.1 The Basic Theory of Shadow Mapping

Shadow mapping is a kind of image-based shadow algorithm. The basic shadow mapping is usually used to generate the hard shadow. As shown in the Fig. 2.6, a simple scene consists of a point light source, an occluder plane and a receiver plane. The vertical bar on the right in Fig. 2.6 utilize the color to represent the depth values in scene. *ZNear* and *ZFar* separately represents the minimal and maximum value of all the depth values which are normalized into [0, 1]. We will firstly create a 2d texture buffer which is so-called shadow map or depth map to record the smallest depth values from the light source point, as shown in Fig. 2.6. Then during rendering, we will compare the depth value of each point to its corresponding minimal depth value stored in shadow map, and the comparison results will determine the shadow value of current point. As shown in Fig. 2.6, point a_1 is lit and point a_2 is in shadow.

The implementation of shadow mapping usually contains two steps: (1) Render the shadow map (depth map) and (2) Render the scene using the shadow map. The first step renders the whole scene from the light's point of view. From this step, the shadow map is extracted and saved in the video memory of GPU in 2d texture format. The second step is to draw the scene from the usual camera view-

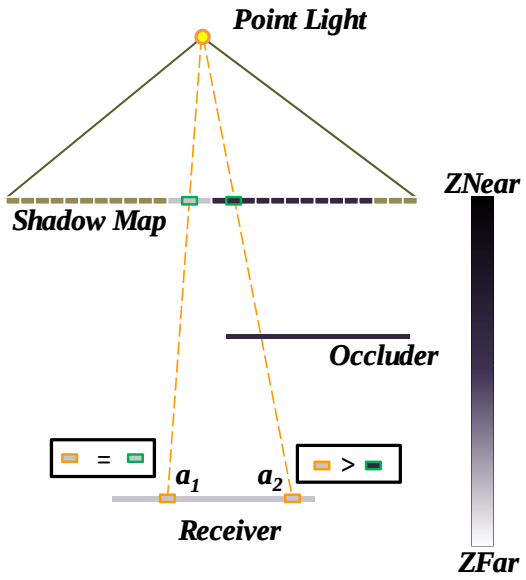
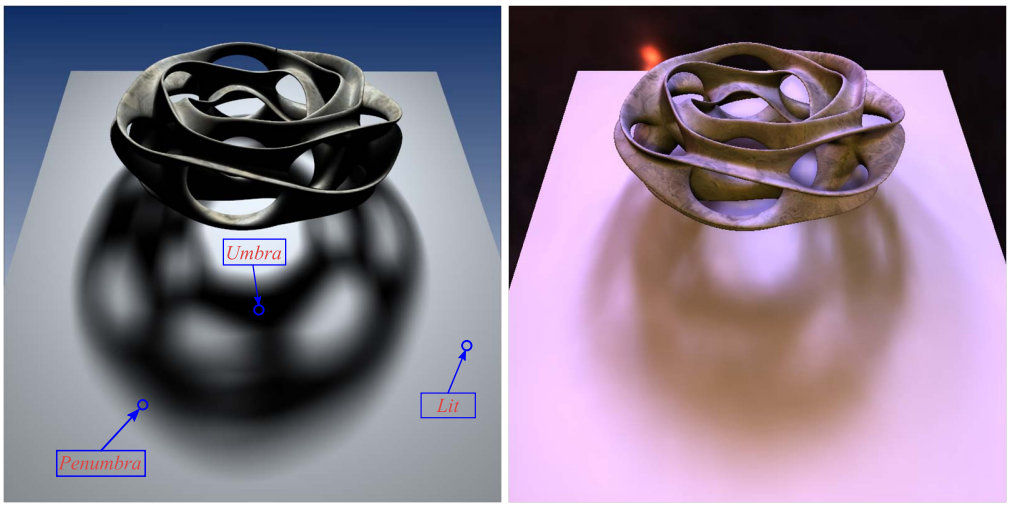


Figure 2.6: The illustration of shadow mapping theory.

point, applying the shadow map. This process has three major components, the first is to find the corresponding coordinates of the object as seen from the light, the second is the depth test which compares that the depth value of current point against the depth map, and finally, once accomplished, the scene point must be drawn either in shadow or in lit. For the details of shadow mapping implementation, we refer the reader to check the code samples at [NVIDIA05].

One of the key disadvantages of shadow mapping is spatial aliasing, which is due to that the sampling rate (resolution) of shadow map does not match the resolution of current screen pixels [Aila04]. A simple way to overcome this limitation is to increase the shadow map size, but due to memory, computational or hardware constraints, it is not always possible. Commonly used techniques for real-time shadow mapping have been developed to circumvent this limitation. These include Cascaded Shadow Maps [NVIDIA08], Trapezoidal Shadow Maps [Martin04], Light Space Perspective Shadow Maps [Wimmer04] or Parallel-Split Shadow Maps [Zhang06]. Also notable is that generated shadows, even if aliasing free, have hard edges, which is not always desirable. In order to emulate real world soft shadows, several solutions have been developed, either by doing several lookups on the shadow map or creating pre-filterable non-standard shadow maps to emulate the soft-edged shadows (Fig. 2.5 (b)). Notable examples of these are Percentage Closer Filtering [Reeves87], Variance Shadow Maps [Donnelly06a], Convolution Shadow Maps [Annen07] or Exponential Shadow Maps [Annen08b].

2.4.2 Percentage Closer Soft Shadow Mapping



(a) Soft shadow under single area light (b) Soft shadow under environmental lighting

Figure 2.7: The illustration of soft shadows.

In this section, we will introduce the Percentage Closer Soft Shadow Mapping method (PCSS) [Fernando05a] which introduce a kind of visually-pleasing soft shadow rendering framework. Our soft shadow rendering methods in Chapter. 5,6,7 are all based on this framework. Let's firstly introduce the basic concept of soft shadow. As shown in Fig. 2.7 (a), the soft shadow from single area light usually consists of three different parts: (1) if the current scene point is fully occluded and can not see any part of the light source, then it is fully dark and is in *umbra*. (2) if the current scene point is partially occluded and can see part of the light source, then it is in *penumbra*. (3) if the current scene point is not occluded at all and can see the whole light source, it is in *lit*. Soft shadow plays very important roles in the photo-realistic rendering, and lots of algorithms [Hasenfratz03b] have been propose to deal with it.

Before introducing the PCSS method, let's firstly check what is the percentage closer filtering (PCF) for hard shadow mapping [Reeves87]. As shown in Fig. 2.8, compared with standard shadow mapping, the PCF method sample the shadow map and do the depth comparison multiple times. The final shadow value for current point x is essentially a linear combination of the results from all the shadow comparisons. How many shadow comparisons will be done for each scene point? Usually a fix-sized *filter kernel* will be assigned for each scene point, and the number of texels in the filter kernel determines the number of shadow map sampling and depth comparison. Based on PCF, we can generate visually pleasing soft-edged shadow. However, soft-edged shadow is not soft shadow, since

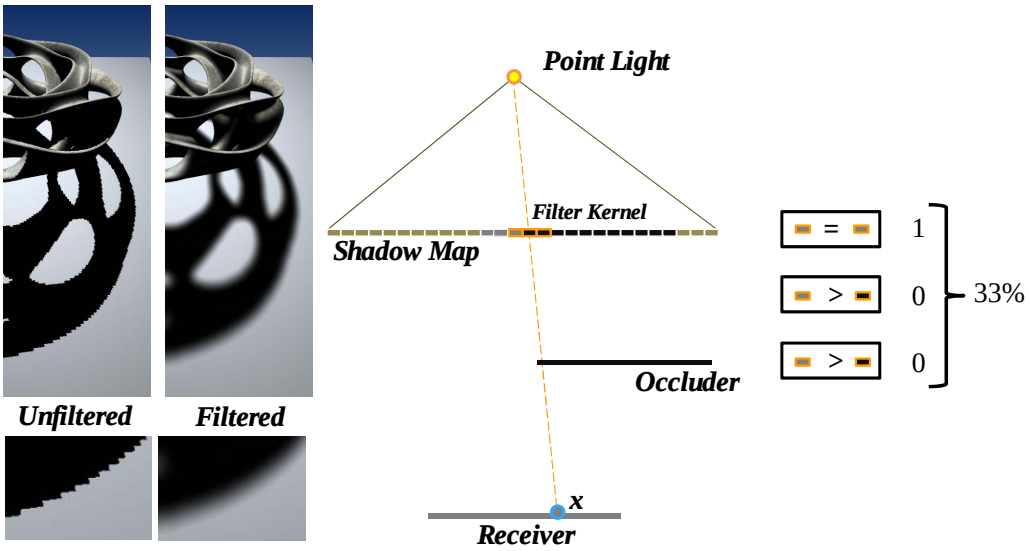


Figure 2.8: The illustration of percentage closer filtering shadow mapping.

the size of filter kernel is isotropic for every scene point. An interesting observation here is the normal texture filtering techniques, such as mipmap, summed area table (SAT) [Crow84], etc, cannot be directly applied for PCF. The reason is the depth comparison has to been conducted before the filtering step. Therefore, only the brute-force point sampling is applied in normal PCF shadow mapping and slows its performance for large filter kernel. It is also the motivation of pre-filtering shadow mapping methods [Donnelly06a, Annen07, Annen08b]. Our visually pleasing soft shadow mapping methods are all based on pre-filtering theory of shadow mapping, and we will discuss it more in later chapters.

Although PCSS method [Fernando05a] is based on percentage-closer filtering, its target is rendering visually pleasing soft shadow, not soft-edged shadow. The major difference between soft shadow and soft-edged shadow is the penumbra. As stated before, the sizes of the filter kernel for different scene points are the same when generating soft-edged shadow. The penumbra in soft-edged shadow appears isotropic in different parts of the scene (Fig. 2.5 (b)). Compared with it, soft shadow provides valuable cues about the relationships between objects, becoming sharper as objects contact each other and more blurry (softer) the further they are apart (Fig. 2.7 (a)). Therefore, the penumbra in soft shadow should appear anisotropic and the filter kernel for different scene points are usually different.

The PCSS method mainly contains two steps, and we show the first step in Fig. 2.9. The simple scene consists of one area light with size S, one occluder object and one receiver. In this step, the average blocker depth d_b for the current point x is computed by averaging all depth values of *blocker texels* within an

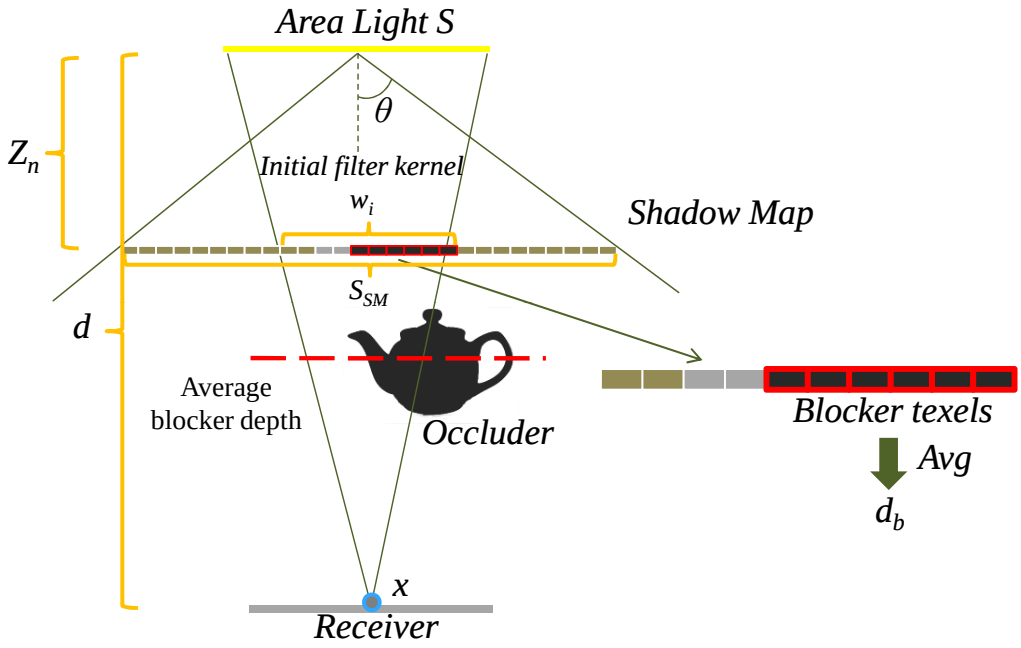


Figure 2.9: The computation of average blocker depth in PCSS.

initial filter kernel w_i. The blocker texel is the texel the depth value of which is smaller than x's depth. Usually we will initialize the position of shadow map in object space to be Z_n, which is the near plane of the camera setting. The size of w_i can be computed by $\frac{d-Z_n}{d} \cdot S \Big/ S_{SM}$. Here, S is the size of area light, and the $\frac{d-Z_n}{d} \cdot S$ part can be easily derived based on similar triangles. The denominator S_{SM} represents the size of current shadow map in object space. It can be computed as $2 \cdot Z_n \tan \cdot \Theta$ and the Θ is the half of the camera's FOV angle for generating shadow map. Divided by S_{SM},the kernel size w_i will be normalized into [0, 1] so that it can be applied in texture coordinate space. Apparently, for different scene points, the average blocker depths d_b will be different.

In the second step as shown in Fig. 2.10, based on average blocker depth d_b, we can compute the size of filter kernel w_b of PCF for x. The computation is similar as previous formula of w_i: $w_b = \frac{d-d_b}{d} \cdot S \Big/ S_{SM}$ Note, currently we need to shift the position of shadow map in object space from Z_n to d_b. Hence, the formula of S_{SM} is $2 \cdot d_b \cdot \tan \Theta$. Then we can do the normal PCF with w_b for current scene point to compute its soft shadow value. Since the w_b is different for different point, the penumbra will appear anisotropic and reflect valuable cues about the relationships between scene objects.

One assumption in the derivation of PCSS is that the occluders/receivers are all planar and in parallel. Although it is an approximation of scene, the PCSS

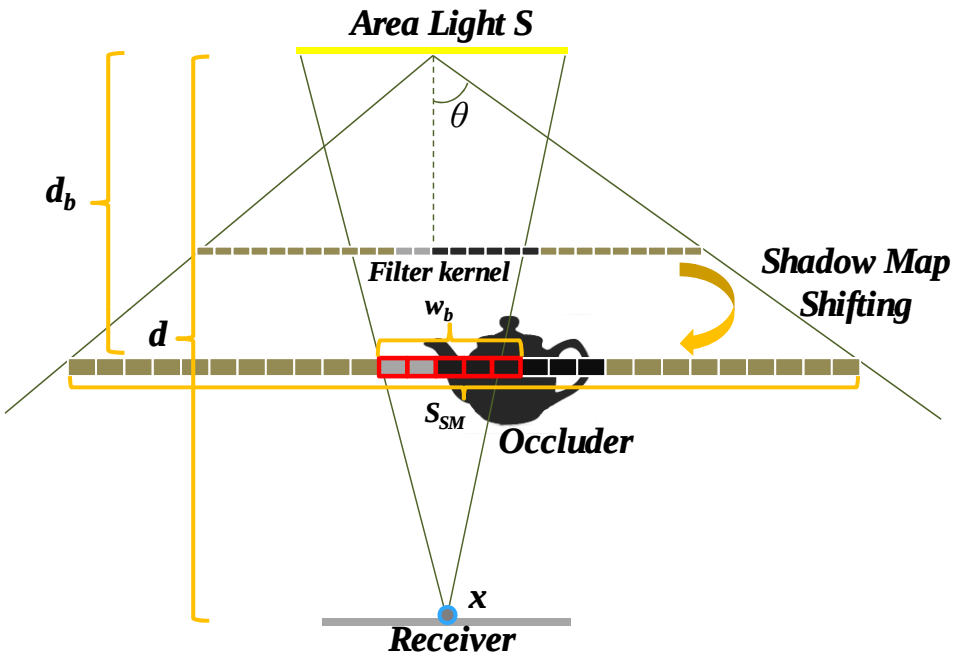


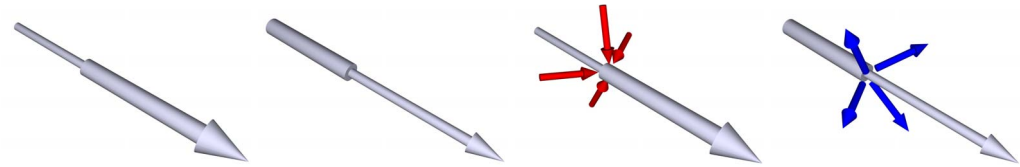
Figure 2.10: The PCF-based soft shadow computation in PCSS.

method usually achieves visually plausible quality and real-time performance for small light source. Moreover, the implementation of PCSS method only incurs shader modification and is easy to be integrated into existing rendering system. As a result, it has become quite popular. However, when dealing with medium/large area lights, the performance of PCSS method becomes slow since it depends on brute-force PCF point sampling. The algorithmic pipeline of the PCSS method can be regarded as a general soft shadow mapping framework based on the planarity assumption. In Chapter. 5 and 7, we will introduce two pre-filtering soft shadow mapping methods which are based on the PCSS framework. Our methods can tackle the performance problem of PCSS based on pre-filtering to achieve both visually plausible quality and high speed.

Considering the rendering artifacts, the planar assumption of PCSS is some kind of "single blocker depth assumption" which essentially flattens blockers. When the light size become bigger, this assumption is more likely to be violated and more specifically, umbra tends to be underestimated. From another side, PCSS only generates one shadow map from the center of light source. When using a single depth map to deal with occluders which contain a high depth range, the single silhouette artifacts [Assarsson03] can be introduced. We will discuss the artifacts more in later chapters.

2.5 Participating Media Rendering

Participating media is common in realistic world, like fog, smoke, etc. Rendering participating media is important for lots of applications, ranging from entertainment and virtual reality to simulation systems (flying simulators) and safety analyses (driving conditions). As stated before, the rendering equation (eq.2.15) only computes the global illumination effect when there is no participating media in the scene and it does not count the energy transportation in the media. In this section, we will briefly introduce the basic theory for participating media rendering.



(a) Emission (b) Absorption (c) In-scattering (d) Out-scattering

Figure 2.11: Interaction of light in a participating medium.

As radiation travels through a participating medium it undergoes three kinds of phenomena: emission, absorption, scattering (Fig. 2.11). Emission is a process by which a particle in current media converts from a higher energy state into a lower one through a photon, resulting in the production of radiant energy. Absorption consists of the transformation of radiant energy into other energy forms. For a differential distance dx, the relative reduction of radiance is given by $\kappa_a(x)dx$, $\kappa_a(x)$ being the *coefficient of absorption* of the medium at point x. Scattering means a change in the radiant propagation direction. It is generally divided into out-scattering and in-scattering depends on the radiant energy is reduced or increased.

Out-scattering reduces the radiance in the particular direction along dx by the factor $\kappa_s(x)dx$, $\kappa_s(x)$ being the *scattering coefficient*. Mathematically the reduction of radiance during the transportation in media is expressed as $dL(x) = -\kappa_t(x)L(x)dx$, where $\kappa_t = \kappa_a + \kappa_s$ is the *extinction coefficient*. The solution of this differential equation is Beer's law [Ingle88]:

$$L(x) = L(x_0)e^{-\int_{x_0}^{x} \kappa_t(u)du} = L(x_0)\ \tau(x_0,x) \tag{2.21}$$

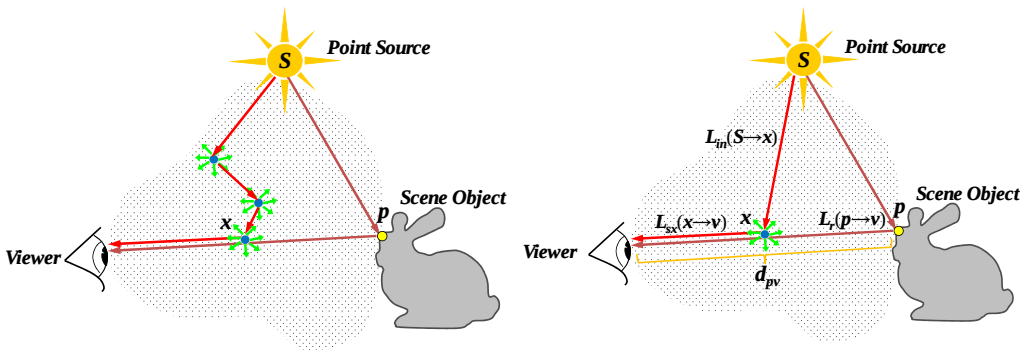
$\tau(x_0,x)$ is so-called *transmittance* from x_0 to x. The *scattering albedo* is defined as $\Omega(x) = \frac{\kappa_s}{\kappa_t}$ and represents the ratio of scattering in the whole extinction. Note that Beer's law simply models the reduction of radiance due to out-scattering and absorption. In the contrary, the in-scattering enhance the radiance along the propagation direction. Similar as radiance extinction, to calculate the increase of the radiance, both emission and in-scattering have to be taken into account.

The spatial distribution of the scattered radiance at a spacial point x is modeled by the phase function $p(x, \Omega_o, \Omega_i)$. The phase function has the physical interpretation of being the scattered intensity in direction Ω_o, divided by the intensity that would be scattered in that direction if the scattering were isotropic (i.e. independent of the direction). Phase functions in Computer Graphics are usually symmetric around the incident direction Ω_i, so they can be parameterized by the angle Θ between the incoming and outgoing direction. Different phase functions have been proposed to model different media. The simplest phase function is the isotropic one (constant) and represents the counterpart of the diffuse BRDF for participating media. Mie phase functions are generally used for scattering where the size of the media particles is comparable to the wavelength of light. It is applied to many meteorological optics phenomena like the scattering by particles responsible for the polluted sky, haze and clouds. Mie phase functions are generally complex and heavily depend on the particles' size and conductivity. There are several approximations to Mie phase functions and one of them is Henyey-Greenstein (HG) phase function:

$$p(\Theta) = \frac{1}{4} \cdot \frac{1-g^2}{[1+g^2-2g\cos\Theta]^{3/2}} \tag{2.22}$$

The HG phase function, by the variation of one parameter, $1 \leq g \leq 1$, ranges from backscattering through isotropic scattering to forward scattering.

2.5.1 Transport Equation in Single Scattering Media



(a) Multiple scattering (b) Single scattering

Figure 2.12: Schematic representations for the single and multiple scattering cases.

After introducing the background and basic concept of participating media, we will move further to the *transport equation* in this section, which is the rendering

solution for participating media. The complete discussion of transport equation is long and tedious, and we would refer the reader to [Cerezo05] for detailed formulas and explanations. The scattering dominates the cost of participating media rendering. In lots of real medias, the scattering radiance usually scatters several times before finally arriving at view point. It is so-called *multiple scattering* case, as shown in Fig. 2.12 (a). The multiple scattering effect is very complex and time-consuming for rendering since lots of random direction samplings will be involved. When the participating medium is optically thin (i.e. the transmittance through the entire medium is nearly one) or has low albedo, then the source radiance can be simplified to ignore multiple scattering within the medium. We can assume scattering radiance only scatters one time before arriving at view point, and it is so-called *single scattering* case (Fig. 2.12 (b)). Here, we will only illustrate the transport equation in single-scattering case.

As shown in Fig. 2.12 (b), the scene setting contains a point light source S, a scene object and a participating media region. To compute the final radiance $L(p \rightarrow v)$ arriving at view point v from point p, two parts of radiant energies should be considered:

$$L(p \rightarrow v) = L_r(p \rightarrow v) \quad (p,v) + L_s(p \rightarrow v) \tag{2.23}$$

The first part is the standard reflected radiance $L_r(p \rightarrow v)$ attenuated by transmittance (p,v). $L_r(p \rightarrow v)$ can be computed using rendering equation (Eq.2.15). The second part is the in-scattering radiance $L_s(p \rightarrow v)$ which enhance the radiance at every point x during the transport path $\overrightarrow{pv}$. For arbitrary point x, the in-scattering radiance $L_{sx}$is:

$$L_{sx}(x \rightarrow v) = L_{in}(S \rightarrow x)\Omega(x)p(x, x \rightarrow v, S \rightarrow x) \quad (x,v) \tag{2.24}$$

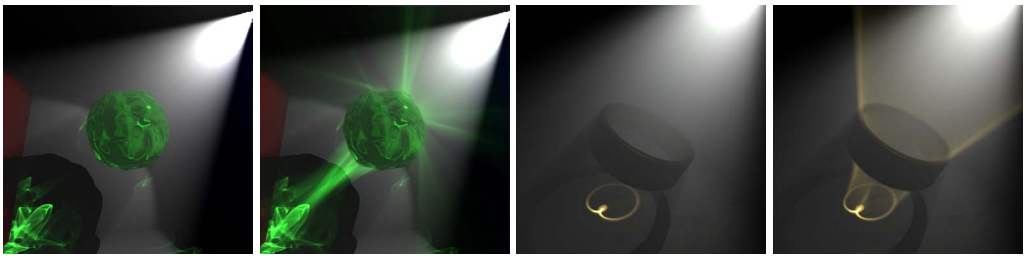
The $\Omega(x)$ is the scattering albedo, which tells us how much energy is scattered compared to be absorbed. The $p(x, x \rightarrow v, S \rightarrow x)$ is the phase function at point x and (x,v) is the transmittance from x to v. Since the in-scattering radiance comes from all the points on the path $\overrightarrow{pv}$ with distance d_{pv}, the overall in-scattering $L_s(p \rightarrow v)$ is:

$$L_s(p \rightarrow v) = \int_{d_{pv}} L_{in}(S \rightarrow x)\Omega(x)p(x, x \rightarrow v, S \rightarrow x) \quad (x,v)dx \tag{2.25}$$

2.6 Caustics

The bright patterns of light focused via reflective or refractive objects onto diffuse (matte) surfaces are called *surface caustics*. Surface caustics provides some of

the most spectacular patterns of light in nature. An example is the caustic formed as light shines through a glass of wine onto a table. In Fig. 2.13 (a) and (c), we show the examples of surface caustics through refraction and reflection. Surface



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

Figure 2.13: Illustration of surface and volume caustics under reflection and refraction.

caustics can be rendered by ray tracing the possible paths of the light beam through the glass, accounting for the refraction and reflection. Photon mapping [Jensen96] is one implementation of this.

Considering ray transport in participating media, light first interacting with a specular surface and subsequently being scattered inside a participating medium to generate some kind of intricate illumination patterns. This kind of beautiful illumination patterns in participating media are so-callled *volume caustics*. In Fig. 2.13 (b) and (d), we show the volume caustics effects through refraction and reflection. Volume caustics can also be simulated by volumetric photon mapping [Jensen98], but it is computationally expensive and impossible for interactive applications. In Chapter. 8, we will introduce a novel interactive volume caustics rendering method for single scattering participating media.

2.7 Image Displaying Solutions

Most of the aforementioned rendering techniques solve the rendering equation (e.g. radiosity) in object space, and still needs an additional rendering step to convert the solutions in 3D object space into the final 2D image. There are usually two major image display solutions: one is *ray tracing* and the other is *rasterization*.

Ray tracing itself is a kind of image-based rendering technique. It starts the rendering process inversely to trace the ray from the camera through each pixel on the view plane into the scene. During the transportation, each ray will intersects with the scene geometry. At each intersection point, the stored solution of rendering equation will be queried and displayed in current pixel. E.g., in a diffuse-only radiosity algorithm, the last step is usually relying on ray tracing to

firstly look up the stored radiosities at the vertices of the intersected patch, then compute the bilinearly-filtered result and finally convert it to exit radiance. As intersecting all polygons in a scene is time-consuming, researchers have designed various forms of hierarchical acceleration structures to improve the performance of intersection test. However, most of these structure is no suited for dynamic objects and requires a rebuild process whenever objects change their position or shape. Therefore, the performance of image displaying using ray tracing is the major issue and prevents its wide applications in interactive/real-time scenarios.

Rasterization [Catmull74] is based on sorting technique called the z-buffer and operates in a different way than ray tracing. Rasterization iterates over all the primitives and renders them into a so-called *framebuffer* based on the current camera settings. First a view matrix is applied to every vertex in the scene to transform all objects into camera space. Then the projection matrix transformation projects all polygons from 3D camera space into 2D screen-space. A scan-line algorithm then processes each polygon and computes its coverage on raster grid (pixels). For each pixel, its radiance value and depth value are computed by linearly interpolating the lighting results and depth values from the vertices. The radiance value is stored in the framebuffer and the depth value is stored in a new buffer so-called *zbuffer*. Before the final display, each pixel will compare its depth value with what has already stored at this pixel position in zbuffer to determine its spacial relationship with the old content. Only when it is in front of the old content, its radiance value will be finally displayed. Otherwise, the current pixel will be discarded. All the aforementioned global illumination rendering techniques can utilize rasterization as the displaying solution. The rasterization pipeline has become the standard rendering pipeline in current graphics hardware. Therefore, it is currently the most efficient solution for displaying. Furthermore, with the evolvement of recent graphics hardware, the rendering pipeline has become fully programmable and very friendly for algorithm development. All of our methods in this thesis are developed based on programmable graphics hardware and we will introduce it in next section.

2.7.1 Programmable Hardware Accelerated Rendering Pipeline

In August 1999 the first graphics processing unit (GPU) was introduced to the consumer level hardware market. It integrates the entire graphics pipeline in one graphics chip and supports user programmability for some stages. After this, the *programmable function pipeline* has been widely supported in graphics hardware to replace the previous *fixed function pipeline*. Currently the newest GPU has become very flexible and friendly in programmability.

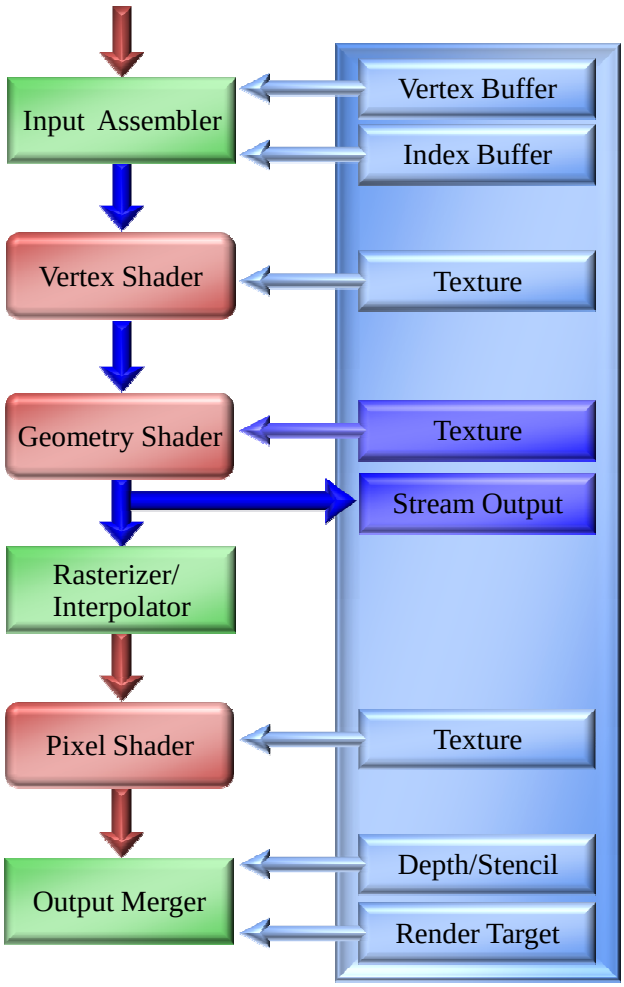


Figure 2.14: The rendering pipeline in Direct3D 10.

GPU is usually accessed via a graphics API, such OpenGL [Segal99] or Direct3D [Mic00]. In all of our projects we make use of Direct3D only but OpenGL could be used also since both APIs provide the same functionality. The newest version of Direct3D is Direct3D 11 which is released in October 2009. However, all the projects in this thesis have been developed using Direct3D 10, and we did not get involved any new features in Direct3D 11. Therefore, the following introduction will be based on Direct 10 [Blythe06]. The graphics rendering pipeline defined in Direct3D 10 is shown in Fig. 2.14. Overall, the first three stages *input assembler* (IA), *vertex shader* (VS) and *geometry shader* (GS) process the input scene geometry, so can be classified into *geometry processing*. The following *rasterization* stage will convert the 3D geometry into 2D screen pixels. The *pixel shader* (PS) and *output merger* (OM) stages then will do the *pixel process-*

ing and generate the final framebuffer result. Here, a ***shader*** is defined to be a set of software instructions which is used primarily to calculate rendering effects on graphics hardware with a high degree of flexibility. Vertex shader, geometry shader and pixel shader are all such kind of pipeline stages which contain software instructions to achieve full programmability.

Geometry Processing Input vertex and index buffers are streamed into input assembler. It read primitive data (points, lines and/or triangles) from user-filled buffers and assemble the data into primitives that will be used by the other pipeline stages. The IA stage can assemble vertices into several different primitive types (such as line lists, triangle strips, or primitives with adjacency).

Then all the vertices are sent to vertex shader where all vertex related operations can be programmed to take place. Vertex shader that performs transform and lighting (T&L), operations and a post T&L stage. T&L includes for example model/view transformations, texture coordinates assignment, and lighting. This unit was the first to allow the user to replace fixed functionality by customized shader programs. It further supports vertex texture access. After T&L, vertex shader will pursue some more functionalities including perspective correction, viewport mapping, and clipping.

Geometry shader is a new feature in Direct3D 10 and it allows manipulation of meshes on a per-primitive basis. Instead of running a computation on each vertex individually, there is the option to operate on a per-primitive basis. With this, the input vertices can be passed in as a single vertex, a line segment (two vertices), or as a triangle (three vertices). The attractive feature of GS is to create new primitives on the fly. The GS in Direct3D 10 can read in a single primitive (with optional edge-adjacent primitives) and emit zero, one, or multiple primitives based on that. Using GS with this feature, fins can be extruded from the original mesh, which will be useful for effects like Motion Blur. It is possible to emit a different type of geometry than the input source. For instance, it is possible to read in individual vertices, and generate multiple triangles based on those without CPU intervention. The Stream Output (SO) mechanism allows the GS to circulate its results back to the Input Assembler or a texture buffer such that it can be re-processed. For example the new scene geometry can be created in the first pass (Bezier patches and/or skinning) and then shadow-volume extrusion can be done on a second pass.

Rasterization After all geometry operations are complete, the processed data is streamed into the rasterization unit. Here, all polygons are scan-converted into 2D raster grids (pixels). During the conversion, firstly the triangle is setup to generate all the pixels which cover the projection area of current triangle. Then each generated pixel will compute its properties such as color, the perspective correction coordinate, and texture coordinates by linear interpolation of the corresponding

properties from the three vertices in current triangle. The generated pixels then continue their journey to the next processing stage.

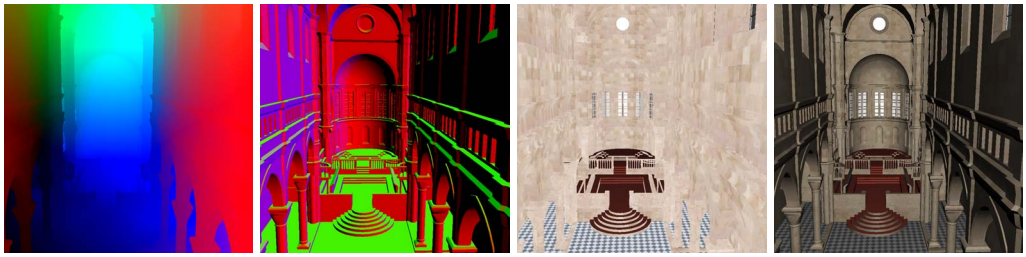
Pixel Processing All pixels generated during rasterization are subject to certain pixel operations which can be summarized as a pixel shader. The pixel shader is fully programmable and allows for customized (dependent) texturing, per-pixel lighting, and many other shading features. Though pixels leaving the pixel shader are properly shaded they are not yet sorted according to their spatial depth. Also, pixels can be transparent in which case their coverage must be accumulated when overlapping. This is why pixels are subject to further raster operations in output merger after shading. Raster operations include visibility tests, proper blending with color entries already resident in memory, as well as anti-aliasing and stencil tests. After the pixel processing, the result pixels will be written into framebuffer and sent to hardware for final display.

General Purpose Graphics Processing Unit GPU is designed specifically for graphics rendering. It can process independent vertices and fragments, but can process many of them in parallel. This is especially effective when the programmer wants to process many vertices or fragments in the same way. In this sense, GPU is stream processor that can operate in parallel by running a single kernel on many records in a stream at once. It is also called *single instruction multiple data* (SIMD) processor. With the evolvement of GPU, nowadays it further increases the flexibility to add new operations and allows for more general purpose programming to facilitate the GPU as powerful multi-core *general purpose graphics processing unit* (GPGPU) rather than a graphics accelerator. GPGPU is the future development trend of GPU. Recently, several general parallel programming architectures, such as CUDA [NVI08], OpenCL [Khronos08] etc, have been proposed and they will also enhance future GPU accelerated rendering development.

2.7.2 Deferred Shading

The standard rendering in GPU starts from the input geometry object, and pass through the whole pipeline to generate final result. Usually we call such a rendering process as *forward rendering*. In forward rendering, the shading computation in pixel shader will be executed for all the pixels from rasterization. However, only the most front pixels will be remained for final result. Hence, the shading computations for the discarded pixels are wasteful. Motivated by this, *deferred shading* technique has been proposed. Deferred shading postpones shading calculations for a pixel until the visibility of that pixel is completely determined. In other words, it implies that only pixels that really contribute to the resultant image are shaded.

The usual implementation of deferred shading contains two passes. In the first



(a) Position (b) Normal (c) Diffuse Materials (d) Final Rendering

Figure 2.15: Deferred shading with G-buffers.

pass, the scene geometry properties, such as position, normal, material, etc, are rendered into intermediate buffer storage to be combined later. These buffers are usually called *geometry buffers* (g-buffers), as shown in Fig. 2.15. When generating the g-buffers in modern GPU, we can relying on the *multiple render targets* (MRT) technique to avoid redundant vertex transformations. In the second pass, a simple full-screen quad is rendered to invoke the shading computation in pixel shader. All the g-buffers can be read by pixel shader and compute the final shading.

Deferred shading can achieve high performance by saving unnecessary shading computation. It also achieve the simpler management of complex lighting resources, ease of managing other complex shader resources and the simplification of the software rendering pipeline. Therefore, deferred shading currently has been widely applied in video games. Most of our GPU-based implementations are based on deferred shading. Because of the use of MRT with a floating point format when generating g-buffers, the memory bandwidth of deferred shading is higher than forward rendering. It is somehow tricky about how to efficiently generate g-buffers with less band width. We would refer the interested readers to this tutorial [Policarpo05] for more details.

Chapter 3

Related Works

In this chapter we briefly summarize and discuss the most related work, which is split into different sections covering general global illumination rendering techniques, real-time soft shadow generation, visibility in indirect lighting, and caustics and participating media. And at the end we introduce several recent research works that are based on the work in this thesis.

3.1 General Global Illumination Rendering Techniques

In the Section. 2.3 of Chapter. 2, we introduce several rendering techniques which accurately or approximately solve the rendering equation to achieve global illumination rendering effects. In this section, we will review and discuss the state-of-the-art representative works of these rendering techniques.

3.1.1 Ray-Tracing

The fist important ray tracing method is *Whitted tracing* [Whitted80]. When a ray hits a surface, Whitted tracing could generate up to three new types of rays: reflection, refraction, and shadow. The reflected ray continues on in the mirror-reflection direction from a shiny surface and the refracted ray travels through transparent material to enter or exit a material (as shown in Fig. 2.1.3). To further avoid tracing all rays in a scene, a shadow ray is used to test if a surface is visible to a light. If the surface at this point faces a light, a ray is traced between this intersection point and the light. If any opaque object is found in between the surface and the light, the surface is in shadow and so the light does not contribute to its shade. During the Whitted tracing process, a single ray per pixel is used to

sample the real world by casting it through center of the pixel and evaluating what it hits. Hence Whitted tracing is only a single sample approximation for the value of each screen pixel, and while computationally simple, it may suffer from inaccuracy. From the point of rendering effects, Whitted tracing intrinsically assumes that the light source is point or spot light (hard shadow) and the reflective material is diffuse or specular (perfect reflection).

In real cases, the area or environmental light sources are common and the surface materials are mostly glossy. To overcome the limitations of Whitted tracing, distributed ray tracing (DRT) [Cook84] is proposed. DRT exploits randomly distributed oversampling and Monte Carlo integration to solve the rendering equation (Eq.2.15). The randomly distributed oversampling is a process where instead of sampling a single value, multiple samples are taken and averaged together. The location of where the sample is taken is varied slightly so that the resulting average is an approximation of a finite area covered by the samples. The DRT can achieve all the sophisticated global illumination effects, such as glossy reflection, translucency, soft shadows and so on. The slight disadvantage of DRT is that the result image might be noisy if not enough samples are used, but otherwise it produces physically correct results. Most of the ground truth comparison results generated in our research works are based on DRT.

Other variants of the original ray tracing approach include path tracing and photon mapping [Lafortune93, Jensen96]. As stated before, all ray tracing algorithms have in common that rays or photons need to be intersected with the geometry to find the closest hit points. To accelerate the intersection query, the hierarchical data structures, such as kd-tree, bounding volume hierarchy (BVH), etc, are usually adopted to organize the input scene. However, creation and update of these hierarchical data structures for fully dynamic scene a rather costly operation and has long prevented ray-tracing approaches from being interactive. Recently, algorithms have been proposed for interactive global illumination using ray tracing [Wald02, Wald03]. However, a cluster of 24 PCs was required to achieve interactivity. With the rapid development of GPU, nowadays Whitted tracing is able to achieve real-time performance for dynamic scene [Purcell02, Roger07]. DRT also can rely on the GPU for acceleration. Recently a new real-time ray tracing engine so-called OptiX [Parker10] has been proposed by NVIDIA®, which is based on using the CUDA GPU computing architecture. Although OptiX accelerates DRT a lot, it is still far from the real-time performance when the amount of random samplings increases. Another interesting research direction is to accelerate the kd-tree creation and update in GPU [Zhou08a]. We would refer the reader for a recent state-of-the-art report for real-time ray tracing for dynamic scene [Wald09].

3.1.2 Radiosity

Radiosity[Goral84, Cohen93] is a finite element method to compute a global illumination solution, where links between mutually visible finite elements are created and radiosity is propagated along those links until a steady state is reached. This computation is generally not real-time, even with improvements such as hierarchical radiosity [Hanrahan91]. There exist approaches to handle and make use of temporal coherence in dynamic scenes; a global illumination solution is incrementally updated by shooting negative light to compensate for changes in lighting or geometry [Chen90, George90, Puech90]. However, updating the link structure is difficult, since visibility needs to be taken into account.

The introduction of programmable graphics hardware has fostered research in GPU-accelerated radiosity. To overcome the visibility problem, Cohen et al. [Cohen85] introduce the hemicube for the visibility computation, and Nielsen et al. [Nielsen02] accelerate the hemicube method with the help of hardware texture mapping. Floating point textures to store the result of the radiosity computation are first utilized by Carr et al. [Carr03]. Progressive refinement radiosity [Cohen88] maps well to a graphics hardware implementation because it requires no explicit storage of the radiosity matrix and it allows the model to be displayed interactively as the solution progresses. The progressive refinement radiosity can be completely implemented in GPU [Coombe04]. However, the algorithm was restricted to planar quadrilaterals. In [Wallner09], a GPU radiosity solver for triangular meshes which was based on [Coombe04] has been proposed.

As stated before, the subdivision schemes for input geometry scene in radiosity methods have strong impact for the rendering quality. It is usually difficult to design a perfect subdivision scheme to very complex scene. Therefore, radiosity is difficult to be directly applied for real applications. Another kernel problem of radiosity is the performance of visibility determination. Neither the ray casting and hemicube are efficient for checking visibility. That's also our motivation to develop *implicit visibility* in Chapter. 4.

3.1.3 Precomputed Radiance Transfer

Precomputed Radiance Transfer (PRT) permits real-time rendering of limited global illumination effects on static objects, such as soft shadows and diffuse/glossy interreflections [Sloan02, Ng03, Liu04]. The global illumination solution is simply parameterized by the incident lighting, which is assumed to be represented by means of basis functions, such as spherical harmonics [Sloan02] or wavelets [Ng03]. By this means, real-time rendering of the static scenes becomes feasible. PRT exploits the limitation to static objects by precomputing all the visibility queries and baking them into the parameterized solution. It has been

shown, that for static scenes the resulting shading can be computed and rendered in real-time on current GPUs. However, dynamic or deformable objects are inherently difficult for PRT techniques, since the visibility cannot be precomputed anymore.

Shadow computation for dynamic scenes can be accelerated by simplifying the geometry. Ren et al. [Ren06] and Sloan et al. [Sloan07] approximate dynamic objects using a sphere hierarchy, whereas Kautz et al. [Kautz04] use a two-level mesh hierarchy. These methods only support deformation on low-polygonal models, and assume that object topology remains static. Further, only low-frequency rendering effects are reproduced in these methods. Similarly, limited dynamic scenes with moving rigid objects can be handled[Zhou05, Sun06], but without taking indirect illumination into account. Recent work [Liu07, Iwasaki07] extends these ideas to render interreflections of dynamic rigid objects. Handling the deformable models remains a challenge.

Material or lighting design usually require the input geometry to be static or contain only rigid transformation. Recently several research works [Sun07, Ben-Artzi08] have conducted BRDF editing with GI effects based on PRT. PRT also can be applied in lighting design [Kristensen05a]. However, for computer game or other interactive applications, PRT can only be applied for the GI lighting of static environment not for deformable characters. Motivated by overcoming the limitation of PRT, our research works develop global illumination rendering for fully-dynamic scene. More introductions about PRT can be found in recent course [Kautz05].

3.1.4 Ambient Occlusion

Ambient occlusion (AO) [Zhukov98] captures a subset of global illumination effects, by computing for each point of the surface the amount of incoming light from all directions and considering potential occlusion by neighboring geometry. The accurate AO usually relies on ray casting to compute the occlusion for all the direction, so its performance can not achieve interactive for fully dynamic scene. In [Pharr04], the ambient occlusion value for a scene is precomputed and stored in textures or as a vertex component on a per vertex base. This kind of off-line precomputation limits this technique only for static scenes.

In order to avoid the precomputation of ambient occlusion terms, Bunnell [Bunnell05] propose to transform meshes into surface discs (surfels) of different sizes, covering the original surfaces. Rather than computing visibility information between points on the mesh, they approximate the shadowing between these discs to determine ambient occlusion. However, highly tessellated objects are needed to get high quality shadows as visibility is estimated per-vertex only. This algorithm is further extended to work on a per fragment basis [Hoberock07]. Kon-

tkanen and Laine [Kontkanen05] present a technique for computing inter-object ambient occlusion. For each occluding object, they define an ambient occlusion field in the surrounding space which encodes an approximation of the occlusion caused by the object. This information is then used for defining shadow casting between objects in real-time. This method works very well for scene objects with rigid transformation, but is limited when arbitrary deformations are mandatory. In [Kontkanen06], an AO method for the character animation is proposed. The animation parameters can be used to control the AO values parametrically on the surface. This technique works very efficient at producing shadows on legs and arms, but cannot account for the neighboring geometry.

The Screen-Space Ambient Occlusion (SSAO) [Mittring07, Bavoil09] techniques can handle full dynamic scene in real-time by treating the Z-Buffer as a geometric guess of the scene and tracing rays on a per-pixel basis to evaluate the AO value. This technique represents the state-of-the-art in real-time AO, but has a strong limitation: being performed entirely in screen space, it ignores any object located outside the field of view C yet these objects may have a significant influence on the ambient occlusion residing on visible objects. A recent paper [Ritschel09b] uses SSAO-like techniques to approximate indirect lighting along with a directional model of visibility. While the results are visually pleasing, the technique shares the same problem as SSAO. To overcome the limitation of SSAO, another recent AO method [Reinbothe09] relies on the surface voxelization of input scene to combine object and image space techniques in a deferred shading context.

3.1.5 Other Global Illumination Methods

Except the aforementioned four kinds of classical global illumination rendering techniques, there are still some other interesting GPU-based GI methods which is related with our research works and worthy to mention in this section.

The instant radiosity technique by Keller [Keller97] traces photons and regards photons stored at hit points in the scene as secondary light sources. These secondary light sources can be treated as *virtual point light* (VPL). Summing up the contributions of all VPLs yields the final result. Interactive frame rates can be achieved with this technique but banding artifacts are likely to appear. The main bottleneck is the need to compute shadowing for each VPL at every step, preventing real-time simulation for complex scenes. If ignore the visibility for the VPL, one-bounce indirect illumination can be rendered at real-time rates in GPU [Dachsbacher05, Dachsbacher06]. However, this allows light to bleed through surfaces, creating unrealistic results. Several bounces of indirect illumination [Nijasure05] can be taken into account by iteratively collecting incident lighting. However, real-time rates can only be achieved with very coarse lighting

approximations.

The Environmental lighting is important for realistic rendering, and it is usually represented as Environment Map and stored as cubemap texture in GPU. Agarwal et al. [Agarwal03] propose an efficient point sampling strategy for environment maps, in the context of ray tracing. This was later accelerated [Ostromoukhov04], and extended to full importance sampling [Clarberg05]. However, all of these techniques are limited to point samples. Similar to Arbree et al. [Arbree05], we employ an environment sampling strategy based on extended light sources in Chapter. 5. We approximate an environment with a collection of square light sources, whereas Arbree et al. use disk-shaped sources.

3.2 Real-time Soft Shadow Generation

A complete review of existing shadow algorithms is beyond the scope of this article and we refer the reader to Woo et al. [Woo90], Hasenfratz et al. [Hasenfratz03b], and to a recent course [Eisemann09] for a detailed overview. In this section, the most related pre-filtering hard shadow mapping techniques and soft shadow methods will be introduced.

3.2.1 Hard Shadow Mapping with Pre-Filtering

Edge anti-aliasing is a classical problem for hard shadow mapping [Williams78]. Unfortunately, standard filtering cannot be applied directly to the shadow map, because the shadow test has to be carried out before the filtering takes place [Reeves87]. A straight-forward step for anti-aliasing is directly applying the graphics hardware's filtering function to filter shadow mapping results. Unfortunately, because the shadow test has to be carried out before the filtering takes place [Reeves87], standard filtering functions cannot be directly applied to depth map. To overcome the brute-force point samplings, several pre-filtering shadow mapping methods [Donnelly06a, Annen07, Annen08b, Salvi08] have been proposed recently to solve this problem. The general idea is to transform the standard shadow test function into a linear basis space. At each frame, the depth values in the depth map can then be pre-filtered (coefficients in the basis). Hence, one can rely on readily available filtering functions, such as mip-mapping or summed-area tables [Crow84] to sample the pre-filtered coefficients. In final shadow rendering step, shadow test function can be approximately reconstructed to achieve shadow in constant-time.

Before going into the details of different pre-filtering shadow mapping methods, let's introduce the basic theory behind the pre-filtering. As shown in Fig.3.1, considering the 3D scene points x_1 and x_2, their 2D projection position in shadow

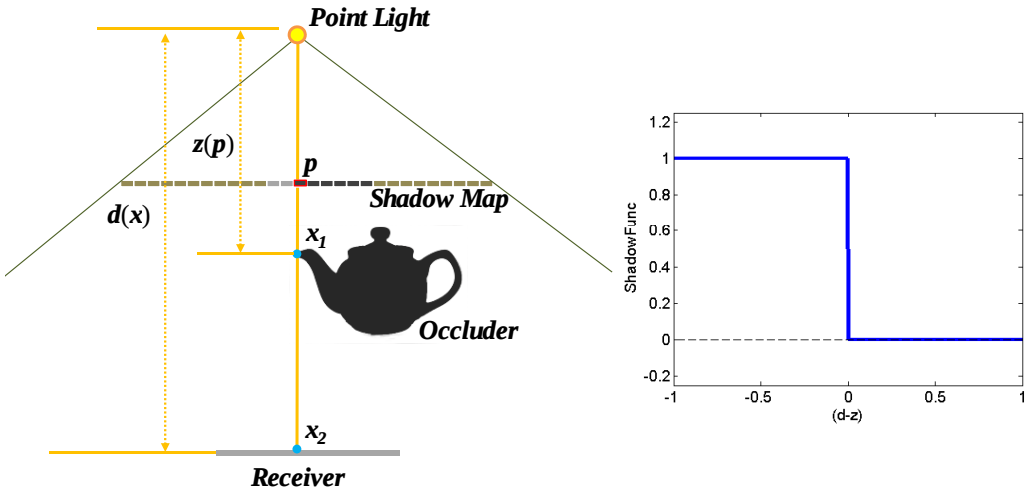


Figure 3.1: The theory of pre-filtering shadow mapping.

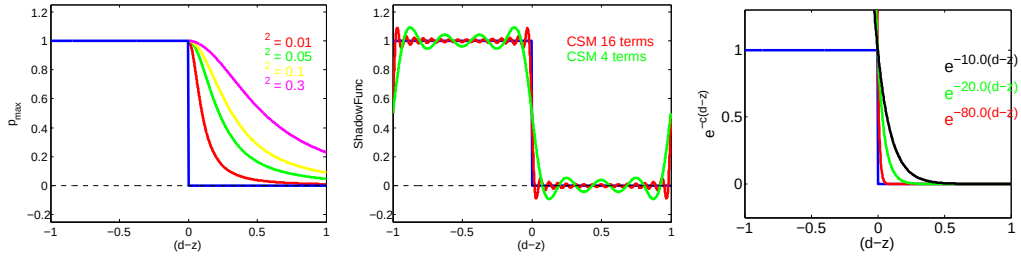
map will be the same at p. Based on the shadow map theory, we know the distance value recorded in the map is the closest one which is from the light source to point x_1. We can represent the recorded depth values in shadow map as a function related with the 2D projection position: $z(p)$. How about the distance from the light source to any point behind x_1, like x_2? It can also be represented as a function related with the 3D scene position: $d(x)$. Based on such a formulation, the shadow comparison operation can be represented as the following Heaviside step function:

$$f(d(x),z(p)) = f(d,z) = \begin{cases} 1 & d \leq z \\ 0 & d > z \end{cases} \tag{3.1}$$

The plot of this Heaviside function is also shown in Fig.3.1. Essentially, all the pre-filtering shadow mapping methods are trying to reconstruct this Heaviside function to achieve shadow comparison.

The *variance shadow map* (VSM) [Donnelly06a] is a probabilistic approach that supports shadow pre-filtering. When generating the depth map, z and z^2 values are stored and used as the mean and variance respectively during rendering to estimate the probability whether a point is in shadow or not. The shadow test is based on one-tailed version of Chebyshev's inequality which only bounds one side of the Heaviside step function. If there exists depth values which are bigger than current pixels depth inside filter kernel, the variance-based inequality evaluation only provides a "big" upper bound and hence incurs incorrect lit in shadow. This is an intrinsic problem for VSM which is so-called "non-planarity" lit. The reconstructed function of VSM is shown in Fig.3.2(a). It is easy to see when the variance ^{2} becomes bigger, the reconstructed results of VSM will become worse. This will produce noticeable high frequency light leaking artifacts for scenes with

high depth complexity. Lauritzen et al. [Lauritzen08] successfully suppress light leaking by partitioning the shadow map depth range into multiple layers. However, the incorrectly-lit due to "non-planarity" problem still exists since there is no correct definition for the left side of shadow test function.



(a) VSM Func (b) CSM Func (c) ESM Func

Figure 3.2: Different pre-filtering shadow functions.

Convolution shadow maps (CSM) [Annen07] transforms depth values into Fourier space. Depending on the truncation order, depth value z is converted into several Fourier basis terms. In the final rendering, pre-filtered Fourier basis terms are fetched from textures to reconstruct a smoother shadow in order to approximate the Heaviside step shadow function. The reconstructed function of VSM is shown in Fig.3.2(b). Since Fourier reconstruction function is "double-bounded", the "non-planarity" problem does not exist for CSM. Yet the shadow quality depends on the truncation order and high order will incur impractical memory requirements for storing basis textures. When using few basis terms, ringing artifacts and incorrect contact shadow can be observed.

Exponential shadow maps (ESM) [Annen08b] [Salvi08] use the exponential function to approximate the Heaviside shadow test function, as shown in Fig.3.2(c). Since the exponential function is also *single-bounded*, it utilizes the standard percentage closer filtering (PCF) for "non-planarity" regions [Annen08b], which are detected relying on min-max pyramid texture. For hard shadows, PCF is good for repairing it since usually the percentage of "non-planarity" parts in the whole rendering scene is low. Yet for soft shadow, such a percentage becomes higher with the increasing light size. Hence brute-force PCF for many "non-planarity" parts will be prohibitively time-consuming.

3.2.2 Soft Shadow Volume

Based on shadow volumes [Crow77], Chin and Feiner [Chin92] construct separate BSP trees for the scene, for the umbra volume and for the outer penumbra volume. Shadow receivers are then classified into three regions: fully lit, umbra,

and penumbra. An analytic shadow term is computed by traversing the BSP tree of the scene and clipping away the occluded parts of the polygonal light source. Tanaka and Takahashi [Tanaka97] propose culling methods for efficiently determining the set of objects that can affect the shadowing of a given point. Assarsson and Akenine-Möller [Assarsson03] describe an approximate soft shadow volume algorithm, which offers realtime performance in simple scenes. Two gross approximations are made: assumption that the silhouette of an object is constant from all receiver points, and a heuristic occluder fusion method. Hence, it is less suitable for scenes with rich geometry. Laine et al. [Laine05b] and Lehtinen et al. [Lehtinen06] remove these limitations in the context of ray tracing. Laine and Aila [Laine05a] transpose the processing order of ray tracing. Instead of searching for a triangle that blocks the current ray, they find all rays that are blocked by the current triangle. This leads to different scalability characteristics and memory requirements compared to ray tracing. All the soft shadow volume method are based on geometric information, so that they are sensitive to scene complexity. Therefore, it is inappropriate to utilize soft shadow volume methods for real complex scenes, like tree, foliage, etc.

3.2.3 Soft Shadow Mapping with Backprojection

In recent work [Atty06, Guennebaud06], researchers have transferred ideas from classical discontinuity meshing [Stewart94, Drettakis94] to the shadow mapping domain. Such techniques treat the shadow map as a piecewise constant approximation of the blocker geometry and each shadow map texel is considered as a rectangular micro-patch parallel to the light source. The shadow value is computed as the fraction of coverage of blocker geometry projected back onto the area light. Although these backprojection-based methods stem from physically correct theory, crude approximations of blocker geometry may yield either incorrect occluder fusion or light leaking. The work by Guennebaud et al. [Guennebaud07] and bitmask soft shadows [Schwarz07] remove most of these problems, but increase the algorithmic complexity or computation time. More recently, Yang et al. [Yang09] accelerate backprojection soft shadow mapping by introducing a hierarchical technique, which results in better performance for large penumbra but is still complex for real applications.

3.2.4 Soft Shadow Mapping with Pre-Filtering

We have already discussed percentage closer soft shadow mapping (PCSS) [Fernando05a] method in details in Section. 2.4.2 of Chapter. 2. One key insight of PCSS is that both average blocker depth computation and soft shadow test are based on brute-force point sampling of the depth map. When the

area light size becomes large, many sampling points (e.g., 30×30) are required to avoid banding artifacts. Motivated by this, it is natural to push aforementioned pre-filtering hard shadow mapping methods further to support PCSS soft shadow.

Based on PCSS, *SAT-based variance shadow map* (SAVSM) [Lauritzen07] is proposed to improve the speed relying on VSM [Donnelly06a]. Although the soft shadow test step is sped up using pre-filtering, there is no obvious way to correctly pre-filter average blocker depth values based on the VSM theory. The average blocker depth evaluation step is still performed by brute-force point sampling of the depth map. Furthermore, despite the correct estimation of the average blocker depth in SAVSM, it may still show "non-planarity" lit problems.

Soler and Sillion [Soler98] propose an image-based shadow algorithm based on convolution. Convolutions can be computed efficiently, even for large penumbrae. Soler and Sillion do not employ a depth buffer and therefore require an explicit notion of blockers and receivers, and cannot directly support self-shadowing. Our research works in Chapter. 5 apply a similar convolution in the context of shadow mapping, which naturally allows for self-shadowing. In Chapter. 5, we propose *convolution soft shadow map* (CSSM) based on CSM [Annen07]. CSSM is also implemented in PCSS soft shadow framework. Relying on *double-bounded* Fourier reconstruction, CSSM can pre-filter not only shadow test but also average blocker depth computation. Yet the number of Fourier bases for reconstruction is usually high (≥ 4), which results in high amount of texture memory, and limits its practicability. To overcome this, in Chapter. 7 we further propose *variance soft shadow mapping* methods which not only successfully applies pre-filtering for average block depth evaluation based on a novel depth computation formula, but also successfully handles the "non-planarity" lit problem.

3.3 Visibility in Global Illumination

Visibility determination usually dominates the performance of global illumination algorithm. Dachsbacher et al. [Dachsbacher07] develops a real-time global illumination method that handles visibility by transferring *anti-radiance* for scenes with limited dynamics. While this bears many similarities to our implicit visibility algorithm in Chapter. 4, our implicit visibility handling is slightly more flexible, as we impose no restrictions on the dynamics of the scene, but is also slightly more expensive.

Instant Radiosity [Keller97] can achieve interactive frame-rates for large scenes by using a crude point-based representation of geometry when computing the shadow map for each VPL [Ritschel08b]. Such an imperfect shadow map method is very efficient and visually plausible. However, the point hierarchy requires pre-processing which limits its applications for some scene setting with

topology changes. Their following works [Ritschel09a] introduce the hierarchical organization of point samples to amend holes in shadow map, and further improve the quality of indirect lighting.

Several global illumination methods use the idea of *clustered visibility*. The lightcuts method [Walter05] imposes a hierarchy over groups of virtual point lights; only a single visibility test is performed for each group. Generating clusters of lights with only a single shadow map for each cluster was used by Hasan et al. [Hašan07] for GPU-friendly illumination from many lights. This idea was further extended to visibility cuts [Akerlund07] and GPU implementations [Ritschel08a][Cheslack-Postava08]. Kristensen et al. [Kristensen05b] group VPLs into light clouds for real-time relighting of static scenes. In all these approaches, a single binary visibility test for a sender cluster is used. We extend this idea by taking the extent of the sender (cluster of VPLs) into account through the use of a soft shadowing method.

Our algorithm in Chapter. 6 is inspired by the idea of clustering an environment map into a set of area lights for real-time natural illumination [Annen08a]. We extend this idea to deal with indirect lighting using a real-time GPU-based clustering technique.

3.4 Caustics and Participating Media

Caustics are phenomena caused by the focusing and de-focusing of light rays upon interaction with specular surfaces. Both, specularly reflective and specularly refractive objects give rise to complex volumetric light distributions. The interaction of these light distributions with opaque surfaces results in *surface caustics* while the effect of their interaction with *participating media* is known as *volume caustics*. The rendering literature on the subject can be grouped accordingly.

3.4.1 Surface Caustics

Surface caustics are generated by rays that refocus on diffuse surfaces after refraction or reflection on a specular surface. In computer graphics, photon mapping [Jensen01] is the gold standard technique to successfully simulate this kind of phenomenon. Relying on the fast processing speed of modern GPUs, several methods were developed to approximate reflection [Estalella06, Umenhoffer07, Yu05] and refraction [Oliveira07, Davis07, Wyman05] effects in real-time. While the aforementioned techniques were developed for rendering reflected and refracted viewing rays, they are equally well applicable for fast photon path calculations on graphics hardware.

The first GPU photon map implementation was described by Purcell et al. [Purcell03]. Recently, several methods for approximating the effect of surface caustics on the GPU have been proposed [Shah07, Hu07, Wyman06, Szirmay-Kalos05]. As in normal photon mapping, these methods follow specularly reflected and refracted photons from the light source and store their hit positions into a so called photon buffer. A second pass then re-organizes this information and splats it into a caustic map, which is then projected onto the scene similar to shadow mapping. In [Wyman08b, Wyman09], the speed and quality of this basic technique is improved using a hierarchical data structure to discard non-contributing and redundant photons. Yu et al. [Yu07] present a real-time caustics rendering method based on image space computations, modeling the effect of specular objects as a distorted generalized linear camera model.

3.4.2 Participating Media

Participating media introduce global shading effects for viewing rays. Instead of shading single surface points based on an approximation of the incident radiance, line integrals over viewing rays have to be computed. Computing the incident radiance is significantly complicated by the effect of multiple scattering [Kajiya84].

Volumetric photon mapping [Jensen98] decouples radiance transport from the light source into the medium and the integration step to determine the radiance of the viewing ray. The final gathering step along a ray can be computed directly per ray [Jarosz08] or in screen space [Boudet05]. A complete description of off-line methods for rendering participating media is, however, beyond the scope of this paper; for a good overview we refer the interested reader to [Cerezo05].

Typical applications of real-time rendering in the presence of participating media are the visualization of clouds [Dobashi00, Harris01] and smoke [Ren08, Zhou08b]. Approximating the shafts of light, that are a typical effect of single scattering, can be achieved by blending layered materials [Dobashi02] or by warping volumes [Iwasaki02]. Sun et al. propose an analytical airlight model for real-time single-scattering in homogeneously scattering media [Sun05]. This work was extended to real-time rendering of volumetric shadow regions by Wyman and Ramsey [Wyman08a]. Our work uses this technique to render the shadow regions surrounding volumetric caustics.

3.4.3 Volume Caustics

Volume Caustics can be computed using the volumetric photon mapping technique [Jensen98], an extension to standard photon mapping [Jensen01], by tracing and storing photons throughout the volume covering the participating medium. Volume caustics are typically seen in under water scenarios where shafts of

light are visible due to focusing of light rays by the water surface. Rendering of these effects was demonstrated by Nishita et al. [Nishita94] and by Ernst et al. [Ernst05] using a triangle-based caustics volume reconstruction method that is very geometry-intensive.

The eikonal rendering technique proposed by Ihrke et al. [Ihrke07] uses a volumetric object description to render most of the effects of refraction and participating media in real time. However, if the lighting conditions or scene geometry change, several seconds are required for the re-computation of the incident radiance distribution. Sun et al. [Sun08] modify the previous technique by combining it with a photon mapping algorithm. They achieve fully dynamic rendering of refraction, absorption and single-scattering effects in participating media at interactive frame rates. Volumetric object representations, however, induce a discrete representation of refractive objects and tend to blur surface and caustic details.

3.4.4 Lines as Rendering Primitives

Lines are often used as rendering primitives in visualization of vector field data [Zöckler96, Mallo05]. They are rarely employed as general primitives, even though their use has been advocated by some authors [Wong05] and efficient applications in natural scene rendering have been described [Deussen02].

Recently, the interactive global illumination literature has produced some applications of lines as fast intermediate rendering primitives. Krüger et al. [Krüger06] demonstrate a screen-based surface and volume caustic rendering technique. The authors directly splat energy to screen pixels using refracted lines as querying primitives. The focus is on surface caustic rendering even though some results are shown for volume caustics as well. Another, connected technique by Sun et al. [Sun08] also uses lines between specular object and receiver as rendering primitives. Here, however, the lines are rasterized into an intermediate illumination volume, similar to [Ihrke07], which enables a correct evaluation using a second ray marching step.

Our work in Chapter. 8 can be seen as a combination of the previous two techniques. We combine the strength of the screen-based approach, which is very high resolution, high performance rendering at low memory foot-prints, with the physical accuracy of the photon-based approach, which in its original implementations is either too slow for real-time applications [Jensen98] or yields blurry results [Ihrke07, Sun08] due to high memory consumption and thus limited resolution.

Contemporary to our work, [Papadopoulos09] presents real-time caustics and godrays for underwater scenes using an implementation which is similar to the method of Krüger et al. [Krüger06]. In contrast to our work, no comparison to ground truth is shown and only a homogeneous medium is supported.

3.5 Successive and Active Future Work

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

In [Meyer09], a non-recursive and efficient data-parallel algorithm to create links and a patch hierarchy for implicit visibility method in Chapter. 4 is proposed. It is running entirely on the GPU using CUDA for implementation. Hence, this method is fast enough to create links and a compact patch hierarchy from scratch for every frame. Including the simulation of light transport, we can render dynamic scenes with indirect light at interactive frame rates.

Inspired by the theory of CSSM in Chapter. 5, Jason and Bavoil [Jansen10] introduce a novel algorithm called Fourier Opacity Mapping (FOM) for rendering artefact-free pre-filtered volumetric shadows in cases where spatial opacity variations are smooth (e.g. smoke, gas and low-opacity hair). This method is robust enough and has been adopted in the shipping video games, such as Batman, Arkham Asylum, etc, for shadowing smoke particle systems. Another recent paper [Davidovič10] adopts our visibility clustering strategy in Chapter. 6 for computing visibility in off-line detailed glossy illumination.

Our concept about *Lines as Rendering Primitives* in Chapter. 8 for volume caustics is also adopted by [Sun10]. This method is an efficient off-line technique to render single scattering in large scenes with reflective and refractive objects and homogeneous participating media. Compared with our interactive volumetric caustics rendering method, it is more physically correct but also takes much more time for rendering.

Part II

Interactive Global Illumination Using Implicit Visibility

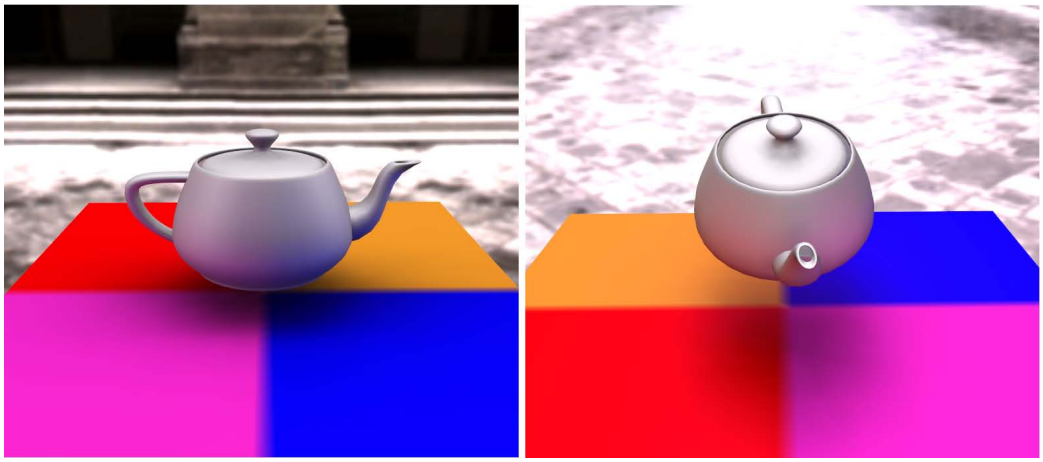
Chapter 4

Interactive Global Illumination Using Implicit Visibility

4.1 Introduction

In order to render a scene photo-realistically many local and global illumination (GI) effects have to be faithfully reproduced. Today, real-time rendering of local illumination effects is state-of-the-art and used in many computer games and interactive environments. Unfortunately, scenes rendered in this way often have an artificial look as they lack more sophisticated appearance details such as interreflections. The GI computation adds this additional bit of realism by taking into account not only light that comes directly from the light source but also indirectly through reflection from other surfaces. However, the simulation of the GI effects is very complex and up to now it has been illusive to render full global illumination solutions in real-time on a single PC. The problem's complexity originates from the fact that during lighting simulation every scene element interacts with many others. Furthermore, visibility between scene elements has to be precomputed, as light can only travel between mutually visible points in the scene. This expensive-to-compute information is used in all traditional rendering algorithms [Cohen93, Lafortune93, Jensen96, Veach97].

In this chapter, we propose a novel algorithm to render the GI effects at interactive frame rates on a single PC. The core of our method is a hierarchical radiosity-like link structure describing the light transport between individual scene elements. To overcome the computational bottleneck of having to compute visibility information explicitly at each frame, we propose the concept of *implicit visibility*. By this means, we are able to quickly derive visibility between scene elements implicitly from the hierarchical link structure while it is being built. We propose methods to efficiently construct this link structure and show that the final



(a) One Bounce, 6x12x12, 6.43FPS (b) One Bounce, 6x12x12, 6.43FPS

Figure 4.1: Teapot with indirect lighting (3878 vertices).

global illumination solution can be quickly computed on the GPU (Figure 4.1). Our method can reproduce interreflections under environment map lighting as well as area light sources at interactive frame rates — even for dynamic scenes with deformable objects. Interactivity is achieved by sparsely sampling visibility, which makes our method most suited for diffuse or low-glossy scenes under large area lighting.

This chapter is organized as follows: Section. 4.2 describes how we reformulate and solve the rendering equation to accommodate the concept of implicit visibility. Section. 4.3 describes in detail the construction and administration of our hierarchical link structure, and explains how to efficiently map these concepts onto the GPU. We demonstrate the high visual quality of our results in Section. 5.5 and conclude in Section. 4.5 with an outlook to future work.

4.2 Global Illumination using Implicit Visibility

In the following, we derive the theoretical fundamentals of fast interactive global illumination based on implicit visibility. We firstly review the rendering equation [Kajiya86], which has been introduced in the Section. 2.2 of Chapter. 2. Then, we rewrite the rendering equation in such a way that a global illumination solution can be computed in a way similar to early non-diffuse radiosity methods [Immel86]. In contrast to radiosity methods, however, we compute visibility implicitly while

building the link structure. The rendering equation can be written as follows:

$$L_v(x \to \Theta_o) = L_e(x \to \Theta_o) + \int_{\Omega_x^+} f_r(x, \Theta_i' \leftrightarrow \Theta_o') \cdot L(x \leftarrow \Theta_i) \cdot (\mathbf{n}_x \cdot \Theta_i) d\Omega_{\Theta_i}, \tag{4.1}$$

where x is a point in the scene, L_e is the emitted light, f_r is the BRDF, $\mathbf{n}_x$ is the normal at x, Θ_i and Θ_o are the global light and viewing directions, and Θ_i' and Θ_o' are light and view in local coordinates.

Similar to [Immel86] we discretize the sphere into N_{bin} small spherical bins, each of which has a solid angle Θ_{bin_i}. This allows us to rewrite the rendering equation as:

$$L(x \to \Theta_o) = L_e(x \to \Theta_o) + \sum_{i=1}^{N_{\text{bin}}} K_i(x, \Theta_o), \tag{4.2}$$

with

$$K_i(x, \Theta_o) = \int_{\Omega_{\text{bin}_i}} f_r(x, \Theta_i \leftrightarrow \Theta_o) \cdot L(x \leftarrow \Theta_i.) \cdot (\mathbf{n}_x \cdot \Theta_i) d\Omega_{\Theta_i}$$

We now rewrite the K_i as an integral over all surface elements y inside Ω_{bin_i} instead of solid angles:

$$K_i(x, \Theta_o) = \int_{y \in \Omega_{\text{bin}_i}} f_r(x, \Theta_i \leftrightarrow \Theta_o) \cdot L(x \leftarrow \Theta_i) \cdot V(x,y)(\mathbf{n}_x \cdot \Theta_i) \cdot \frac{(\mathbf{n}_y \cdot (-\Theta_i))}{r^2} dA_y, \tag{4.3}$$

where V is the binary visibility between two points.

It is now possible to make several simplifying assumptions to speed up the computation. First, we assume that for each element inside a bin the outgoing radiance is constant across its extent. Furthermore, we assume that the size of each element is very small, such that the cosine between the integration direction and the normal is essentially constant. Finally, we assume that surface elements are either completely visible or completely occluded. This allows us to rewrite Equation (4.3) as:

$$K_i(x, \Omega_o) \approx \sum_{j=1}^{\#y \in \Omega_{bin_i}} f_r(x, \Theta_{\text{i},j} \leftrightarrow \Theta_o) \cdot L(x \leftarrow \Omega_{\text{i},j}) \cdot V(x,y_j)(\mathbf{n}_x \cdot \Theta_{\text{i},j}) \frac{(\mathbf{n}_{y_j} \cdot -(\Theta_{\text{i},j}))}{r^2} A_{y_j}, \tag{4.4}$$

where $\Theta_{\text{i},j}$ is the direction to the surface y_j. Note that, we only evaluate the binary visibility once (between x and the surface element's center) and turn the original integral into a sum over surface elements.

We make the final assumption that a surface element always covers the extent of a spherical bin Ω_{bin_i} completely. This means that only the closest element needs to be considered and Equation (4.4) becomes:

$$K_i(\Theta_o, x) \approx f_r(x, \Theta_{\text{i},s} \leftrightarrow \Theta_o) \cdot L(x \leftarrow \Theta_{\text{i},s}) \cdot (\mathbf{n}_x \cdot \Theta_{\text{i},s}) \cdot \frac{(\mathbf{n}_{y_s} \cdot (-\Theta_{\text{i},s}))}{r^2} \cdot A_{y_s}, \tag{4.5}$$

where y_s is the closest surface element.

We use this formulation to render a global illumination solution by means of a radiosity-like algorithm. The discretization into bins allows us to borrow the idea of shadow mapping. We create a (hierarchical) link structure between scene elements, where we store links in the discretized bins at each element, as opposed to a simple list of links used for standard radiosity algorithms. When creating the link structure, each bin will only store the shortest link, i.e., the link connecting to the *closest* surface element. Using this scheme, the visibility information will be implicitly retrieved from the link structure. This can be seen as a variant of omni-directional shadow mapping [Brabec02]; for each point x we discretize visibility for its upper hemisphere (Figure 4.2e).

4.2.1 Conceptual Overview

Conceptually, our algorithm is very similar to standard radiosity. We create links between scene elements and light sources, and transfer energy between them until the solution is converged (or a certain number of iterations has been reached).

In contrast to standard radiosity, we do not store a simple list of links at each scene element, but structure the links by storing them in bins. A non-hierarchical version of our algorithm would simply try to connect all scene elements with each other. Whenever a link is about to be created, its respective bin is queried and checked if there already exists a link and if that link is shorter or longer than the new link. In case the new link is shorter, it replaces the old one; if not, the old one remains. After all links have been created, normal shooting or gathering iterations can be run to transfer energy. Similar to Immel et al. [Immel86], this allows for diffuse as well as glossy direct and indirect illumination.

Of course, this basic algorithm is inefficient as a non-hierarchical link structure grows quadratically in the number of scene elements. In the following, we therefore develop a hierarchical version of this algorithm, which enables us to obtain near-real-time frame rates for dynamic scenes on a single PC.

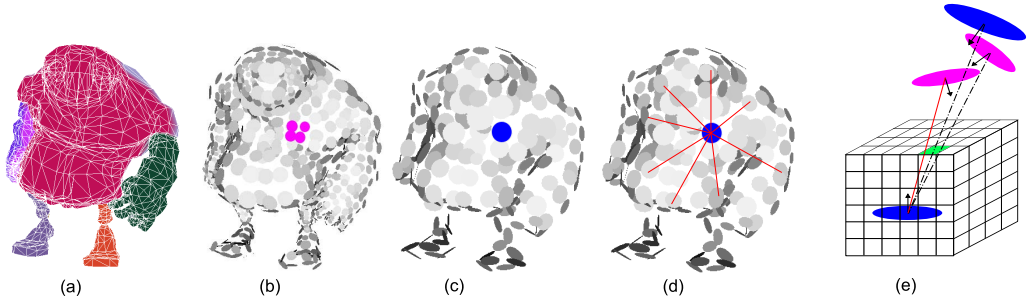


Figure 4.2: (a) Color-coded surface segments representing the coarsest approximation level of the geometric hierarchy. (b) Four surface elements of the finest level of the hierarchy and the corresponding element on the next coarser level (c). (d) Each surface element features (visibility/radiance) links to many other surface elements. (e) The sphere of directions for each element is discretized into cube-map bins, each one of them storing the shortest link to another disc.

4.3 Hierarchical Implicit Visibility

Our method is visualized in Fig. 4.2 and pseudo-code can be found in Algorithm. 4.3. In a preprocessing step, we create a *geometric hierarchy* for each object. This geometric hierarchy is only computed once and is then re-used at run-time to construct the *hierarchical link structure*, which is the data structure for computing the actual global illumination solution. At run-time, we first update the data associated with the surface elements (positions, etc.), as they might have changed from the last frame. We then construct the hierarchical link structure using implicit visibility as indicated before. After construction, the hierarchical link structure needs to be refined in a second pass to propagate the implicit visibility information to all levels. The propagation of energy is very similar to standard radiosity. We will detail our method in the following.

4.3.1 Geometric Hierarchy Preprocessing

In order to facilitate illumination computations, we represent our objects using a hierarchy of *surface elements*. A surface element is an oriented disk with a position, normal and area. The surface elements at the finest scale are based on the vertices of the input model(s) (Figure 4.2b). We chose discs centered around vertices as they can be easily computed for any type of mesh. The position and normal information of each surface element is known from the input model. Similar to [Bunnell05], its area is computed as one-third of the total area of all triangles sharing this vertex.

To speed up the run-time process, we precompute a geometric hierarchy of

Algorithm 1 – Main Algorithm

1: *CreateSurfels()*: Create surface elements based on the input geometry information (vertex, face, etc).
2: *CreateGeometricHierarchy()*: Create hierarchical geometric structure for each object.
3: **for** each frame **do**
4: *UpdateElements()*: Update the geometry information for initial geometric hierarchy.
5: *CreateHierarchicalLinks()*: Create hierarchical links between elements.
6: *RefineHierarchicalLinks()*: Refine links (top-down, remove unnecessary links).
7: *PushdownLinks()*: Push all links to leaf node.
8: **for** each light bounce **do**
9: *ComputeIlluminateLeafNodes()*: Gather incident energy from links and compute illumination results in leaf nodes.
10: *PullupEnergy()*: Pull up the indirect lighting energy from leaf nodes.
11: **end for**
12: **end for**

the surface elements, which is then re-used at run-time. Similar to other radiosity methods, we want to cluster the surface elements in a way such they are adjacent and oriented similarly. Different methods exist to achieve this goal [Garland01, Smits94]. However, our models are allowed to deform at run-time preventing an optimal precomputed solution. We adopt the simple technique by Bunnel [Bunnell05] and use UV texture space segments (typically provided by the artist to enable texturing) as the coarsest cluster unit (see Figure 4.2a for an example).

For each UV segment, we create one *hierarchical quad-tree* representing a spatial disc hierarchy for all vertices in the segment. The root node of the tree is a surface element approximating the whole UV segment, the leaf nodes are the discs corresponding to single vertices. Each surface element in the tree can have up to four smaller child surface elements on the next lower level (Figure 4.2b and c). For each surface-element in the hierarchy, we store its position (average position of all its child surface elements), the overall surface area, as well as the average normal direction. The hierarchical structure is only computed once. However, the average position as well as normal is re-computed every frame in order to support dynamic models. The area of most elements varies very little during animation, therefore, the area does not have to be recalculated for each frame. Please note, that the terms node, surface element and disc are used interchangeably.

4.3.2 Creating the Hierarchical Link Structure

For each frame, we recompute a hierarchical link structure, which is used to perform light propagation but also implicitly determines visibility. This subsection details how this structure is created and refined (corresponds to steps 4–7 in Algorithm. 4.3).

Update Elements of Geometric Hierarchy

Our method allows objects to move around and even deform. Therefore, the stored geometric information needs to be updated accordingly while preserving the hierarchy. At each frame, we therefore update position and normal of each surface element. The data of each parent node is updated based on its children. Note that the hierarchy itself remains untouched. This process is similar to updating a bounding volume hierarchy in ray-tracing [Wald07].

Initial Hierarchical Link Structure

After the geometric information has been updated, we can proceed and build the hierarchical link structure. As stated before, we base the link creation on the precomputed geometric hierarchies. We start by linking *all* top level nodes of the geometric hierarchies. Whenever a link between two nodes (called A and B in the following) is about to be created, we perform the following checks:

- If the solid angle of B as seen from A's position is bigger than the solid angle of the link's respective bin, then the B-node should be subdivided, i.e., we try to link A to B's children (going down the geometric hierarchy). The same check is performed for A as seen from B.

- If there is already a link that has a shorter distance, which is determined by checking the link stored in the respective bin, no link will be created.

In all other cases, we create a link between the two surface elements A and B and store it in the respective bins of A and B.

As can be seen, there are two main metrics to determine whether two nodes can be connected. First, the solid angle determines whether the two surface elements are too big to be connected and should be subdivided. If the surface elements are bigger than a bin, it might happen that bins don't get filled with links, even though there is an element in that direction. This would prevent the implicit visibility to be evaluated correctly. Therefore, we go further down in the hierarchy. Second, the length of the link is used to determine if the other surface element is visible at all.

Discretization We chose the cube-map parameterization to discretize the sphere of directions. In other words, a bin corresponds to a texel in the cube-map. The main advantage of a cube-map lies in the efficient mapping of a direction to a bin.

Refining the Hierarchical Link Structure

During the creation of the hierarchical link structure, it is possible that a surface element and its child nodes contain different links in the same bin/direction because the order in which links are created is arbitrary. This is illustrated in Figure 4.3. The link creation process first happens to connect surface elements *A* and *B*. Then, in the next step, *A* and *C* are connected but since they are closer together, the links are created further down in the hierarchy (*A* and *C* are subdivided). The original link between *A* and *B* remains however, as the new shorter link is created further down in the hierarchy and does not remove the original link.

The purpose of refining the hierarchical links is to delete those incorrect (and redundant) links. To this end, we traverse the tree(s) in a breadth-first manner. During traversal, we compare the links in the bins of all parent-nodes and the links in the bins of the currently visited node. If, for a given bin, the current node contains a shorter link than a parent node, the parent-node's link is removed and pushed to the siblings of the current node (if they don't contain shorter links). If

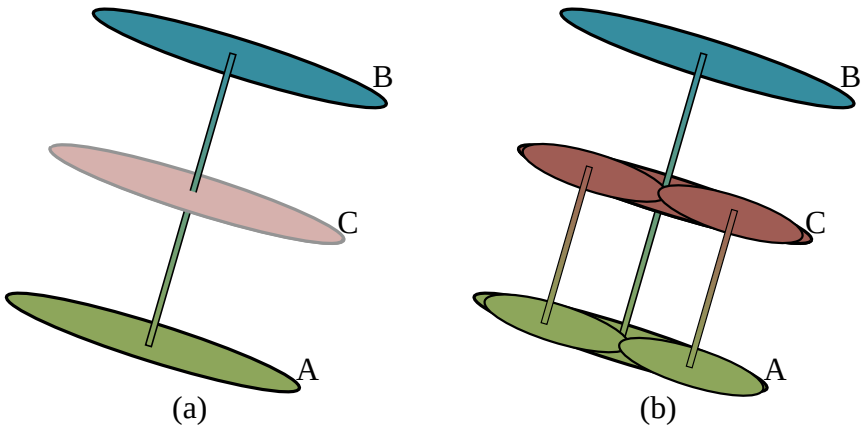


Figure 4.3: Linking problem: In (a), the elements *A* and *B* are connected with a single link. In the next step, the algorithm tries to connect *A* and *C*. Since they are close by, both *A* and *C* get subdivided and links are created further down in the hierarchy. Now there are inconsistent links at different levels in the hierarchy: *A* is still linked to *B*, even though there are shorter links further down in the hierarchy between *A* and *C*.

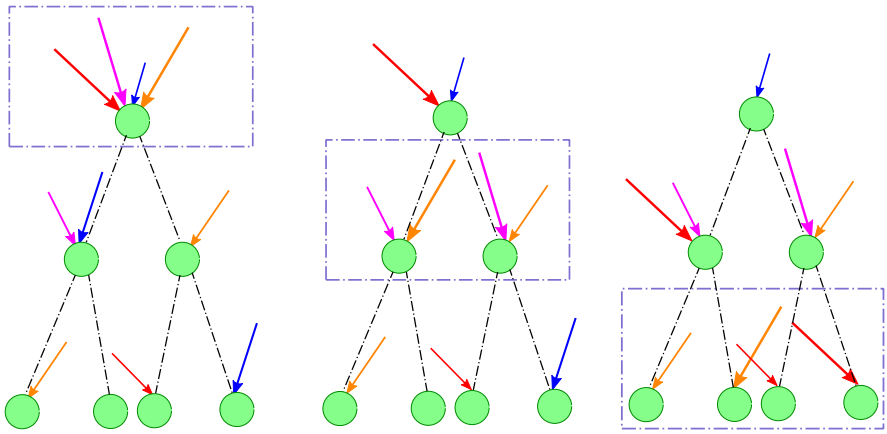


Figure 4.4: Refining hierarchical links: Different colors refer to different bins, and the length of each arrow represents the link's length. We traverse the tree breadth-first, and compare the links of parent and child nodes. If there is a shorter link in a node further down in the hierarchy, we remove the parent-node's link and push it to the siblings of the node. If there is a longer link, it is removed.

the parent node contains a shorter link, the child-node's link is simply removed. This refinement removes any incorrect links. Note that the refinement of links can be done in a single traversal of the tree by keeping track of which bins have links further up in the hierarchy.

Push-Down of Links

Our goal is to implement the illumination computation on the GPU. Unfortunately, GPUs only support very limited scatter operations, i.e., data cannot be written to arbitrary positions but usually only to the current raster position. The push part of the push-pull used by hierarchical radiosity algorithms [Cohen93] requires a scatter operation, as data is written to all the child nodes of a parent node.

We avoid this scatter operation and enable an efficient GPU implementation by pushing down links from all the interior nodes of our hierarchy to the leaf nodes. More specifically, the previously bidirectional links between two nodes are split into two unidirectional links through which energy is received at each node. All the receiving ends of the links are then pushed down the hierarchy to the leaf nodes. This step can be combined with the link refinement from the previous subsection.

4.3.3 Illumination Computation

Global illumination is computed in a similar manner to hierarchical radiosity. Energy is transferred between nodes along links. We chose a gathering approach, i.e., at each node, we gather all the energy from all incident links. The incident light is then convolved with the BRDF and converted to outgoing radiance. In case of diffuse BRDFs, the outgoing radiance is constant for all outgoing directions and we just store a single RGB triple. In case of glossy reflections, we augment our bin structure and store the outgoing radiance per direction in it.

As we have pushed down all receiver links to leaf nodes, outgoing illumination is only computed at leaf nodes (no other nodes can receive energy). Nonetheless, just like in hierarchical radiosity, we need to pull up the outgoing energy to the parent nodes, which is achieved by traversing the tree bottom-up and accumulating energies. These two steps need to be iterated to account for indirect illumination. This procedure can be easily implemented on the CPU, but it unlocks its full potential only when implemented on the GPU.

GPU implementation

We store our surface elements, i.e., positions and normals, in two floating point textures. The hierarchical tree is stored in a texture in a pointer-less manner based on node indices. E.g., a full quadtree with 21 nodes and three hierarchy levels has indices 1–16 for the leaf nodes, indices 17–20 for the second level, and index 21 as the root node. Hence, the index of a node is sufficient to compute the indices of child and parent nodes. A third texture is used to store all the links. In order to allow for fast construction of this texture, we simply flatten the $6 \times N \times N$ bin structure of each node and store its content in the 2D domain (we actually store this data split over several textures). A fourth texture contains the outgoing (and unshot) energy for each node.

Computing the illumination is rather straightforward given these textures. For each leaf node, we loop over all its links and gather and sum the unshot energy from them. It is then converted into outgoing radiance by multiplying with the albedo of the node. After the energy has been gathered at all leaf nodes, we need to perform the traditional push-up operation. The pointer-less tree representation allows us to do this very efficiently by traversing bottom-up through all nodes of the tree. For each node, we accumulate the outgoing radiance weighted by the area ratio. These two steps can be repeated to account for several bounces of indirect illumination.

For final display, we convert the texture containing outgoing radiance into a vertex texture, which is used to set the color at the vertices of the model.

Speedup The direct lighting computation can be sped up, as there are generally

few links to the light sources. Instead of going through all bins, we create a special texture that contains for each node: the number of light links and the actual links (indices to nodes). Now, we only need to go through those links to gather energy.

4.3.4 Light Sources

We support area light sources as well as environmental lighting. Area light sources are geometry like any other object, with the notable difference that their initial outgoing radiance is set to be non-zero.

Environmental lighting could be handle the same way, but allows for a simple optimization. Instead of creating geometry for the environment, we initially omit it completely and create our hierarchy with objects only. We now use the observation that any empty bin (in the leaf nodes) can see the lighting environment and therefore receives light from it. Our optimized direct lighting step (see above) checks for this, and gathers light from the environment for any empty bin.

4.4 Results

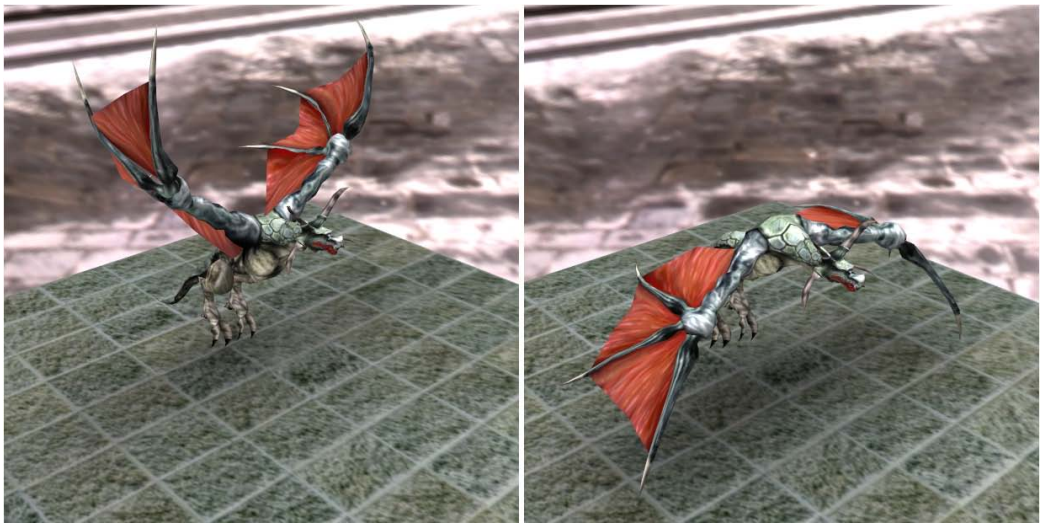
Our method enables interactive rendering of fully dynamic scenes with direct and indirect illumination. Figure 4.1 shows an example where a teapot reflects the colored pattern of a ground plane. Also note the soft shadow cast by the environmental lighting. This runs at interactive speeds (around 7 FPS) on an NVIDIA 8800. Deformable objects, as shown in Figure 4.5, can also be handled easily.

Figure 4.6 demonstrates that our method can handle shadows and indirect lighting effects between objects. Note in (b) how there is a green sheen on the grey chess piece.

Figure 4.9 compares a reference image (a) computed with path tracing to a result of our method (b). Despite all the approximations we make, as detailed in Section 4.2, the differences are minor. Our method produces slightly softer shadows, which is less noticeable for a directional discretization of $6 \times 16 \times 16$. The differences become more prominent for coarser discretizations and artifacts appear.

Figure 4.7 shows a teapot illuminated with an area source. Overall, the shading compares well to the reference image. However, discretization artifacts become obvious in the shadow area. These artifacts can be reduced by either increasing the number of bins or by using larger area lights.

Figure 4.8 shows a monster on a ground plane with direct, one-bounce indirect, and two-bounce indirect lighting. Three levels of directional discretization are compared. One-bounce lighting is sufficient for a pleasing result. A coarse



(a) One Bounce, 6x12x12, 7.53FPS (b) One Bounce, 6x12x12, 7.53FPS

Figure 4.5: Flying dragon (deformable model, 2670 vertices).

directional discretization produces slight artifacts, but the speed gains are considerable. Table 4.1 details the time spent on the different steps of our algorithm for this particular scene. The initial creation of the geometric hierarchy takes about 72ms but is only done once in a preprocess.

Figure 4.10 compares a reference image (a) computed with path tracing to our method (b). The differences are minor. However, our method renders with several frames per second. We also demonstrate that there is virtually no difference between our hierarchical (b) and a brute-force non-hierarchical version (c). Coarsely tessellated objects can cause light leakage, see (d) where the teapot only has 792 vertices (992 triangles).

Our GPU implementation also supports glossy direct illumination, which we demonstrate in Figure 4.11. In order to maintain interactive frame rates, we limit indirect illumination to diffuse interreflections in our GPU implementation (even though the proposed method itself can handle glossy interreflections). Our results compare favorably to the reference solution (Figure 4.11a).

We have found that our algorithm has roughly a complexity of $O(N \log N)$, with N being the number of vertices, which is similar to other hierarchical radiosity methods.

4.4.1 Discussion

Despite the high-frame rate and the faithful reproduction of global illumination effects as documented by the ground truth comparisons, the proposed approxima-

(a) Direct Lighting, 6x12x12, 3.86FPS (b) One bounce, 6x12x12, 4.06 FPS

Figure 4.6: Shadow and indirect lighting effects between objects (with (a) 5165 vertices and (b) 4782 vertices).

Discretization	$6\times8\times8$	$6\times12\times12$	$6\times16\times16$
Update Elements	12ms (6%)	12ms (8%)	12ms (11%)
Create Hierarchy	23ms (21%)	43ms (29%)	78ms (38%)
Refine & Push	30ms (29%)	35ms (24%)	60ms (30%)
Illumination	35ms (35%)	40ms (28%)	48ms (24%)
Total	112ms	148ms	218ms

Table 4.1: Timings for the monster (3378 vertices, 1-bounce illumination, see Fig. 4.8).

tions and discretization may lead to visual artifacts. If only a coarse cube-map discretization of the directional hemisphere is used, block artifacts may be visible in the light simulation (e.g., Figure 4.7 and 4.8). However, using $6\times12\times12$ bins, we achieve a good compromise between speed and visual quality.

Additional inaccuracies may occur due to the uneven distribution of solid angles across bins. Furthermore, although the fixed world-space alignment of the bin cube-maps across all geometric hierarchy levels enables fast computation, differences in directional sampling for different surface element orientations may lead to inaccuracies and temporal aliasing when objects undergo deformations (see accompanying video).

Moreover, rendering quality depends on the initial triangulation of the models as we base our lighting simulation on the models' vertices. Starkly uneven triangulation may therefore require re-meshing to prevent artifacts. If a model is not tessellated finely enough, light leakage might occur, as not every bin can be filled with a link for accurate occlusions. However, for $6\times16\times16$ or fewer bins and

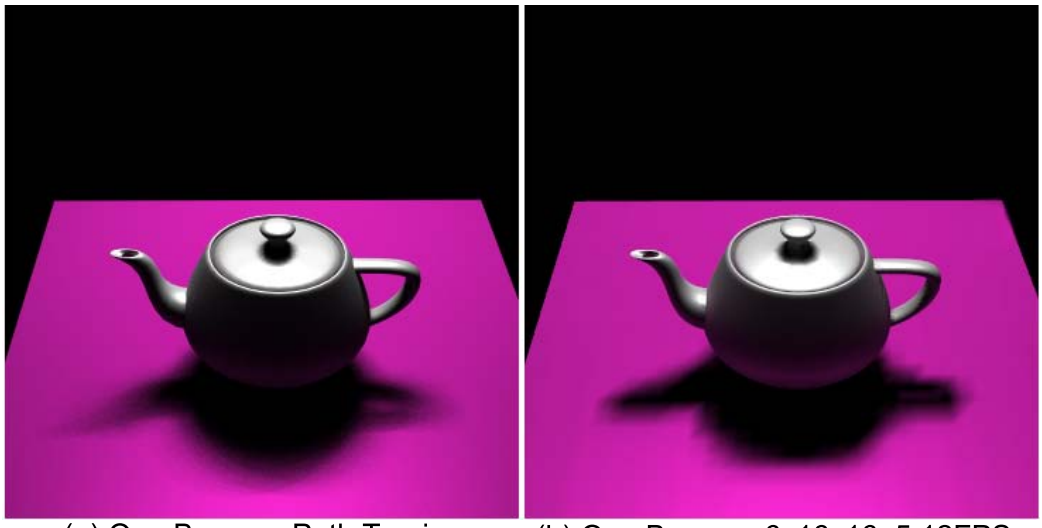


Figure 4.7: Discretization artifacts may become visible under area lighting. However, increasing the number of bins reduces artifacts.

models of about 5000 vertices, we have rarely encountered it.

Currently, we also trade rendering performance for accuracy and precompute the geometric hierarchy once, ignoring the fact that an adaptation of the hierarchy according to the deformation may be beneficial.

Despite these trade-offs and approximations, which are necessary to obtain high performance, the good visual quality of our results shows that interactive full global illumination on a single PC is feasible.

4.5 Summary

We presented a new global illumination method that builds on and extends the traditional hierarchical radiosity approach by implicitly computing visibility. This new concept circumvents time-consuming explicit visibility queries, the main performance bottleneck in traditional approaches. Our method allows for rendering of full global illumination solutions for moderately complex and arbitrarily deforming dynamic scenes at near-real-time frame rates on a single PC. It faithfully reproduces a variety of complex lighting effects including diffuse and glossy interreflections, and handles scenes featuring environment map and area light sources.

As part of future work, we plan to investigate explicit temporal coherence strategies to further improve animation quality. Decoupling the tessellation of the mesh from shading computation is another interesting line of research.

Figure 4.8: A monster rendered under environmental lighting with direct illumination, one-bounce, and two-bounce indirect illumination and varying discretizations (3378 vertices).

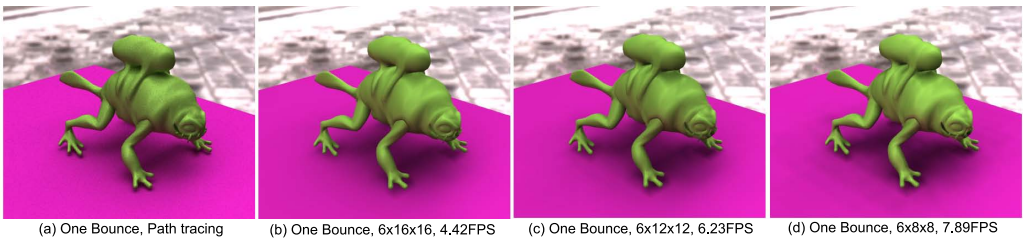


Figure 4.9: Frog with one-bounce indirect lighting and varying bin discretization (3495 vertices).

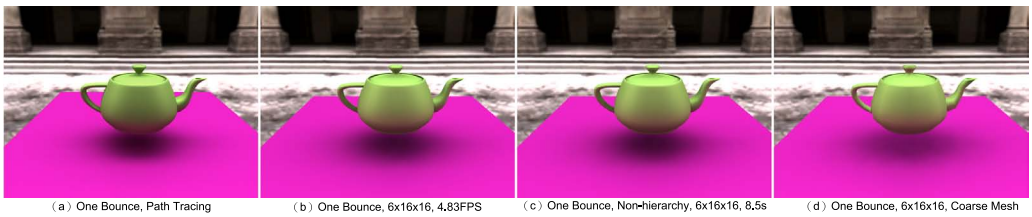


Figure 4.10: Environment lighting: (a) Ground truth using path tracing. (b) Non-hierarchical CPU implementation of our method. (c) GPU version of our algorithm. (d) Light leaking if mesh tessellation is too coarse (teapot: (a)-(c) 2582 vertices, (d) 792 vertices).

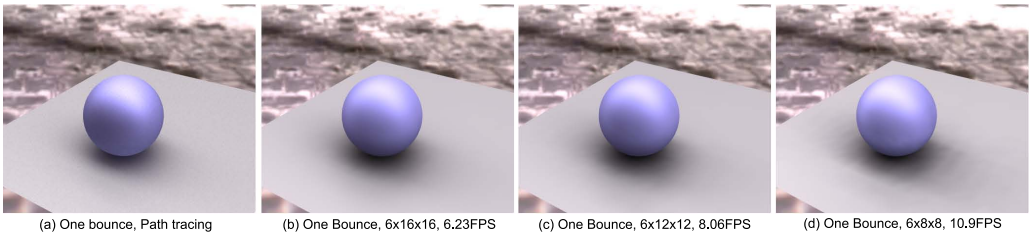


Figure 4.11: Glossy sphere (2278 vertices).

Part III

Pre-filtering Soft Shadow Maps and their Applications

Chapter 5

Real-time All-frequency Shadows In Dynamic Scenes

5.1 Introduction

Real-time, photo-realistic rendering of computer-generated scenes requires a high computational effort. One of the main bottlenecks is visibility determination between light sources and receiving surfaces, especially under complex lighting such as area light sources or environment maps.

Recent methods for rendering soft shadows from area lights operate in real-time, but either tend to be too intricate and expensive for rendering multiple light sources [Guennebaud06, Guennebaud07, Schwarz07], or break down for detailed geometry [Assarsson03]. Furthermore, these methods usually do not support environment map lighting. Other algorithms based on precomputation [Sloan02] are good at reproducing shadows from environment maps in static scenes, but have difficulties with fully dynamic objects, despite recent progress [Ren06].

The goal of our works in this chapter is to enable real-time, all-frequency shadows in completely dynamic scenes and to support area light sources as well as environment lighting. The key contribution is a very fast method for rendering plausible soft shadows which is so-called *convolution soft shadow mapping* (CSSM). CSSM requires only a constant-time memory lookup, thereby enabling us to render soft shadows at hundreds of frames per second for a single area source. Environment-lit scenes can be rendered from a collection of approximating area light sources. Even though shadows are only approximate, the results are virtually indistinguishable from reference renderings, but are produced at real-time frame rates.

5.2 Plausible Soft Shadows Using Convolution

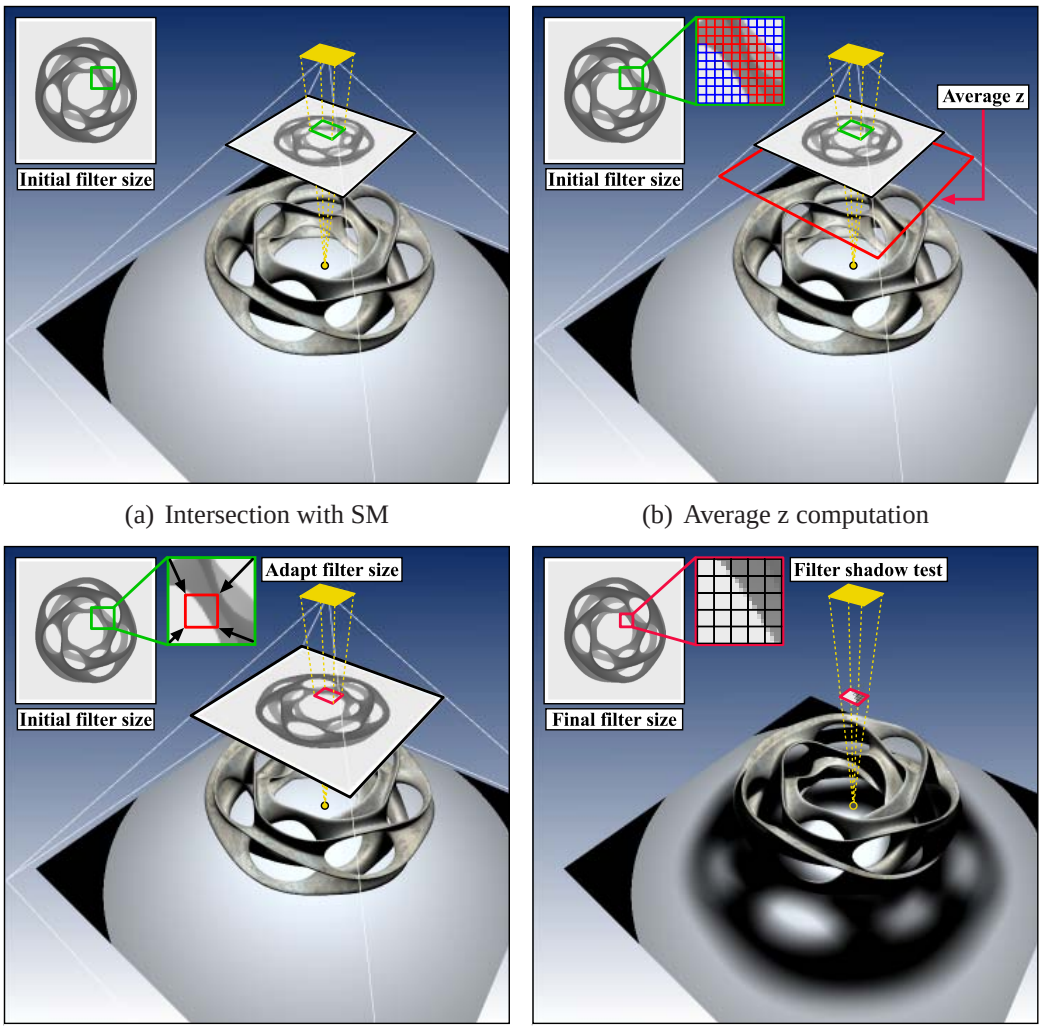
Rendering soft shadows for area light sources is challenging. Our goal is to render several area light sources in real-time without having to sacrifice visual quality. We argue that computing penumbrae at full physical accuracy is intractable in this case. Instead, reducing shadow accuracy slightly enables us to achieve very high frame rates while keeping the visual error at a minimum.

We build on convolution-based methods which simulate penumbrae by filtering shadows depending on the configuration of blocker, receiver, and light source [Soler98, Fernando05b]. These methods are approximate in general, but produce an exact solution if the light source, blocker, and receiver are planar and parallel [Soler98]. Fortunately, deviating from this geometric configuration still produces plausible results.

The advantage of computing shadows using convolution is two-fold: it is compatible with image-based representations, in particular shadow mapping [Williams78] and thus scales well to scenes with a high polygon count. Second, convolutions can be computed efficiently using the Fourier transform [Soler98], or even in constant time if the shadows have been prefiltered using mipmaps or summed area tables [Lauritzen07].

However, applying convolution to shadow maps in order to produce soft shadowing is not trivial. The size of the convolution kernel needs to be estimated based on the blocker distance [Soler98], but when multiple blockers at different depths are involved there is no single correct blocker distance. To get a reasonable approximation of blocker depth, we adopt the soft shadow framework of *percentage closer soft shadows* (PCSS) and firstly compute the average depth of the blockers over the support of the filter. This framework was introduced by Fernando [Fernando05b] and we have already introduced it in details in the Section. 2.4.2 of Chapter. 2. Unfortunately, estimating this average blocker depth is expensive since it (seemingly) requires averaging depths from the shadow map in a brute force fashion. The strength of our technique is that it allows for both efficient filtering of the shadows as well as efficient computation of the average blocker depth. Both of these operations can be expressed with the same mathematical framework, and will be described in Section. 5.2.1.

The main visual consequence of the average blocker depth approximation is that the penumbra width may not be estimated exactly (it is correct for the parallel-planar configuration described above though). We show that this approximation does not produce offensive artifacts, and even closely approximates the ground truth solution. Figure 5.1 presents an overview of our soft shadow method and will be detailed in the following section.



(a) Intersection with SM

(b) Average z computation

(c) Move SM to average z

(d) Filter shadow test

Figure 5.1: An overview of the CSSM method. First, an initial filter size is determined according to the cone defined by the intersection of the area light source, the shadow map plane, and the current receiver point (a). This filter size (green) is used to fetch the z_{avg} value from the prefiltered average z-textures. We then virtually place the shadow map plane at the z_{avg} and determine the final filter width (red) for soft shadow computation as shown in (c). In the last step, the incoming visibility value is looked up from the CSM texture (d).

5.2.1 Convolution Soft Shadows

As indicated above, soft shadows can be rendered efficiently through shadow map filtering and we therefore build on Convolution Shadow Maps (CSM) [Annen07]. As will be shown, CSM can be extended to also compute the average blocker depth, which is required to estimate penumbra widths. We also introduce extensions that allow us to safely reduce the approximation order to further push rendering performance.

Review In order to keep the discussion self-contained, we briefly review CSM. Let $\mathbf{x} \in \mathbb{R}^3$ be the world-space position of a screen-space pixel and the point $\mathbf{p} \in \mathbb{R}^2$ represents the corresponding 2D position of a shadow map pixel. The shadow map itself encodes the function $z(\mathbf{p})$, which represents the depth of the blocker that is closest to the light source for each $\mathbf{p}$, and $d(\mathbf{x})$ is the distance from $\mathbf{x}$ to the light source.

We define the shadow function $s()$, which encodes the shadow test, as:

$$s(\mathbf{x}) = f(d(\mathbf{x}), z(\mathbf{p})) := \begin{cases} 1 & \text{if } d(\mathbf{x}) \leq z(\mathbf{p}) \\ 0 & \text{if } d(\mathbf{x}) > z(\mathbf{p}). \end{cases} \tag{5.1}$$

If the function $f()$ is expanded into a separable series:

$$f(d(\mathbf{x}), z(\mathbf{p})) = \sum_{i=1}^{\infty} a_i(d(\mathbf{x}))B_i(z(\mathbf{p})), \tag{5.2}$$

we can spatially convolve the result of the shadow test through prefiltering:

$$\begin{aligned} s_f(\mathbf{x}) &= \big[w * f\big(d(\mathbf{x}), z\big)\big](\mathbf{p}) \\ &\approx \sum_{i=1}^{N} a_i\big(d(\mathbf{x})\big)\big[w * B_i\big](\mathbf{p}), \end{aligned} \tag{5.3}$$

where the basis images B_i are prefiltered with the kernel w, which in practice is achieved through mipmapping each $B_i(\mathbf{p})$ or computing summed area tables [Crow84]. At run-time, one only needs to weight the prefiltered basis images by $a_i(d(\mathbf{x}))$ and sum them up.

5.2.2 Estimating Average Blocker Depth

The above prefiltering of the shadow test results allows us to apply convolutions to soften shadow boundaries. However, for real soft shadows the size of the convolution kernel needs to vary based on the geometric relation of blockers and receivers [Soler98]. We follow Fernando [Fernando05b] and use the average depth value

z_{avg} of all blockers that are *above* the current point $\mathbf{x}$ to adjust the size of the kernel.

Estimating the average blocker depth appears to be a very expensive operation. The obvious solution of sampling a large number of shadow map texels in order to compute the average depth value z_{avg} is very costly, and achieving good frame rates for large convolution kernels is not only difficult [Fernando05b] but also counterproductive for constant time filtering methods [Donnelly06b, Annen07, Lauritzen07].

The key insight into making this step efficient is that this selective averaging can be expressed as a convolution and can therefore be rendered efficiently. To see this, let us first compute a simple local average of the z-values in the shadow map:

$$z_{avg}(\mathbf{x}) = \left[w_{avg} * z\right](\mathbf{p}). \tag{5.4}$$

Here, w_{avg} is a (normalized) averaging kernel. However, we only want to average values that are smaller than $d(\mathbf{x})$. Let us therefore define a "complementary" shadow test $\bar{f}$:

$$\bar{f}(d(\mathbf{x}), z(\mathbf{p})) = \begin{cases} 1 & \text{if } d(\mathbf{x}) > z(\mathbf{p}) \\ 0 & \text{if } d(\mathbf{x}) \leq z(\mathbf{p}), \end{cases} \tag{5.5}$$

which returns 1 if the shadow map z-value $z(\mathbf{p})$ is smaller than the current depth $d(\mathbf{x})$, and 0 otherwise. We can now use this function to "select" the appropriate z samples by weighting them:

$$z_{avg}(\mathbf{x}) = \frac{\left[w_{avg} * \left[\bar{f}\left(d(\mathbf{x}), z\right) \times z\right]\right](\mathbf{p})}{\left[w_{avg} * \bar{f}\left(d(\mathbf{x}), z\right)\right](\mathbf{p})}. \tag{5.6}$$

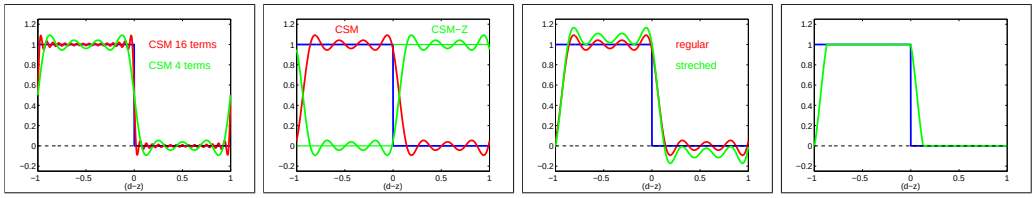
The denominator normalizes the sum such that it remains an average and is simply equal to the complementary filtered shadow lookup: $1 - s_f(\mathbf{x})$. For the numerator we can approximate the product of the complementary shadow test and z using the same expansion as used in regular CSM:

$$\bar{f}\left(d(\mathbf{x}), z\right) z = \sum_{i=1}^{N} \bar{a}_i\left(d(\mathbf{x})\right) \bar{B}_i\left(z(\mathbf{p})\right) z(\mathbf{p}). \tag{5.7}$$

Here, coefficients $\bar{a}_i$ are coefficients and $\bar{B}_i$ basis images for $\bar{f}$. We can now approximate the average as:

$$z_{avg}(\mathbf{x}) = \frac{1}{1 - s_f(\mathbf{x})} \sum_{i=1}^{N} \bar{a}_i\left(d(\mathbf{x})\right) \left[w_{avg} * \left[\bar{B}_i(z)\, z\right]\right](\mathbf{p}). \tag{5.8}$$

We will therefore compute new basis images $\left[\bar{B}_i\left(z(\mathbf{p})\right) z(\mathbf{p})\right]$ alongside the regular CSM basis images. We refer to this new approach for computing the



(a) CSM 4 and 16 terms (b) CSM vs CSM-Z (c) Shift and Scale (d) Final Clamp

Figure 5.2: Fourier series expansion. (a) depicts the difference between a 16- and 4-term reconstruction. (b) CSM and CSM-Z are exactly opposite to each other. Ringing suppression is possible with appropriate scaling and shifting (c), followed by clamping the function to $[0,1]$ (d).

average blocker depth as CSM-Z. See the appendix for a full derivation of the convolution formula for z_{avg}.

Initializing Average Depth Computation When we want to estimate or approximate the penumbra size for a given camera sample we have to do this by finding the area over the shadow map over which we will perform the z_{avg} computation. A first idea is to intersect the frustum formed by the camera sample $\mathbf{x}$ in 3D and the virtual area light source geometry with the shadow map plane (as depicted in Figure 5.1(a)). Unfortunately, there is no clear definition of such a plane, as the shadow map itself only represents a height field and does not have a certain plane location. We have found the near plane to work well for all our results. Please check the corresponding detailed introductions in the Section. 2.4.2 of Chapter. 2. However, an iterative procedure is possible where one re-adjusts the location after an initial z_{avg} has been found.

5.2.3 CSM Order Reduction

Annen et al. [Annen07] propose to expand f using a Fourier series. Unfortunately, this series is prone to ringing artifacts and the shadows at contact points may appear too bright unless a high order approximation is used as shown in Figure 5.2(a). We propose two changes that allow us to reduce the order significantly. First, we notice that with appropriate scaling, shifting, and subsequent clamping, ringing can be avoided completely. Figure 5.2 illustrates this. Scaling and shifting $f(d,z)$ such that ringing only occurs above 1 and below 0 is shown in (c). Whenever the function $f(d,z)$ is reconstructed we clamp its result to $[0,1]$, avoiding any visible artifacts (d).

A second problem with a low order series is that the slope of the reconstructed shadow test is not very steep when $(d-z) \approx 0$, as can be seen in Figure 5.2(d), and yields shadows that are too bright near contact points. A simple solution is to apply a non-linear transformation $G(v) = v^p$ to the filtered shadow value $s_f(\mathbf{x})$

with $p \geq 1$. This tends to darken the shadows and thus hides light leaking. If $p = 1$, nothing changes. On the downside, darkening also removes smooth transitions from penumbra regions, so we want to only apply it where necessary. When $d(\mathbf{x}) - z_{avg}(\mathbf{p})$ is small, we know that $\mathbf{x}$ is near a contact point where leaking will likely occur. Fortunately, this is also where penumbrae should be hard anyway. We therefore compute an adaptive exponent p based on this difference:

$$p = 1 + A\, exp\big(-B\,\big(d(\mathbf{x}) - z_{avg}(\mathbf{p})\big)\big). \tag{5.9}$$

A controls the strength of the darkening, and B determines the maximal distance of z_{avg} from the receiver point for which darkening is applied to. Figure 5.3 shows this effect for a varying parameter B.

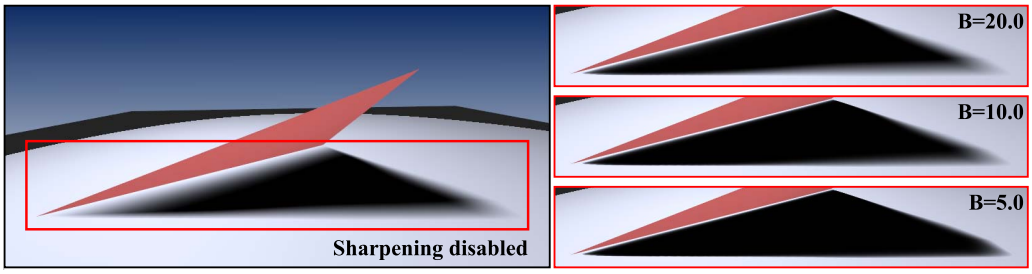


Figure 5.3: An illustration of the impact of sharpening parameters A and B. A is fixed to 30.0, whereas B is set to 5.0, 10.0, and 20.0 showing how B changes the spatial extend of the sharpening.

5.3 Illumination with Soft Shadows

5.3.1 Rendering Prefiltered Soft Shadows

Generating soft shadows with our new algorithm is similar to rendering anti-aliased shadows [Annen07]. First, the scene is rasterized from the center of the area light source and the z-values are written to the shadow map. Based on the current depth map two sets of images are produced: the Fourier series basis and its complementary basis images multiplied by the shadow map z-values.

After we have generated both data structures, we can run the prefilter process. Note that when the convolution formula from Eq. 5.8 is evaluated using a Fourier series, it also requires prefiltering the shadow map due to the constant factor when multiplying $\bar{f}()$ by $z(\mathbf{p})$ (see appendix). In our implementation, we support image pyramids (mipmaps) and summed-area-tables. Other linear filtering operations are applicable as well. When filtering is complete, we start shading the scene

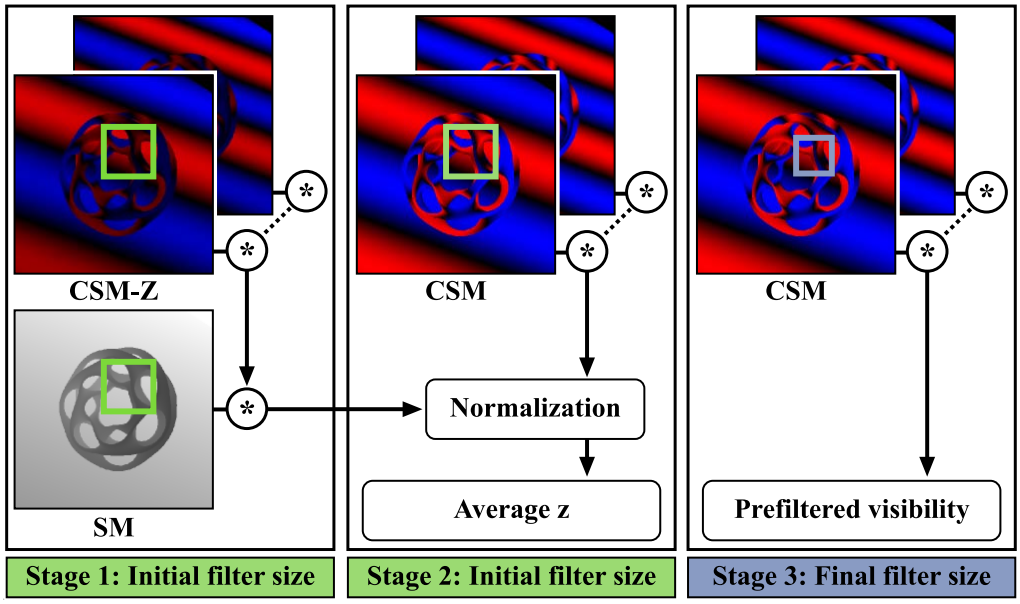


Figure 5.4: Convolution soft shadows pipeline. Stage 1 reconstructs a prefiltered z_{avg}. The z_{avg} is passed to the 2nd stage for normalization. Thereafter, the final filter size is computed as described in 5.1(c), and the visibility is evaluated by a regular CSM reconstruction.

from the camera view and employ convolution soft shadows for high-performance visibility queries. An overview of the different steps is given in Figure 5.4.

For each camera pixel we first determine an initial filter kernel width as previously shown in Figure 5.1(a) to estimate the level of filtering necessary for the pixel's 3D position and feed this to stages one and two. Stage one reconstructs the average blocker depth based on the prefiltered CSM-Z textures and the prefiltered shadow map, which is then passed to the second stage for normalization. After normalization, the final filter kernel width f_w is adjusted according to the spatial relationship between the area light source and the current receiver. In particular, the triangle equality tells us the filter width: $f_w = \frac{\Delta}{d} \cdot \frac{(d - z_{avg})}{z_{avg}} \cdot z_n$, where Δ is the area light source width, d is the distance from $\mathbf{x}$ to the light source, and z_n is the light's near plane. The filter width f_w is then mapped to the shadow map space by dividing it by $2 \cdot z_n \cdot \tan(\frac{fovy}{2})$. A final lookup into the CSM textures yields the approximate visibility we wish to compute for the current pixel.

All three stages together require only six RGBA and one depth texture access (for a reconstruction order $M = 4$).

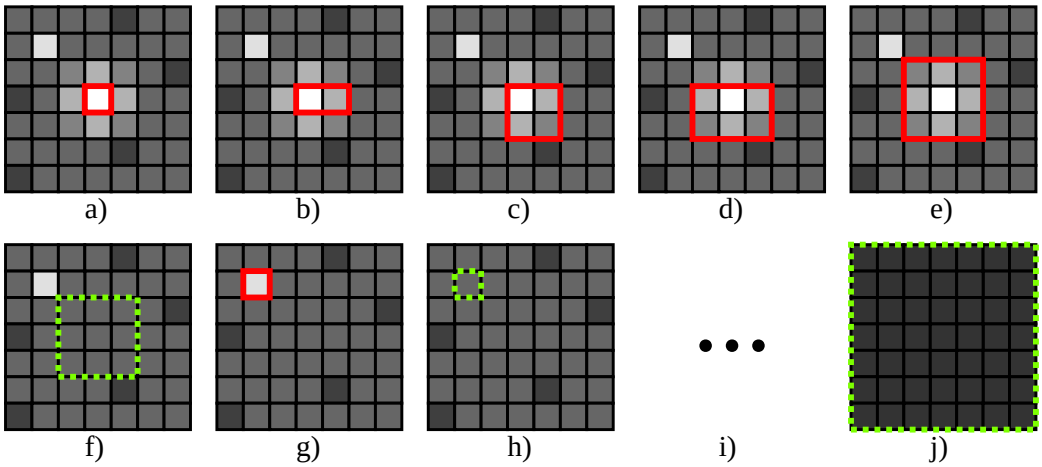
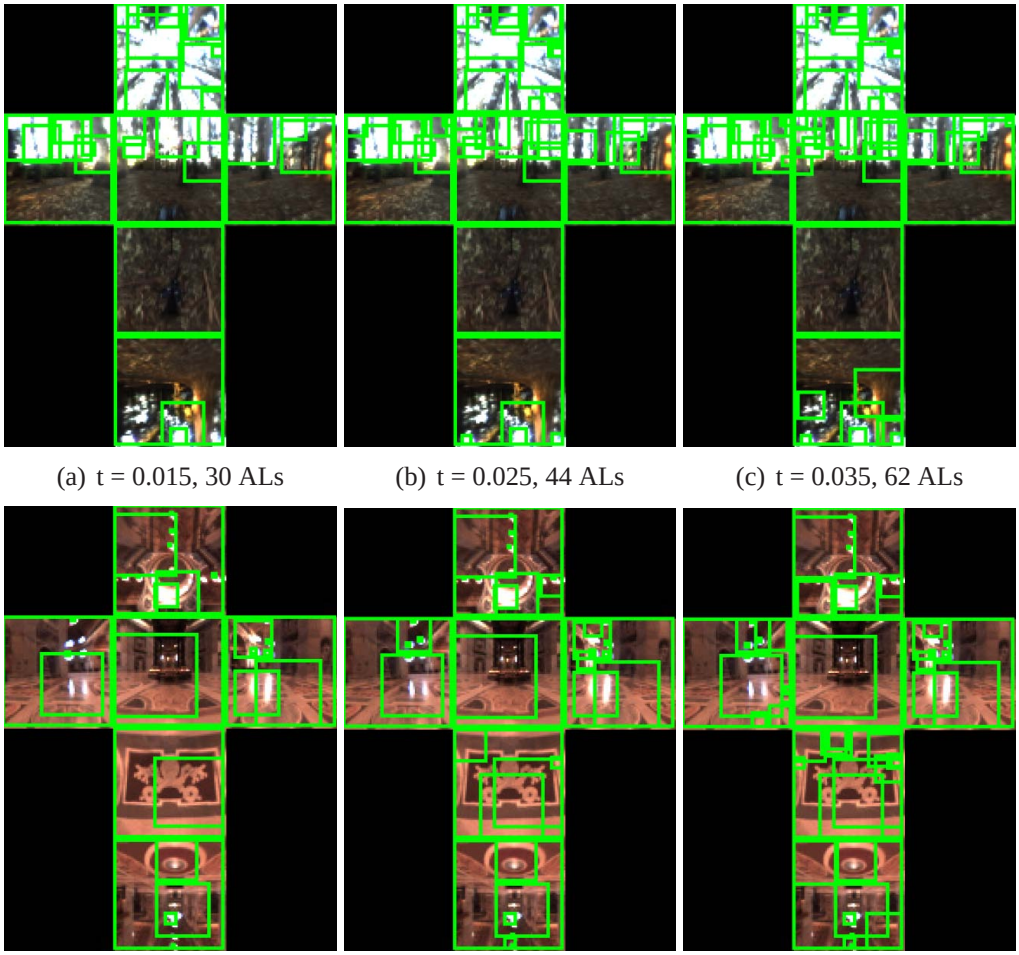


Figure 5.5: Fitting area lights to a cube map face. We first fit a 1×1 area light to the brightest pixel (a). In turn, we try to enlarge the area light at each side until a stopping criteria is reached (b)-(e). We remove the energy for this area light (but leave some amount to blend it with the area around it) (f), and continue with fitting more area lights (g)-(i), until we have area lights covering the whole face.

5.3.2 Generation of Area Lights for Environment Maps

We propose the following greedy algorithm for decomposing an environment map into a number of area light sources. We assume the environment map to be given as a cube map and proceed by decomposing each cube map face separately.

The process works directly in the 2D domain of the cube map face. We first find the pixel with the largest amount of energy and create a preliminary 1×1 area light for it. We then iterate over all four sides of the area light and try to enlarge each side by one pixel. The area light is enlarged if the ratio between the amount of energy E_{delta} that would be added to the light by enlarging it and the amount of energy E_{total} that the enlarged area light would emit is larger than a given threshold t. We repeat this enlargement process until none of the four sides can be enlarged, or the area light covers the complete cube map face. After the enlargement process has stopped, we remove the energy of this portion of the cube map face but leave a residual amount of energy to enable better fits in later iterations and create the final area light for it. The residual amount equals the average amount of energy adjusted to the size of the area. We then continue with fitting more area lights until we have covered the whole cube map face. Figure 5.5 illustrates the process. Note that our method may produce overlapping area lights. The parameter t determines the total number of light sources fitted to each cube map face. Examples are shown in Fig. 5.6.



(a) t = 0.015, 30 ALs (b) t = 0.025, 44 ALs (c) t = 0.035, 62 ALs

(d) t = 0.015, 32 ALs (e) t = 0.025, 45 ALs (f) t = 0.035, 60 ALs

Figure 5.6: A close-up of the area light decomposition for two different environment maps. The threshold values t are given.

5.4 Limitations and Discussion

Failure Cases Our technique shares the same failure cases as PCF-based soft shadowing [Fernando05b]. We assume that all blockers have the same depth within the convolution kernel (essentially flattening blockers), similar to Soler and Sillion's method [Soler98]. This assumption is more likely to be violated for larger area lights. Nevertheless, shadows look qualitatively similar to the reference rendering, as shown in see Figure 5.7. The use of a single shadow map results in incorrect shadows for certain geometries. This problem is commonly referred to as "single silhouette artifacts", which we share with many other techniques [Assarsson03, Guennebaud06].

Average Z Computation Computing the average z-value as described is prone

to inaccuracies due to the approximations introduced by CSM-Z and CSM. These possible inaccuracies may lead to visible artifacts due to the division by $1 - s_f(\mathbf{x})$. Care must be taken to use the very same expansion for CSM-Z and CSM in order to avoid such artifacts.

Ringing Suppression Our proposed ringing suppression using scaling and shifting followed by clamping does indeed reduce ringing and improves shadow darkness near contact points, but also sharpens shadows slightly as can be seen in Figure 5.9. However, this process is necessary to keep frame rates high as it allows the use of fewer terms in the expansion and the differences are barely noticeable. See the comparisons in the results section, all of which are rendered using ringing suppression.

Mipmaps vs. Summed Area Tables The quality that our method can achieve depends on the prefiltering process. Mipmaps are computationally inexpensive, but their quality is inferior compared to SATs as they re-introduce aliasing again at higher mipmap levels. However, SATs require more storage due to the need to use floating point textures [Hensley05] especially when using many area lights. In the case of multiple area lights, as used for environment mapping, artifacts are masked and mipmapping is a viable option. Figure 5.8 compares both solutions.

Textured Light Sources Our method cannot handle textured light sources directly as the prefiltering step cannot be extended to include textures. Instead, we decompose complex luminaires such as environment maps into uniform area lights.

Rectangular Area Lights Rectangular lights are supported, which is especially easy when using SATs. They can also be used in conjunction with mipmapping if the GPU supports anisotropic filtering. The aspect ratio of the area lights is limited by the maximum anisotropy the GPU allows. The increased cost of anisotropic filtering might warrant the use of several square area lights instead. The fitting process described in the last section can be modified to fit square area lights instead of rectangular ones. In fact, this is what we have used for our results.

BRDFs We do not support integrating the BRDF across the light source domain, similar to most other fast soft shadowing techniques. However, for environment map rendering we do evaluate the BRDF in the direction of the center of each area light and weight the contribution accordingly.

5.5 Results

In this section we report on the quality and performance of our method. Our technique was implemented in DirectX 10 and all results were rendered on a Dual-Core AMD Opteron with 2.2GHz using an NVIDIA GeForce 8800 GTX graphics card. Our performance timings are listed in Table 5.1.

	# Area Lights			
SM Type	1	10	20	40
MM: 128^2	258 fps	48 fps	28 fps	18 fps
MM: 256^2	228 fps	44 fps	25 fps	15 fps
MM: 512^2	189 fps	38 fps	20 fps	13 fps
MM: $1K^2$	110 fps	24 fps	5 fps	-
SAT: 128^2	128 fps	15 fps	8.8 fps	-
SAT: 256^2	110 fps	13 fps	7.5 fps	-
SAT: 512^2	89 fps	11 fps	6.0 fps	-
SAT: $1K^2$	52 fps	3 fps	1.5 fps	-

Table 5.1: Frame rates for the Buddha scene with 70k faces from Figure 5.10, rendered using reconstruction order $M = 4$. For many lights and high resolution shadow maps, our method may require more than the available texture memory (reported as missing entries).

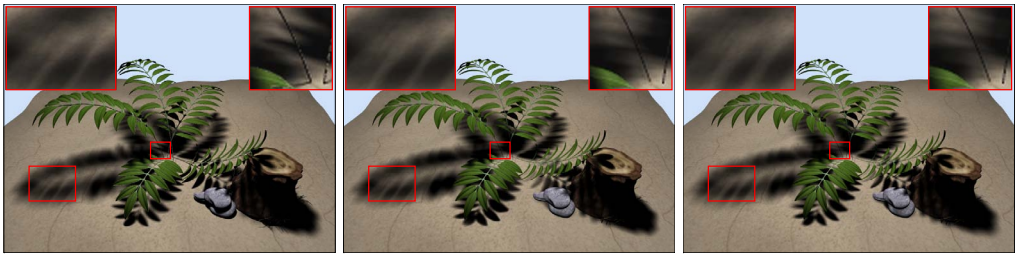
The first result shown in Figure 5.7 compares the shadow quality of several different algorithms to a reference rendering. We analyze two situations in particular, large penumbrae and close-contact shadows (see close-ups). Shadows rendered with our new technique are very close to the reference, bitmask soft shadows perform slightly better at contact shadows and backprojection methods tend to overdarken shadows when the depth complexity increases. Percentage closer soft shadows produce banding artifacts in larger penumbra regions due to an insufficient number of samples.

The overall performance of our technique and its image quality depend on the choice of prefiltering, the number of area lights, and the individual light's shadow map size. The next results illustrate the impact of these individual factors.

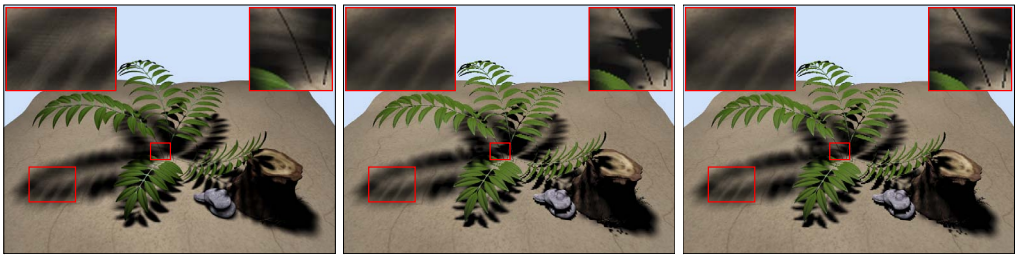
We begin with a side-by-side comparison between mipmap- and SAT-based soft shadows in Figure 5.8. Mipmaps produce less accurate results compared to summed-area-tables for rendering single lights due to aliasing artifacts. For complex lighting environments, however, shadows from many light sources are averaged, which makes mipmapping artifacts less noticeable (Fig. 5.10 and 5.11).

Figure 5.9 illustrates the influence of the reconstruction order and sharpening. We render a foot bone model of high depth complexity and demonstrate the effect of the sharpening function $G()$. While contact shadows (toe close-up) are darkened and slightly sharper than the results rendered with $M = 16$, their larger penumbra areas are not influenced, which maintains the overall soft shadow quality.

Figure 5.10 shows the influence of the number of light sources used for approximating the environment map. Below the renderings we show the fitted area light sources and a difference plot. Rendering with 30 lights (Fig. 5.10(d)) already



(a) Ground Truth (Ray-Traced) (b) Our Method – SAT 4 Terms (60 fps) (c) Our Method – SAT 16 Terms (23 fps)



(d) Percentage Closer Soft Shadows (18 fps) (e) Backprojection (41 fps) (f) Bitmask Soft Shadows (19 fps)

Figure 5.7: Shadow quality comparison of several methods (SM was set to 512×512, scene consists of 212K faces): ray-tracing (a), our method using SATs – 4 terms (b) and 16 terms (c), percentage closer soft shadows [Fernando05b] (d), backprojection [Guennebaud06] (e), and bitmask soft shadows [Schwarz07] (f).

looks quite similar to the reference but some differences are noticeable. With 45 area lights, the differences to the reference are significantly reduced and the result is visually almost indistinguishable. This example illustrates that mipmapping produces adequate results, while offering a more than threefold speedup compared to summed-area tables (see Figure 5.11). The reference images in Figure 5.10 and 5.11 have been generated with 1000 environment map samples [Ostromoukhov04] using ray tracing. Figure 5.11 also compares brute force GPU-based shadow rendering with 500 samples, which achieves a much slower frame rate compared to our method.

Concerning memory consumption, mipmaps (SATs) with $M = 4$ require two 8bit (32bit) RGBA textures for storing the CSM and two 16bit (32bit) RGBA textures for storing the CSM-Z basis values.

5.6 Summary

We have presented an efficient soft shadow algorithm that enables rendering of all-frequency shadows in real-time. It is based on convolution, which does not re-

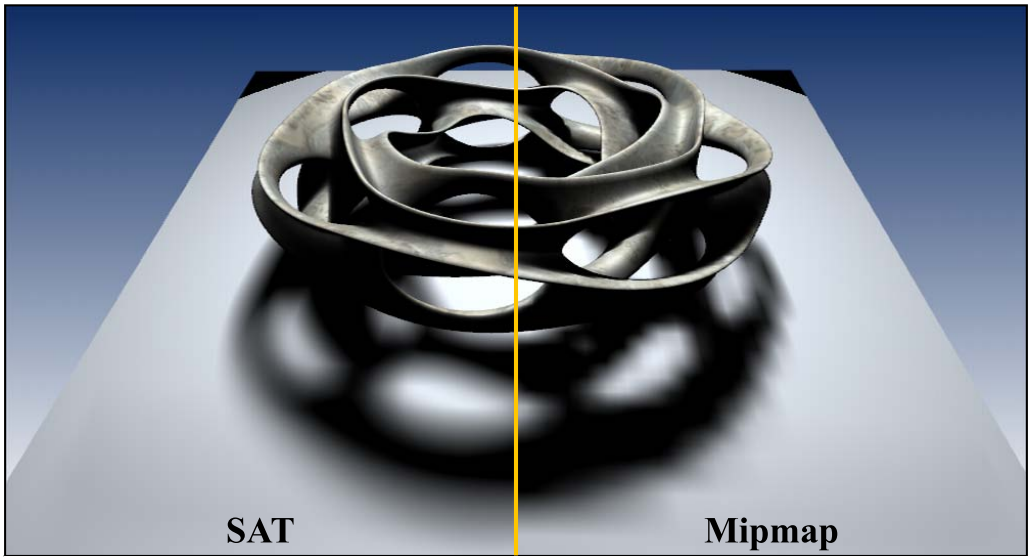


Figure 5.8: The difference in filter quality when using a summed-area-table (left) and a mipmap (right). Successive down sampling with a 2×2 box-filter introduces aliasing at higher mipmap levels.

quire explicit multiple samples and can therefore be carried out in constant time. It is fast enough to render many area light sources simultaneously. We have shown that environment map lighting for dynamic objects can be incorporated by decomposing the lighting into a collection of area lights, which are then rendered using our fast soft shadowing technique. The efficiency of our algorithm is in part due to some sacrifices in terms of accuracy. However, our new soft shadow method achieves plausible results, even though they are not entirely physically correct. As future work, we intend to explore the use of area lights for indirect illumination, which could be an important step toward interactive global illumination for fully dynamic scenes.

Appendix

Our z_{avg} computation uses the Fourier series [Annen07] to approximate $\bar{f}()$ and yields the following:

$$\bar{f}(d(\mathbf{x}), z(\mathbf{p})) \approx \frac{1}{2} + 2 \sum_{k=1}^{M} \frac{1}{c_k} \sin\left[c_k(d(\mathbf{x}) - z(\mathbf{p}))\right], \tag{5.10}$$

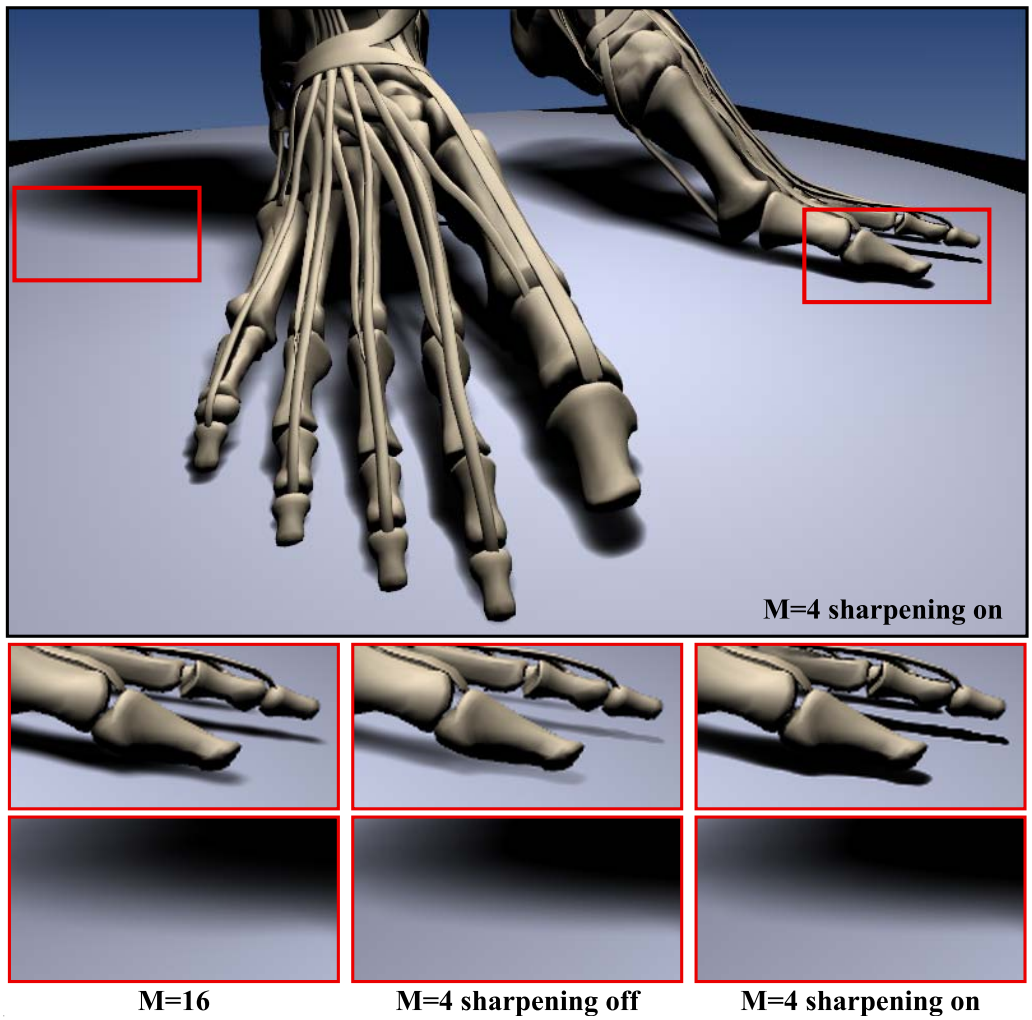
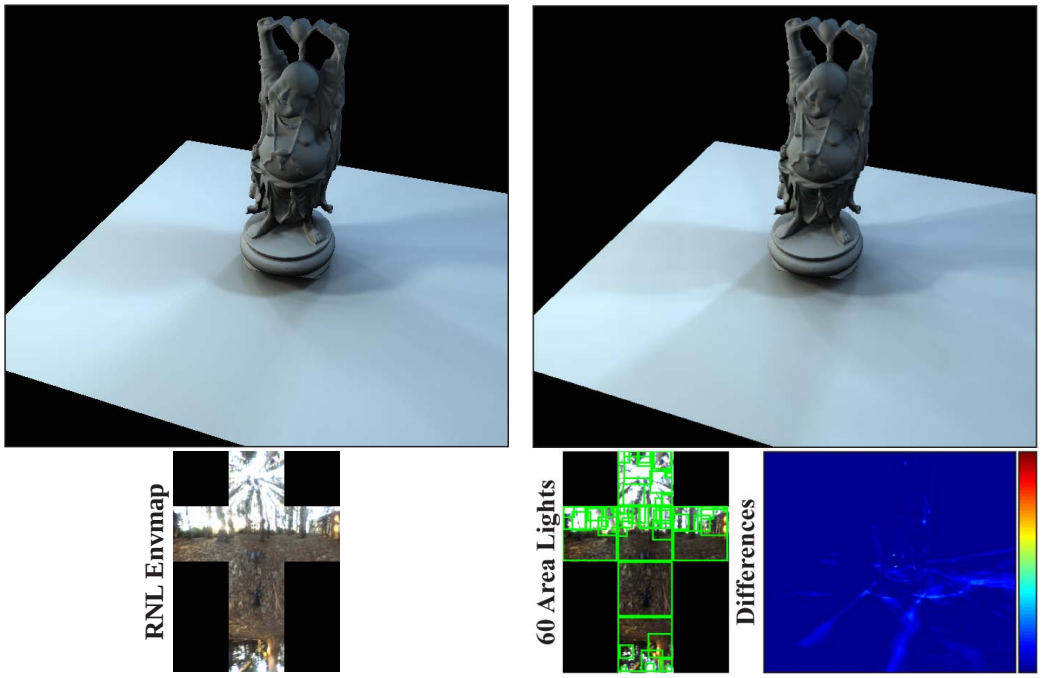


Figure 5.9: Influence of reconstruction order M and sharpening. The close-ups show that shadow darkening is restricted to contact points whereas larger penumbra areas remain unaffected and smooth.

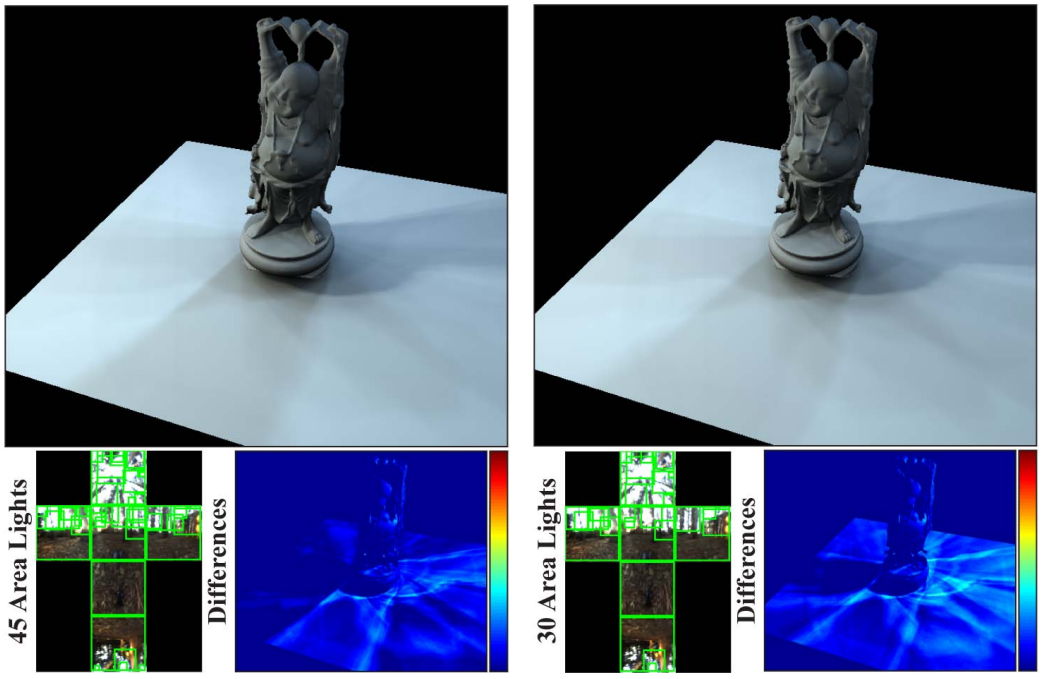
with $c_k = \quad (2k-1)$. Then the convolution from Eq. 8 becomes:

$$\begin{aligned} z_{avg}(\mathbf{x}) &\approx \frac{1}{1-s_f(\mathbf{x})}\left[w_{avg} * \left(\frac{1}{2} + \sum_{k=1}^{M} \frac{2}{c_k}\sin\left[c_k(d(\mathbf{x})-z)\right]\right) z\right](\mathbf{p}) \\ &\approx \frac{1}{1-s_f(\mathbf{x})}\left[w_{avg} * \frac{z}{2} + \frac{2}{c_k}\sum_{k=1}^{M} \sin\left(c_k d(\mathbf{x})\right)\left(w_{avg} * z\cos(c_k z)\right) - \right. \\ &\qquad \left. \frac{2}{c_k}\sum_{k=1}^{M} \cos\left(c_k d(\mathbf{x})\right)\left(w_{avg} * z\sin(c_k z)\right)\right](\mathbf{p}). \end{aligned} \tag{5.11}$$

This means there is an additional basis image containing $z/2$ values (basically corresponding to a shadow map, see Figure 5.4), which needs to be filtered.

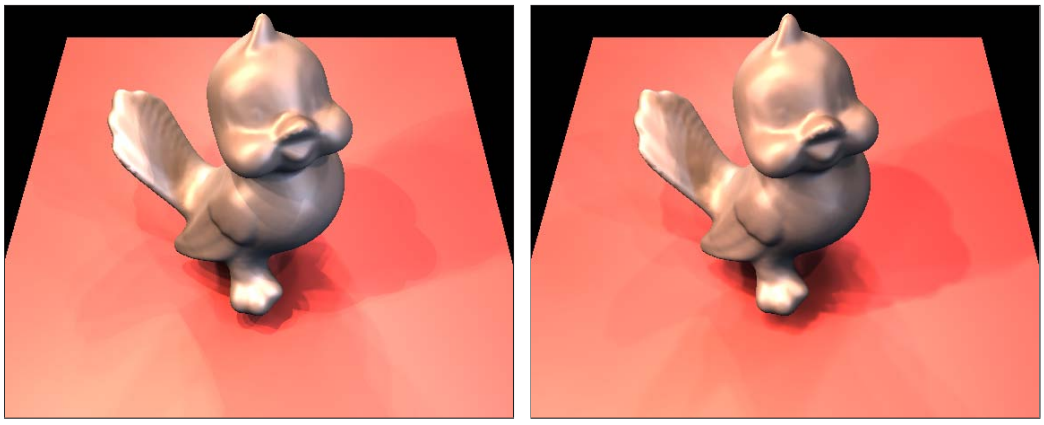


(a) RT SMs (b) 60 ALs, $t = 0.035$ (9.8 fps)

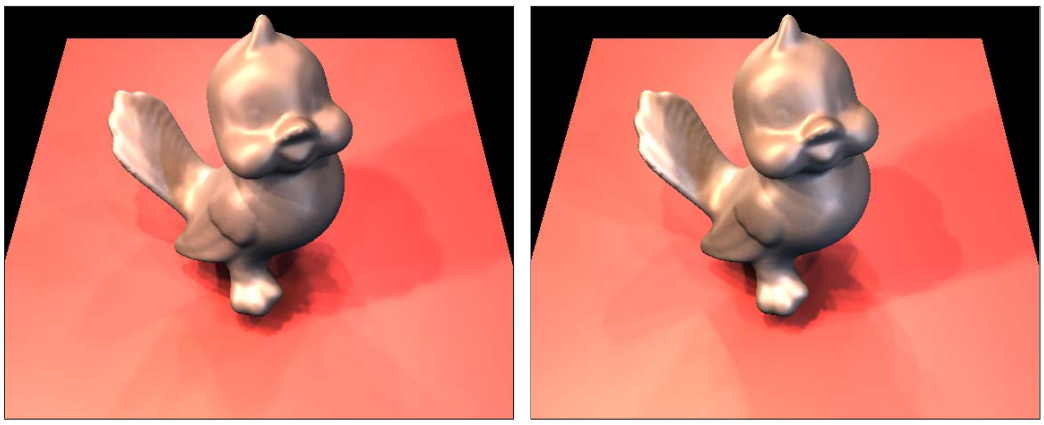


(c) 45 ALs, $t = 0.025$ (14.1 fps) (d) 30 ALs, $t = 0.015$ (18.4 fps)

Figure 5.10: Comparison between ray-tracing 1000 point lights (a), our technique with mipmaps using 60 (b), 45 (c), and 30 (d) area light sources. Each image shows the environment map with the the fitted light sources in green. SM resolution was set to 256×256**.**



(a) Ray-Tracing (b) SAT 30 ALs, 7.5 fps



(c) Mipmap 30 ALs, 25.4 fps (d) GPU SMs, 1.4 fps

Figure 5.11: In this figure we compare our rendering results with 30 ALs (St.Peters Basilica EM) against ray-tracing 1000 point lights and standard GPU-based shadow mapping. (a) ray-tracing, (b) our technique with SATs (c), our technique with mipmaps, and (d) GPU-based shadow mapping which achieves similar quality (500 shadow maps). SM resolution was set to 256×256**.**

Chapter 6

Real-time Indirect Illumination with Clustered Visibility

6.1 Introduction

Realizing that visibility is often the bottleneck in the GI methods, fast visibility determination of the indirect illumination has raised considerable interests. Instant Radiosity [Keller97] is one option to speed up visibility without imposing many restrictions on the scene. Here, indirect light is approximated with a number of *virtual point lights* (VPLs). Visibility between these VPLs and the rest of the scene can be efficiently computed using shadow maps and recent graphics hardware (GPUs). However, currently it is not feasible to generate shadow maps for every VPL as required by non-trivial scenes (e.g. in a computer game) at real-time frame-rates. We propose a solution to tackle this problem by introducing *virtual area lights* (VALs). Instead of using a traditional VPL-based instant radiosity algorithm, we cluster the VPLs into a small number of VALs. Visibility between these few VALs and the scene is computed with a very fast soft shadowing technique instead of using hard shadows for a large number of VPLs.
Our contributions in this chapter include:

- A temporally coherent GPU-based method to cluster a large number of VPLs into a small number of VALs.
- A fast method to render soft shadows from VALs.
- A method to combine illumination from VPLs and visibility from VALs that allows one-bounce global illumination for moderately complex and fully dynamic scenes at interactive to real-time frame rates.

This chapter is organized as follows: we describe our approach in Section 6.2. The Instant Radiosity method and its extension to clustered visibility is described in Section 6.3. Details about the clustering algorithm are given in Section 6.4, followed by our GPU implementation in Section 8.4. We show our results in Section 6.6 before we conclude in Section 8.6.

6.2 Overview

Our goal is to efficiently compute illumination from a large number of virtual points lights (VPLs). Such VPLs are used to simulate global illumination [Keller97] and can be efficiently generated using reflective shadow maps [Dachsbacher05]. To compute visibility for every VPL, shadow mapping [Williams78] is popular but has two limitations: the entire scene geometry has to be processed (transformed, clipped, etc.) for every VPL and the total number of depth map pixels is limited. In recent work, visibility was therefore ignored [Dachsbacher05, Dachsbacher06], approximated [Ritschel08b] or sped up by exploiting temporal coherence [Laine07].

To enable real-time global illumination, we propose to *approximate visibility* by *clustering* the VPLs. Although this significantly reduces the number of required shadow maps, simply drawing a hard shadow for each cluster would result in banding artifacts in penumbra regions. We therefore exploit recent advantages in the computation of real-time soft shadows, i.e. each group of VPLs is treated as one VAL which produces a soft shadow. For the final rendering, we still use all VPLs to illuminate the receiver point, however, visibility is computed from a few VALs only. Fig. 6.1 shows an overview of our algorithm. Note that we use the VPLs only for indirect illumination in this work. Other possible uses of VPLs, such as for environment map lighting, are not considered here.

6.3 Instant Radiosity with Clustered Visibility

To compute the global illumination at a point $\mathbf{x}$, instant radiosity approximates the reflected radiance $L(\mathbf{x},\omega_o)$ in direction ω_o with a set of N VPLs, each carrying a radiant flux Φ_i as

$$L(\mathbf{x},\omega_o) = \sum_{i=1}^{N} L_i(\mathbf{x},\omega_o) V(\mathbf{p}_i,\mathbf{x}),$$

where

$$L_i(\mathbf{x},\omega_o) = f_r(\mathbf{x},\omega_i,\omega_o)\frac{\Phi_i \cos(\theta_i)\cos(\theta_x)}{d_i^2(\mathbf{x})},$$

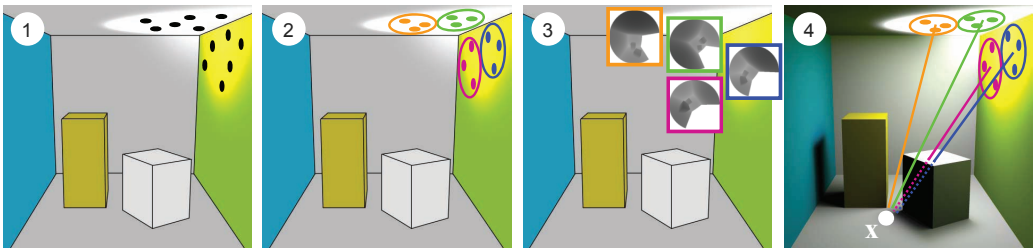


Figure 6.1: Overview of our algorithm: First, a set of N VPLs is generated to represent the indirect light. In the second step, VALs are generated by grouping the VPLs into M clusters. Next, one (soft) shadow map is rendered for each VAL. The final step is the rendering: The receiver point x is illuminated by all VPLs. Instead of computing a visibility value for each VPL, only M (fractional) visibility values are computed and shared within each cluster. To avoid banding, each cluster generates a soft shadow, so the penumbra region is composed of soft shadows.

$d_i(\mathbf{x})$ is the distance between VPL i and receiver, Θ and Θ_x are the angles between VPL i and receiver normal and the transmission direction. V is the binary visibility term between $\mathbf{x}$ and the VPL position $\mathbf{p}_i$. $f_r(\mathbf{x},\Omega_i,\Omega_o)$ is the BRDF at position $\mathbf{x}$ from direction Ω_i to VPL i in direction Ω_o. $\frac{\Phi_i}{\pi}$ is the radiant intensity of VPL i, assuming a Lambertian sender.

Next, the general visibility V (which is more suitable for raytracing [Wald03]) is replaced with visibility $\bar{V}_i$ from VPLs only (which is more suitable for GPUs using shadow maps):

$$L(\mathbf{x},\Omega_o) = \sum_{i=1}^{N} L_i(\mathbf{x},\Omega_o)\bar{V}_i(\mathbf{x}).$$

To accelerate the visibility computation we group the N VPLs into a much lower number of M clusters (VALs) and compute the (now fractional) visibility only for individual VALs:

$$L(\mathbf{x},\Omega_o) \approx \sum_{i=1}^{N} L_i(\mathbf{x},\Omega_o)\tilde{V}_{\mathscr{C}(i)}(\mathbf{x}).$$

Here we use a mapping function $\mathscr{C} : [1\ldots N] \rightarrow [1\ldots M]$ which maps the VPL i to the corresponding virtual area light $\mathscr{C}(i)$. Details on the creation of $\mathscr{C}$ are found in Section. 6.4. Instead of computing the visibility $\bar{V}_i(\mathbf{x})$ between VPL i and $\mathbf{x}$, an approximation $\tilde{V}_{\mathscr{C}(i)}(\mathbf{x})$ between the VAL $\mathscr{C}(i)$ and $\mathbf{x}$ is used. This clustering of visibility is based on the insight that indirect light typically contains few high frequencies and estimates can be used without much perceptual difference

[Ritschel08b]. Please note that each receiver point is still illuminated from all N VPLs, only the number of visibility computations is reduced to M.

6.3.1 Convolution Soft Shadow Maps

To approximate the visibility of one of the M virtual area lights, any soft shadow algorithm can be used (see [Hasenfratz03a] for a recent survey). Due to its high rendering speed, we selected our *convolution soft shadows map* (CSSM) [Annen08a] for efficient implementation. In Chapter. 5, both the theory and the implementations of CSSM are introduced in details.

6.3.2 CSSM with parabolic projection

In Chapter. 5, we demonstrate that environment map lighting can be efficiently rendered by approximating the map with a number of area lights and then using CSSMs to render each of the area lights. We generalize this approach to *dynamic local* area lights for indirect illumination. Given the M clusters of N VPLs, we place an area light at each cluster center.

Since a diffuse surface reflects towards the whole upper hemisphere, both the perspective and the orthographic projection are not sufficient to compute visibility of an area light representing a cluster of VPLs. Instead, we use a shadow map with a *parabolic* projection, where the sender is oriented around the surface normal [Brabec02]. Parabolic convolution soft shadow maps can be realized as follows. First, an initial filter size is estimated from the solid angle of the current sender VAL. While a VPL does not define an area, a VAL allows for such a computation. Then, the average z value z_{avg} is determined in the same way as for a perspective CSSM. The penumbra size p is then estimated from z_{avg} as shown in Fig. 6.2:

$$p \cdot \cos(\quad) = (d - z_{\text{avg}}) \frac{\Delta}{z_{\text{avg}}},$$

where d is the distance between sender midpoint and receiver point $\mathbf{x}$, Δ is the size of the sender and is the slope of the receiver surface, viewed from the sender midpoint. Given the penumbra size, the size of the filter kernel in the shadow map is adjusted to the angle of the penumbra, viewed from the center of the area light:

$$= \arctan\left(\frac{p \cdot \cos(\quad)}{d}\right).$$

Since texture coordinates range from 0 to 1, the size of the filter kernel w can be estimated as / , the fraction between and the semi-circle . Fig. 6.3 shows examples of different soft shadows computed with this approach.

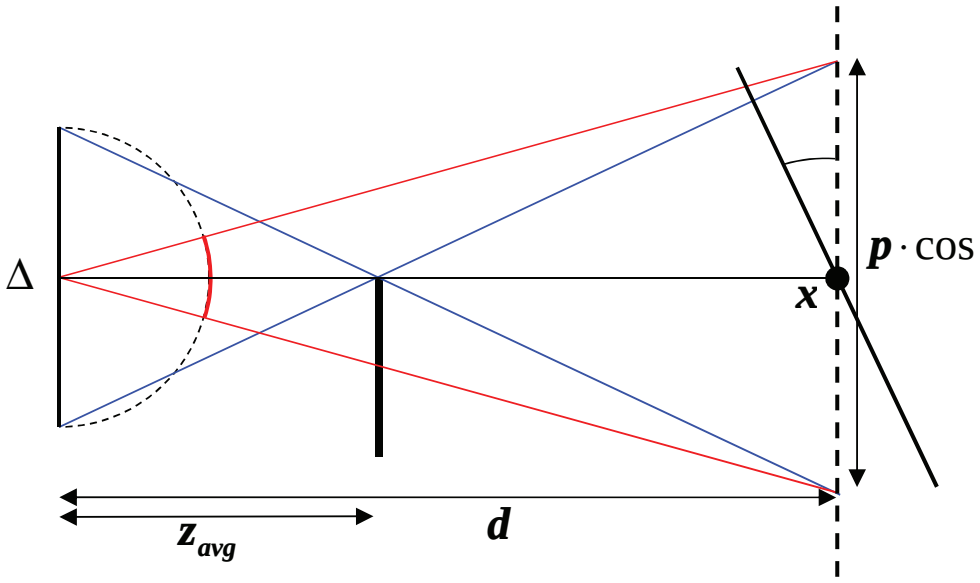


Figure 6.2: Determining the filter size for a paraboloid map.

Discussion

Since parabolic maps use a non-linear projection, it is not correct to approximate the projection of the sender with a squared filter region, which is effectively what CSSMs do. We found the resulting visible error to be small, even for difficult cases as shown in Fig. 6.4. For indirect illumination these errors are acceptable, since many indirect shadows overlap, hiding these artifacts in most cases.

6.4 Clustering

To accelerate the visibility computation for indirect illumination, we group VPLs with similar normals and similar positions into clusters (VALs), i.e., we compute the mapping $\mathscr{C}$. We use a variant of the *k*-means clustering [Carr03] because it is fast and yields good results. After clustering, the position and normal of each VAL are computed by averaging the positions and normals of the contained VPLs. For rendering soft shadows, we additionally compute the area for each VAL (details are described in Section. 8.4).

6.4.1 Clustering criterion

Clustering a set of points with *k*-means consists of two steps. In a first step, starting from arbitrary cluster centers, each point is assigned to the cluster with the

Figure 6.3: Soft shadows generated with the parabolic CSSM method, rendered with more than 200 fps. The left image shows a small area light, a larger emitter is used in the right image.

minimum distance to its center . In a second step, each cluster center is recomputed as the average of all point positions assigned to this cluster. These two steps are repeated until convergence.

In our case VPLs must be assigned to VAL clusters. For grouping VPLs into appropriate clusters, *position* and *normal* of the VPL are taken into account. The distance d between a VPL and a cluster center is therefore computed as:

$$d = w_x \Delta x + w \quad \Delta \quad .$$

where Δx is the euclidean distance between a VPL and the cluster center and Δ is the angle between the VPL normal and the cluster normal. Each term gets a user-defined weight w_x and w . In this way, we create clusters which group nearby VPLs with similar normals. Fig. 6.5 shows how the different weights affect the clustering. In our examples we use the weights $w_x = 0.7$, $w = 0.3$.

Including the normals in clustering is important because artifacts in the VAL plane can appear for clusters with different normals. Since the illumination is computed from all VPLs inside the cluster, the illumination may be non-zero at 90 degrees from the cluster normal (see Fig. 6.6). Because there is no visibility information in the negative halfspace of the VAL, full visibility must be assumed here. If there is a blocker crossing the VAL halfspace, a discontinuity appears, because the blocker is ignored in the negative halfspace of the VAL.

Moreover, the cluster center can be located *inside* the geometry (see Fig. 6.6). To avoid completely occluded VALs, geometry located near the cluster center has

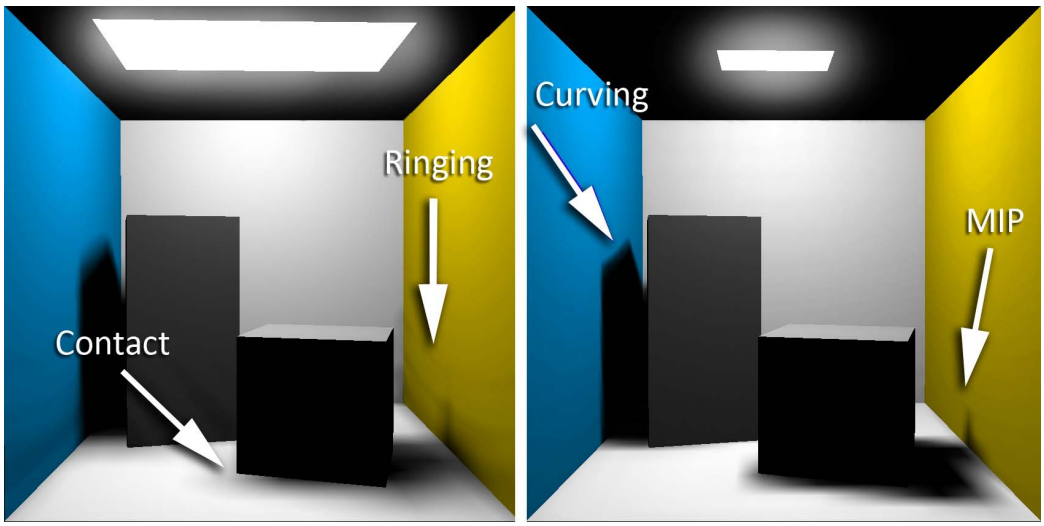


Figure 6.4: Parabolic CSSM limitations: For very large senders, ringing artifacts can appear (left). Penumbra regions are curved when viewed from a grazing angle of a large sender (right). We also inherit problems at contact shadows (left) and MIP discretization (right) from CSMs. Since the indirect illumination consists of many soft shadows, these artifacts are hidden.

to be ignored, which can result in the loss of some existing shadows. Due to all these problems, groups of VPLs with *similar normals* should be preferred which is achieved by giving them a high weight in the clustering.

6.4.2 Temporal coherence

To avoid flickering, the clustering between two successive frames should be similar. To achieve this, a simple strategy would be to use the clustering from the *previous* frame as a starting value for the *k*-means clustering of the *current* frame. In most cases, there are only small changes in light and geometry, so most VPL positions are similar and this quickly converges to a temporally coherent solution.

However, we observed that clusters can be lost, because their center position is in a bad location and all VPLs are assigned to a different cluster center. Fig. 6.7 (left) shows such a case, here a spot light moves from the wall to the ceiling: Because normals are taken into account, all VPLs on the ceiling tend to be grouped into only a few clusters on the ceiling. Several other cluster centers are still located on the wall. Due to the different normals, the distance of any VPL to these centers is bigger than the distance to one of the few clusters on the ceiling. This means that several clusters remain *empty*. When moving the light source, more and more cluster centers stay at an old position, without any VPL assigned to it, and the total number of used clusters decreases over time. If the light moves back to and

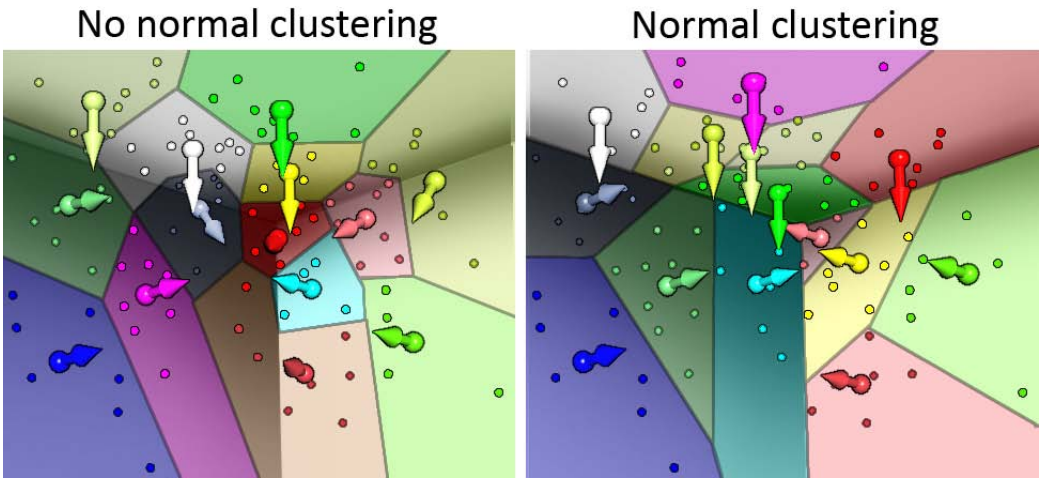


Figure 6.5: Using *k*-means clustering simply based on the euclidean distance between the points results in clusters with varying VPL normals, often located at edges (left). When including the angle between the normals in the distance function, planar groups of VPLs can be formed (right).

old position, an empty cluster might be reactivated, otherwise it will never be used again.

To overcome this problem, we do not reuse the clusters from the last frame, but restart *k*-means from an *identical, initial cluster assignment* at each frame. Since our VPLs are generated from a sequence of Quasi-Monte-Carlo random numbers (see Section. 8.4), all VPLs are placed to similar positions in each frame in case of small movements of light source or geometry. This means that if we use initial clusters based on the the same VPLs every frame, the *k*-means algorithm will converge to a similar result, as shown in Fig. 6.7 (right). Although this increases the total number of *k*-means iterations, the total rendering time is nearly unaffected (see Section. 6.6). The accompanying video shows that our clustering strategy leads to virtual area lights that smoothly float over the surface. The clustering always stays temporally coherent, even in case of animated scenes.

6.5 GPU-Based Rendering from Clustered Visibility

We use a deferred shading renderer, in order to ensure that the expensive indirect illumination is only computed once for every pixel. Geometry is rendered into screen-sized textures for storing position, normal, material and direct illumination, which are then used during the computation of indirect illumination.

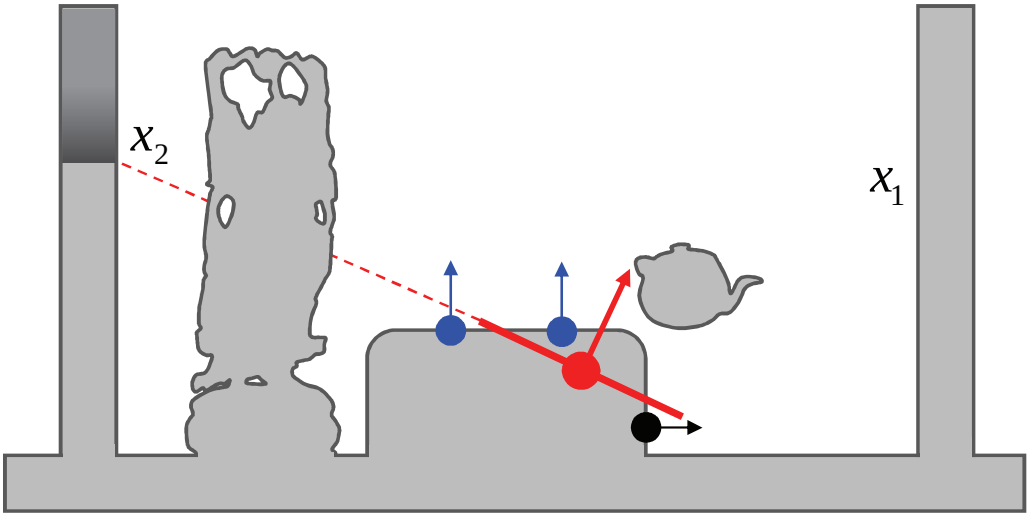


Figure 6.6: Using VALs with varying VPL normals introduces two problems (in this example, three VPLs are grouped into one VAL cluster): First, the cluster center is located inside the geometry, so nearby geometry must be ignored to avoid incorrect self-shadowing. When introducing such a bias, real occluders like the teapot may be clipped away and the shadow at point x_1 disappears. Secondly, discontinuities in the shadow can appear because the illumination is computed from all VPLs: In the example, point x_2 is in the positive half-space of the blue VPLs and the VAL shadow map correctly detects a shadow above x_2. But there is no occlusion information in the negative half-space of the VAL, so everything below x_2 is assumed to be visible. Consequently, the region below x_2 is incorrectly illuminated by the two blue VPLs.

VPL Generation We render a reflective shadow map (RSM) from the light's point of view [Dachsbacher05] (a cube map is used for point lights and a single texture for projective lights) and sample it using a low-discrepancy sampling pattern (Halton sequence) to convert it into N VPLs (Section. 6.3). To this end the RSM is fetched at N Halton-distributed locations using point sampling and the resulting position, normal and color is stored into three N-texel output textures.

Clustering Cluster information is stored in four M-texel textures for position, normal, irradiance and a count of how many VPLs map to a cluster. For each frame, information from the VPL at index $k \cdot N/M$ is used as the initial guess for VAL k (i.e., as cluster k's center). As mentioned earlier, this ensures temporal coherence.

In every k-means iteration, we use scattering [Scheuermann07] and blending

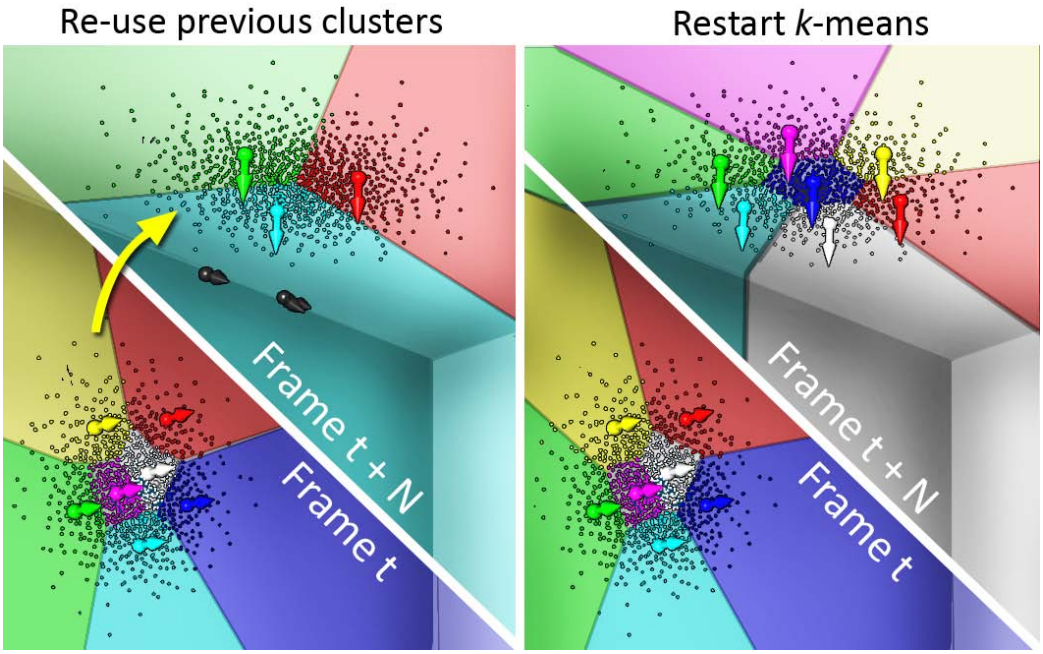


Figure 6.7: A spotlight is moving (arrow) from the wall (frame t, lower half) to the ceiling (frame $t+N$, upper half). Using k-means clustering with the information from frame t, the number of clusters decreases when moving the spot towards the ceiling, as shown on the left. Since normals are taken into account, the distance of any VPL to such a center is too big, so all VPLs are grouped into a few large clusters on the ceiling. To overcome this problem, k-means is restarted using the same initial VPL to cluster assignments each frame. As shown on the right, the number of clusters stays constant.

to update clusters. To this end, for each of the N VPLs a point is drawn using the VPL textures (position, normal, radiance) as input and the four VAL information textures (position, normal, irradiance, count) as output. In a vertex shader, every such point traverses all M clusters, computes the distance, finds the one with the minimum distance and scatters its information to the pixel position of that cluster. We use additive blending and write 1s to the count texture. After every iteration, we draw another full-screen quad, that divides position, normal and radiance by the count resulting in the proper average cluster information.

Note, that in this process, we do not store which VPL maps to which VAL. We create this mapping $\mathscr{C}$ in a final pass and store it as a N-texel texture of pointers into the VAL texture. To this end, we loop over all VPLs, compare them to every VAL and output the pointer to the VAL with minimal distance.

For soft shadow computation we need to know the area of each VAL. We define it as the 2D bounding rectangle of the two-dimensional projection s,t of

the VPL position onto the plane perpendicular to the average normal of the cluster it maps to.

In summary, we compute an M-texel texture that stores cluster position, normal and area complemented by an N-texel texture that stores the mapping from each VPL to a VAL, i.e., representing $\mathscr{C}$.

Paraboloid CSSM Instancing is used to draw the scene into a texture array of depth maps with a single pass (the resolution of one paraboloid map is set to 256×256). From this depth map texture array we generate an array of Fourier basis textures (4 term, 8 bit) and Fourier basis-z textures (4 term, 16 bit half float) (cf. Section. 6.3.1). Finally, a MIP map is built for both the basis and the basis-z texture array.

Indirect Lighting Indirect lighting is computed using interleaved sampling [Segovia06]. We use blocks of $8 \times 8 = 64$ pixels with 1024 VPLs that result in $1024/64 = 16$ VPLs per pixel. While we use VALs for visibility, we still use the full number of VPLs for lighting. So when shading from VPL i we use the VAL at index $\mathscr{C}(i)$ for visibility, looking up $\mathscr{C}$ in the mapping texture.

We use a geometry aware blur to remove the remaining Monte Carlo noise without blurring over edges. As noted by Laine et al. [Laine07], using $= 10\%$ of the scene's extend and $= 0.8$ seems to work reasonably well for our results.

6.6 Results and discussion

In the following, we present results rendered at real-time rates with our technique on a 3 GHz CPU with an NVIDIA GeForce 8800 GTX. All scene components can be fully dynamic (geometry, materials, and lights), as no precomputation is required.

Fig. 6.8 shows a global illumination solution computed with clustered visibility. To illustrate our approach, we included some individual soft shadow images, generated from selected VAL. To verify the correctness of our approach, we successively increase the number of VALs and compare our result with the ground truth solution from instant radiosity and path tracing. As shown in Fig. 6.9, the resulting images are similar, even if visibility is computed from a very small number of VALs.

The performance for our test scenes is summarized in Tbl. 7.1. The rendering time of each individual part of our algorithm is described in Tbl. 6.2. As shown in Fig. 6.10, we can display global illumination in an animated game scenario at interactive fame-rates. Our approach allows for extremely dynamic geometry, such as the iso-surface in Fig. 6.11. Note, that all pre-computed visibility

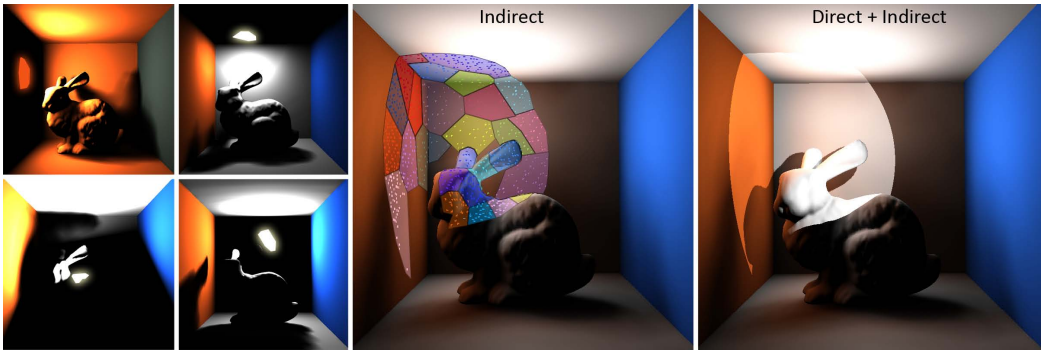


Figure 6.8: Soft shadows generated for indirect illumination. In this scene a spotlight illuminates the corner of the box, so most of the light is indirect. The images on the left show individual soft shadows from some selected VALs. The complete clustering ($M = 30$ VALs) is shown in the center image. The full global illumination solution is shown on the right.

Scene	Faces	VPLs	VALs	fps
Cornell Box	18	1024	30	20.4
Cornell Horse	17 k	1024	30	19.7
Sponza	98 k	1024	30	13.4
Metaballs	5 k	1024	30	20.7

Table 6.1: Frame-rates (800×800 pixels).

methods, and even imperfect shadow maps[Ritschel08b], which are restricted to area-preserving deformations, would fail for this scene.

We support soft and crisp indirect visibility at the same time, as shown in Fig. 6.12.

6.6.1 Discussion

While the use of VALs provides an efficient means to compute indirect illumination, there are some limitations. We currently use reflective shadow maps to generate VPLs [Dachsbacher05], restricting us to point and spot lights. The efficiency of the VALs hinges on using rather low-resolution CSSMs, which in turn means that we cannot resolve very thin indirect shadows. Furthermore, we inherit other CSSM limitations, such as difficulties to resolve contact shadows (see [Annen08a]). Extending image space shadow bias removal [Ritschel09b] to soft shadows is future work. If an insufficient number of VALs is used, individual shadows from each VAL might be visible, as can be seen in Fig. 6.9. Using a sufficient number of VALs prevents any artifacts. Our method also depends on the geometric complexity of the scene, since the scene needs to be rendered once

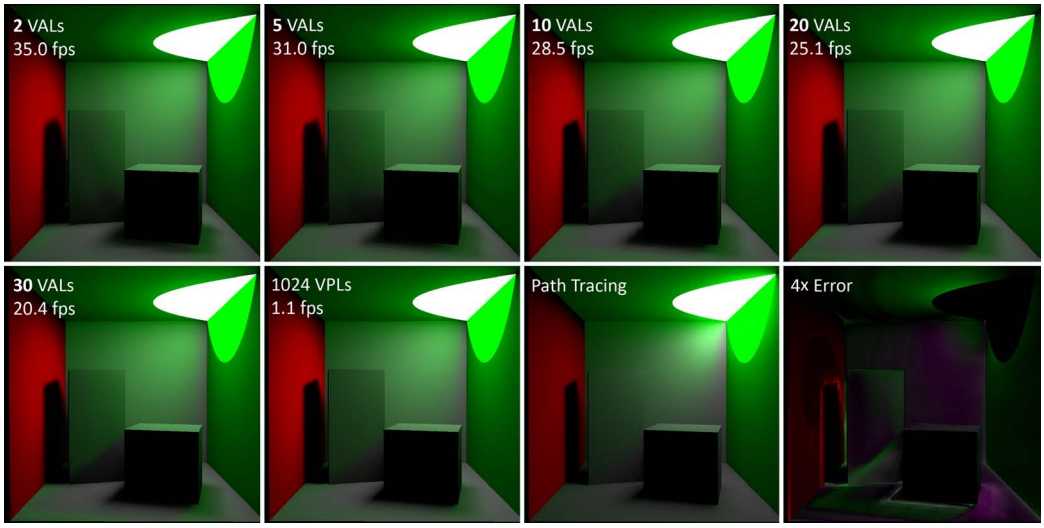


Figure 6.9: When increasing the number of VALs, the indirect illumination converges to the correct result. The images show (in reading order) 2 to 30 VALs that are used for the indirect visibility. The next two images show an IR solution with a hard shadow for each VPL and a path tracing solution. The difference between our solution (M=30) and the standard IR solution is shown on the bottom right. Note that already a very small number of VALs creates a convincing indirect illumination. An 8×8 G-Buffer was used to reduce the number of VPLs per pixel.

for each VAL. However, it might be possible to reduce this dependency with imperfect shadow maps[Ritschel08b].

In contrast to normal instant radiosity, we are less prone to temporal aliasing, since we can start the clustering process with a sufficient number of VPLs yielding good VAL approximations. Furthermore, there is only one major parameter: the number of VALs, which makes our technique more applicable.

6.7 Summary

We demonstrated that indirect visibility can be approximated with a small number of area lights in combination with a soft shadowing method. Due to the fast computation time of the soft shadow algorithm we can display approximated indirect illumination at interactive to real time speed without large differences in image quality.

As future work, we will investigate if geometric simplifications can be included on top of the visibility approximations, e.g. if a combination of imperfect shadow maps [Ritschel08b] and coherent soft shadows is possible. Furthermore,

Step	Time (ms)	Percentage
Deferred rendering	0.4	0.8%
VPL generation	0.1	0.2%
VAL clustering	0.5	1.0%
CSSM	4.2	8.1%
Indirect illumination	35.0	69.0%
Geometry-aware blur	10.7	21.0%

Table 6.2: Performance breakdown for *Cornell Horse*.

Figure 6.10: A complex dynamic scene (100 k faces) with multiple animated dragons in *Sponza* (14 fps). Note, how the light bouncing from the back wall dominates (arrow). Please also see the supplemental video.

we would like to adapt the number of VAL clusters to the illumination complexity, in order to keep the number of clusters at the minimum required number for good visual quality. The extension from one bounce to multiple bounces of light would be an interesting avenue of further research as well as the inclusion of highly glossy materials. Finally, the combination between natural illumination from an environment map and indirect bounces of light should be investigated.

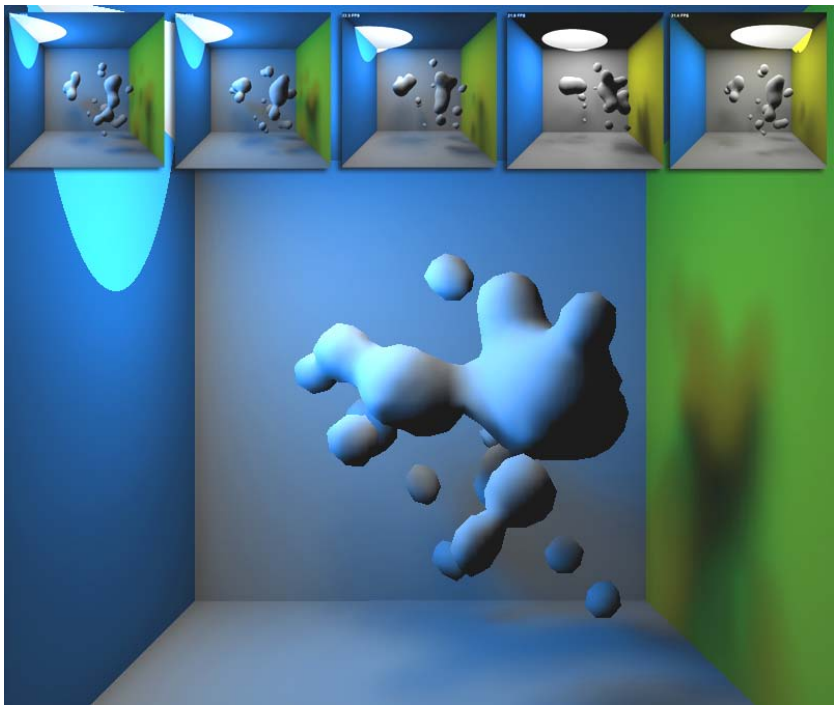


Figure 6.11: Our method rendering global illumination (20.7 fps) for a scene with dynamic topology (5.1 k faces).

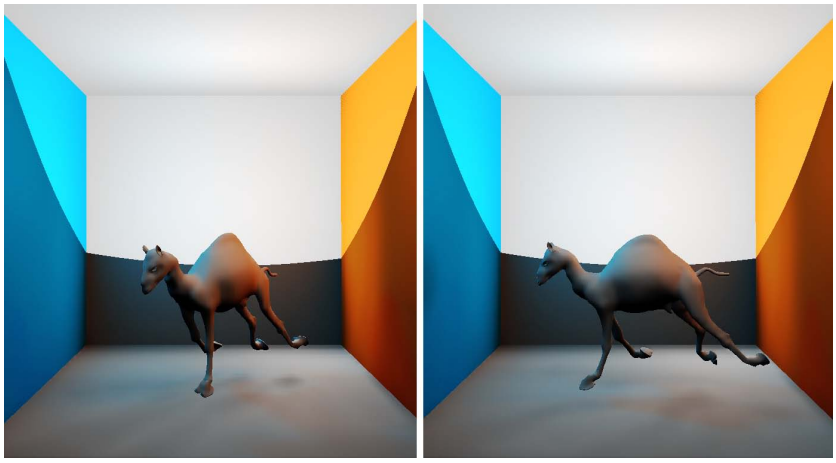


Figure 6.12: A wide spot casting a soft shadow.

Chapter 7

Variance Soft Shadow Maps

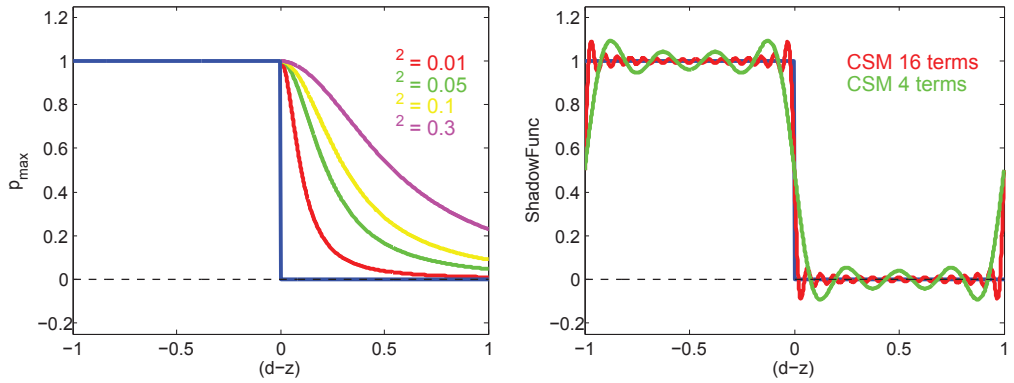
7.1 Introduction

Shadow rendering is a basic and important feature for many applications. However, applications like video games require shadow rendering to be very efficient — ideally less than 10ms per frame. *Shadow mapping* [Williams78] is a purely image-based shadow method which scales well with scene complexity. Hence it fulfills the strict requirement of game engines and has become the de facto standard for shadow rendering in computer games. While the original shadow mapping method only deals with hard shadows, a variant called *percentage closer soft shadow* (PCSS) [Fernando05a] is sometimes used for rendering soft shadow. PCSS achieves visually plausible quality and real-time performance for small light source. Moreover, its implementation only incurs shader modification and is easy to be integrated into existing rendering system. As introduced in the Section. 2.4.2 of Chapter. 2, the PCSS method mainly contains two steps: (1) computing the average blocker depth, and (2) evaluating the final soft shadows. The algorithmic pipeline of the PCSS method can be regarded as a general soft shadow mapping framework based on the planarity assumption.

7.1.1 Soft Shadowing with PCSS

Following the PCSS pipeline, several pre-filtering soft shadow mapping methods [Lauritzen07, Annen08a] have been recently introduced. *Convolution soft shadow map* (CSSM) in Chapter. 5 is built on pre-filterable shadow reconstruction functions using the Fourier basis. The reconstruction functions with different number of basis terms are shown in Figure 7.1(b). It is easy to see that the reconstruction curve of CSSM covers the whole range of $(d-z)$ values. Such a *double-bounded* pre-filtering function can be applied for both average

blocker depth computation and soft shadow test, and fits very well into the PCSS framework. Yet, large amounts of texture memory are required to store Fourier basis terms, making it less practical. Compared to CSSM, *Variance Shadow Maps* (VSM) [Donnelly06a] support pre-filtering based on a one-tailed version of Chebyshev's inequality and requires a much lower amount of texture memory. Unfortunately, there is no obvious way to correctly pre-filter average blocker depth values based on the VSM theory. In [Lauritzen07], the average blocker depth evaluation step is therefore performed by brute-force point sampling of the depth map. The shadow reconstruction curve of VSM are shown in Figure 7.1(a).



(a) VSM Func (b) CSM Func

Figure 7.1: Comparison between different pre-filtering shadow functions. The blue line represents the heaviside step function for the shadow test. d represents the depth value of current point and z represents the depth value sampled from shadow map with a filter kernel.

It is easy to see that this curve only bounds one side of shadow comparison function and is undefined when $(d - z) \leq 0$. Therefore, we call it *single-bounded* pre-filtering shadow function. Existing techniques [Donnelly06a] simply assume the shadow value is equal to 1 in this case. When the average depth value z_{Avg} of a filter kernel is bigger than or equal to the depth value d of the current point, this point will be assumed to be fully lit. When handling hard shadow or when the filter kernel is very small, this assumption is reasonable. However, when handling large kernel for soft shadow, this lit-assumption for the whole kernel can introduce incorrect result (lit pixels instead of partially shadowed). We refer to this as the "non-planarity" problem for *single-bounded* pre-filtering shadow functions [Salvi08]. Such incorrectly-lit artifacts are more serious when the kernel size increases.

7.1.2 Our Method

Motivated by aforementioned problems, *Variance Soft Shadow Map* (VSSM) is introduced to enable real-time, high-quality soft shadow rendering with low-memory cost. Our key contributions in this chapter are:

1. Derivation a novel formula for estimating average blocker depth, which is based on VSM theory.

2. An efficient and practical filter kernel subdivision scheme that handles the "non-planarity" lit problem for *single-bounded* VSM shadow functions. The subdivision scheme can be either in uniform way or in adaptive way which is based on linear quad-tree traversal on the GPU. Such a divide-and-rule strategy succeeds in efficiently removing incorrect-lit.

7.2 Overview

An overview of the VSSM algorithmic steps are given in Algorithm 7.2 and we refer to the line numbers as (L*xx*) in the text. First, we generate a normal shadow map and two textures based on it (L2-L4): a summed-area table (SAT) and a min-max hierarchical shadow map (HSM). Then for each visible scene point P, we do the following: Firstly, the initial filter kernel w_i (blocker search area) is computed (L7) by intersecting the shadow map plane with the frustum formed by P and the light source. We then sample the average depth value z_{Avg} in w_i from the SAT texture and the min-max depth range in w_i from min-max HSM. Comparing the depth value d of P with the min-max depth range, we can quickly find the fully-lit and fully-blocked (lit/umbra) scene points and ignore them for following soft shadow computation (L10). Then for the scene points that are left and which are potentially penumbra, our VSSM method checks whether w_i is a "non-planarity" kernel or not. The condition here is whether $z_{Avg} \geq d$. If w_i is not a "non-planarity" kernel, the average blocker depth will be estimated directly using a new formula (L15), which will be introduced in section 7.3. If w_i is a "non-planarity" kernel, w_i needs to be subdivided either uniformly or adaptively to compute the average blocker depth (L12-L13). The kernel subdivision scheme will be explained in detail in section 7.4. After getting the average blocker depth, the actual penumbra kernel w_p can be computed (L16). Note, the computation for kernel size in this step is similar to L7, and just the shadow map plane is substituted by the average blocker depth plane. Finally, the variance-based soft shadow value of penumbra kernel w_p can be evaluated either directly or using the kernel subdivision scheme.

Algorithm 2 Overview of VSSM algorithm

```
1  Render scene from light center:
2    Render normal variance depth map
3    Generate summed-area table (SAT) for the depth map.
4    Render the min-max hierarchical shadow map (HSM)
5      for the depth map
6  Render scene from view point. For each visible point P:
7    Compute the initial kernel w_i (blocker search area)
8    Check if P is lit or umbra using the HSM
9    if (P is lit or umbra)
10     return the shadow value accordingly
11   if (w_i is "non-planarity" kernel)
12     Subdivide filter kernel
13     Estimate average blocker depth using novel formula
14   else
15     Estimate average blocker depth using novel formula
16   Compute penumbra kernel w_p based on average blocker depth
17   if (w_p is "non-planarity" kernel)
18     Subdivide filter kernel and evaluate soft shadow value
19   else
20     Evaluate soft shadow value directly
21 Render the final image using the visibility factors
```

7.3 Variance Soft Shadow Mapping

In this section, we introduce the theory about how to efficiently estimate average blocker depth for VSSM.

7.3.1 Review of Variance Shadow Maps

Variance shadow maps are based on the one-tailored version of Chebyshev's inequality. Let x be a random variable drawn from a distribution with mean and variance ^{2}, then for $t >$:

$$P(x \geq t) \leq p_{\max}(t) \equiv \frac{^2}{^2 + (t -)^2} \tag{7.1}$$

Considering t represents the current point's depth d, and x represents the sampled depth z from the shadow map, the quantity $P(x \geq t)$ in Eq. 7.1 represents the fraction of texels over a filter kernel that will fail the depth comparison, which is exactly the same as the result of PCF sampling. Since $= E(x) = x$ and $^2 = E(x^2) - E(x)^2$, $E(x)$ and $E(x^2)$ can be generated on-the-fly to pre-filter the shadow test.

Note, only in the particular case of a single planar occluder at depth d_1, casting a shadow onto a planar surface at depth d_2, the upper bound of Eq. 7.1 will be equal to the shadow test result. In most other cases, Eq. 7.1 will not provide an exact value, but a close approximation (Fig. 7.1).

7.3.2 Estimating Average Blocker Depth

In order to fit VSM into the PCSS framework, the difficult problem is how to efficiently estimate the average blocker (first step in PCSS, see Sec. 7.1).

Considering a filter kernel w and the current point's depth t, the pre-filtered depth value z and its square z^2 can be sampled from the VSM. Based on linear filtering, the sampled z is actually the average depth value z_{Avg} in w. The depth values for all the texels in w can be separated into two categories: (1) the depth values which are $\geq t$ and the average of this kind of depth values is defined as z_{unocc}, (2) the depth values which are $< t$ and the corresponding average value is defined as z_{occ}. Let's assume there are N samples in total in filter kernel w. N_1 of them are $\geq t$ and N_2 of them are $< t$. The following equation holds:

$$\frac{N_1}{N} z_{unocc} + \frac{N_2}{N} z_{occ} = z_{Avg} \tag{7.2}$$

It is easy to see $\frac{N_1}{N}$ and $\frac{N_2}{N}$ correspond to shadow test results $P(x \geq t)$ and $P(x < t) = 1.0 - P(x \geq t)$. Therefore, Eq. 7.2 can be written as:

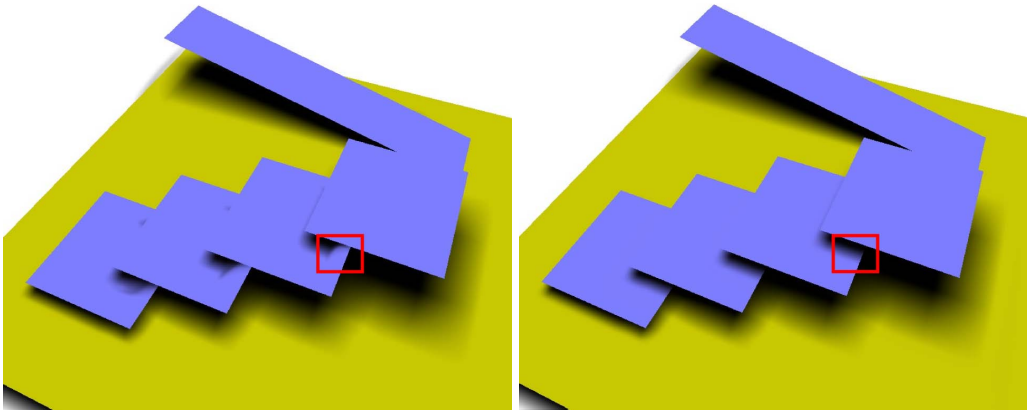
$$P(x \geq t) z_{unocc} + (1.0 - P(x \geq t)) z_{occ} = z_{Avg} \tag{7.3}$$

Therefore, the average blocker depth value z_{occ} is:

$$z_{occ} = (z_{Avg} - P(x \geq t) z_{unocc}) \big/ (1.0 - P(x \geq t)) \tag{7.4}$$

z_{Avg} is known and $P(x \geq t)$ can be evaluated based on Chebyshev's inequality. The only unknown variable left is the average non-blocker depth value z_{unocc}. Observing that in the aforementioned two-plane scene setting, $P(x \geq t)$ is accurate and in this case, $z_{unocc} = t$. We therefore assume $z_{unocc} = t$ and use it for general cases as well. This assumption generates high-quality soft shadows in all our experiments.

While it may now seem straightforward to compute the average blocker depth value, the new formula relies on the VSM shadow value $P(x \geq t)$. As mentioned already, the shadow reconstruction function of VSM is just "single-bounded". If $z_{Avg} \geq t$, this will break the prerequisite of Chebyshev's inequality and the "non-planarity" lit problem can occur (Fig. 7.2(a)). In order to evaluate the average blocker depth correctly, we need to correct $P(x \geq t)$, which we propose to do using subdivision. In the following section, we will explain the kernel subdivision scheme which deals with the "non-planarity" problem (Fig. 7.2(b)).



(a) Without kernel subdiv. (b) With Kernel subdivision

Figure 7.2: Comparison between without kernel subdivision and with kernel subdivision.

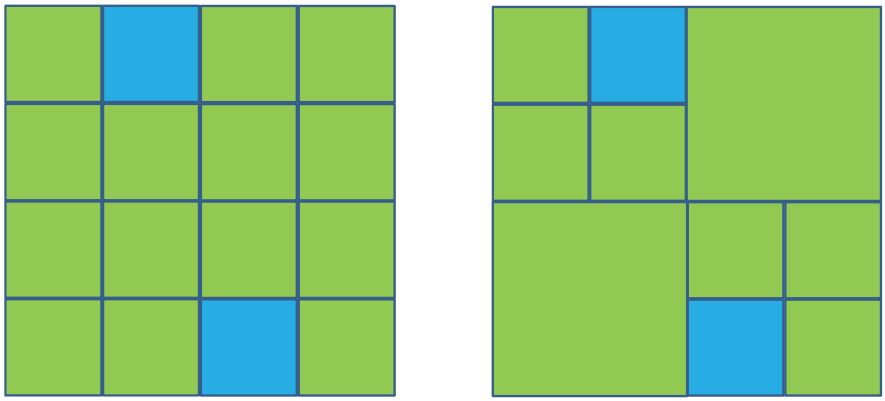
7.4 Non-Planarity Problem and its Solution

In this section, we introduce the uniform and the adaptive filter kernel subdivision schemes to handle the "non-planarity" problem.

7.4.1 Motivation for Kernel Subdivision

For an arbitrary filter kernel w of a scene point P, the "non-planarity" problem occurs if $z_{Avg} \geq t$. Here, t represents the current point's depth d. Standard VSM will assume that the shadow value equals to 1 in this case. When w is small, it is reasonable since the depth values of most texels in w are likely to be bigger than t. However, when the size of w increases, only part of the texels in w contain bigger depth value than d. Therefore, the errors due to the lit assumption for the whole w becomes obvious.

Following the concept of divide-and-rule, we propose to subdivide the kernel w into a set of sub-kernels $\{w_{ci}, i \in [1 \ldots n]\}$. Depending on whether $z_{Avg} \geq d$ in w_{ci}, all the sub-kernels can be categorized into two parts. For the normal sub-kernels fulfilling $z_{Avg} < t$, the Chebyshev's inequality still holds and we can compute the average blocker depth and soft shadow based on it. For the "non-planarity" sub-kernels fulfilling $z_{Avg} \geq t$, there are two options: (1) assuming each of them to be lit or (2) using normal PCF sampling to do the shadow test. Option (1) is similar to the previous VSM strategy. However, since the sub-kernel w_{ci} is much smaller than initial kernel w, the "non-planarity" lit problem can be effectively suppressed. Option (2) is a good alternative, since few cheap PCF samplings (2×2) can generate rather accurate results for w_{ci}. In our implementation, option (2) is chosen.



(a) Uniform kernel subdivision (b) Adaptive kernel subdivision

Figure 7.3: Illustration of 4×4 uniform and adaptive subdivision for filter kernel.

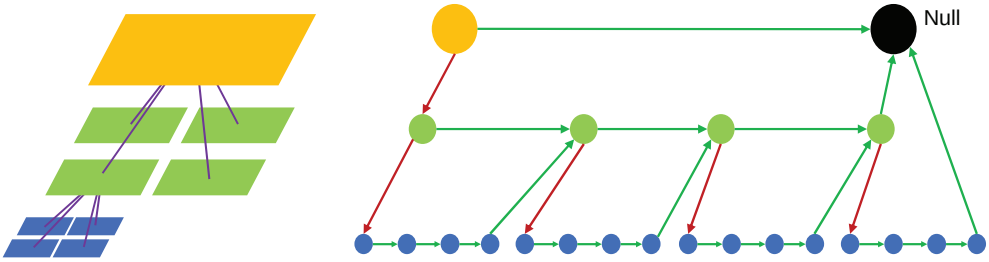
7.4.2 Uniform Kernel Subdivision Scheme

Since the corner points of the initial kernel w are known, it is straightforward to subdivide it into equal-sized sub-kernels. As illustrated in Fig. 7.3(a), the whole quad represents the initial kernel w and each sub-quad inside of it represents a sub-kernel w_{ci}. We loop over each w_{ci} and check whether it is a "non-planarity" kernel or not. In Fig. 7.3, the blue sub-quad represents "non-planarity" kernels and the green one represents the normal kernels. Here we conceptually separate all the sub-kernels into two groups: the normal sub-kernel group $w_{cj}$and the "non-planarity" sub-kernel group w_{ck}. In following, we will illustrate how to estimate average blocker depth using the uniform kernel subdivision scheme.

Let's first consider the normal sub-kernel group: To compute $P(x \geq t)$ using Eq. 7.1 for the whole group, the mean value and the variance ^{2} needs to be determined. More specifically, the $E(x)$ and $E(x^2)$ from all the sub-kernels in this group need to be computed. We define the size of w_{cj} to be T_{cj}, and arrive at the following formulas to compute and ^{2} for the normal sub-kernel group:

$$\begin{aligned} &= \sum_j (E(x)_{cj} \cdot T_{cj}) / \sum_j T_{cj} \\ ^2 &= \sum_j (E(x^2)_{cj} \cdot T_{cj}) / \sum_j T_{cj} - \quad ^2 \end{aligned} \tag{7.5}$$

During the loop, $E(x)_{cj}$ and $E(x^2)_{cj}$ can be sampled from the SAT texture for each w_{cj}. Then, $E(x)_{cj} \cdot T_{cj}$, the sum of all texels' depth in w_{cj}, is accumulated. The same accumulation happens for $E(x^2)_{cj} \cdot T_{cj}$. Finally, the depth and depth square sums are divided by the accumulated sub-kernel size (Eq. 7.5) to get and ^{2}. Hence, the VSM shadow reconstruction function can be evaluated. The average blocker depth d_1 in the normal kernel group can be evaluated using Eq. 7.4.



(a) Kernel Hierarchy (b) Linear quad-tree traversal

Figure 7.4: Linear quad-tree traversal on 2D filter domain.

For the "non-planarity" sub-kernel group, we apply standard PCF sampling for each w_{ck}: m points are sampled in w_{ck} and the sum of all the blocker depth samples are computed. Since the size of w_{ck} is small, usually $m = 2 \times 2$ is enough. In following steps, both the sum of all the blocker depth and the sum of all the blocker sub-kernel size are accumulated. Similar as before, we can get the average blocker depth d_2 of the "non-planarity" sub-kernel group.

After getting d_1 and d_2, the average blocker depth d over the whole kernel w can be computed by combining d_1 and d_2 weighted by the corresponding blocker kernel size separately. Note there is a reasonable acceleration strategy: the variance ^{2} represents the depth value variation in each w_{ci}. Hence, when the z_{Avg} in w_{ci} is bigger than current depth d, and if the ^{2} is also less than a small *Threshold*, such a w_{ci} can probably be treated as fully-lit.

7.4.3 Adaptive Kernel Subdivision Scheme

To achieve better subdivision granularity control, we propose an adaptive kernel subdivision scheme, as shown in Fig. 7.3(b). Compared to the uniform scheme, adaptive kernel subdivision processes sub-kernels in hierarchical way. Its performance is a balance between the hierarchical culling gain and traversal cost. Usually when the number of sub-kernels is large (≥ 64), adaptive subdivision achieves better performance.

Since our filter kernel is always a 2D square, we can construct a quad-tree in the 2D domain (Fig. 7.4(a)). For the filter kernel w, the root node of the quad-tree represents w itself and each tree node represents a sub-kernel $\{w_{ci}, i \in [1 \ldots n]\}$. Be aware the different w_{ci} are not equal-sized anymore and they could exist in different levels of tree hierarchy. In Algorithm 7.4.3, we show the steps for computing the final soft shadow value based on adaptive kernel subdivision scheme.

Standard quad-tree traversal depends on recursive operations, which is not easily implemented on a stackless GPU. We borrow the idea from [Bunnell05]

Algorithm 3 Adaptive kernel subdivision algorithm

```
1  Define VSMEx = 0, VSMEx2 = 0, VSMArea = 0
2         PCFArea = 0,PCFBArea = 0,LitArea = 0
3  Start from the root node of kernel w, TreeNode = root
5  While( TreeNode != Null ):
6    TreeNode w_ci is current node
7    Compute texcoords UV and kernel size T_ci
8    Sample the E(x)_ci and E(x^2)_ci from SAT
9    Compute the variance  ^2
10   If (E(x)_ci ≥ d And  ^2 < Threshold)
11     TreeNode = TreeNode.Next
12     LitArea = LitArea + T_ci
13   Else
14     If (E(x)_ci < d)
15       VSMEx = VSMEx + E(x)_ci × T_ci
16       VSMEx2 = VSMEx2 + E(x^2)_ci × T_ci
17       VSMArea = VSMArea + T_ci
18       TreeNode = TreeNode.Next
19     Else
20       If (TreeNode is not a leaf)
21         TreeNode = TreeNode.Child
22       Else
23         Sample m points inside the kernel w_ci of TreeNode.
24         PCFArea = PCFArea + T_ci
25         PCFBArea = PCFBArea + T_ci × m̄/m
26         //m̄ is the number of occluding samples
27         TreeNode = TreeNode.Next
28 End While
29 Compute shadow reconstr. value L from VSMEx and VSMEx2
30 Compute visibility L1 based on PCFArea and PCFBArea
31 Final visibility is computed using L, 1.0 and L1,
32   weighted by VSMArea, LitArea and PCFArea separately
```

and successfully apply the GPU-based linear quad-tree traversal for our 2D filter kernel domain. To achieve the linear traversal, each quad-tree node needs to define two pointers (as shown in Fig. 7.4(b)): 'Child' pointer (red), which points to the first child node, and the 'Next' pointer (green), which points to the next tree node on the linear traversal path. After carefully setting up the 'Next' pointer for each tree node [Bunnell05], we avoid the usual recursive operation and enables a linear forward traversal on the GPU.

Note, computing soft shadow value needs to consider the fully-lit sub-kernels.

In Algorithm 7.4.3, we define a variable *LitArea* to record the size of all the fully-lit sub-kernels. In L10-L12, when both $E(x)_{ci} \geq d$ and $^2 < \mathit{Threshold}$, the current w_{ci} is fully-lit, and its all child nodes can be ignored. So we accumulate the *LitArea* and move to the next treenode. If $E(x)_{ci} < d$, w_{ci} is a normal sub-kernel. As before, we accumulate the sum of $E(x)_{ci}$ (L15), the sum of $E(x^2)_{ci}$ (L16) and the sum of sub-kernel size T_{ci}(L17). After accumulation, we then move on to the next treenode (L18). Excluding from above two cases, the last case is when $E(x)_{ci} \geq d$ and $^2 \geq \mathit{Threshold}$. In this situation, we should consider whether the current treenode is a leaf or not. If the current treenode is a non-leaf node, go down the tree hierarchy to its child node (L21). Otherwise, if the current treenode is a leaf node, we resort to use PCF for the visibility computation (L23-L26) as before. When it is done, we move to the next treenode. After the whole traversal is finished, the final visibility can be evaluated (L29-L32). Note, here all the three sub-kernel regions: *VSMArea*, *LitArea* and *PCFArea* are required to compute the final result.

7.5 Implementations and Discussion

7.5.1 Min-Max Hierarchical Shadow Map

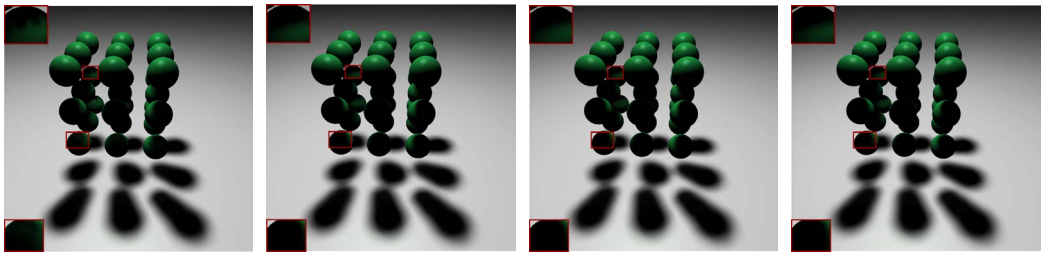
When generating the min-max hierarchical shadow map (HSM), there are two options: Mip-map and N-buffers [Décoret05]. Mip-maps can be generated very efficiently and also require little texture memory. However, the introduced error tends to be obvious when sampling from high mip-map level. In contrast, N-buffers can ensure accurate min-max sampling results for arbitrary filter kernel sizes with more memory and generation time. For the scene setting of Fig. 1.4(b), generating a 1024×1024 HSM, takes 3ms with N-buffers and only 1ms with mip-maps. In our experiments, the HSM using mip-maps already works very well, even for complex scenes. Hence we choose mip-maps for HSM generation.

7.5.2 Number of Sub-Kernels

Applying uniform kernel subdivision to evaluate both average occluder depth and soft shadows, the number of sub-kernels in these two steps can be represented as $m \times m$ and $n \times n$ respectively. If the number is too low, the "non-planarity" sub-kernel will have relatively larger size, so that a low number of PCF samplings (2×2) will not be enough to avoid perceptible artifacts (Fig. 7.5(a)). In Fig. 7.5(b), we increase m to 5 and the artifacts are successfully removed. In fact, we find that $m = 5$ works well in most of our experiments. Furthermore, average blocker

depth evaluation is less sensitive to precision compared with shadow computation. Hence, n is usually larger than m.

Our adaptive subdivision is based on a full quad-tree of 2D sub-kernels. If the height of the quad-tree is H, there are maximally 4^H leaf nodes corresponding to 4^H sub-kernels. If taking $H = 3$, our experimental results show that the quality of adaptive subdivision is basically the same as for uniform subdivision (Fig. 7.5(b) and (c)).



(a) $m = 3, n = 8$ (102 fps) (b) $m = 5, n = 8$ (98 fps) (c) $H = 3$ (94 fps) (d) $m = 5, H = 3$ (117 fps)

Figure 7.5: Comparison between different subdivision cases. m and n represent the number of sub-kernels when using uniform kernel subdivision. H represent the height of quad-tree when using adaptive kernel subdivision.

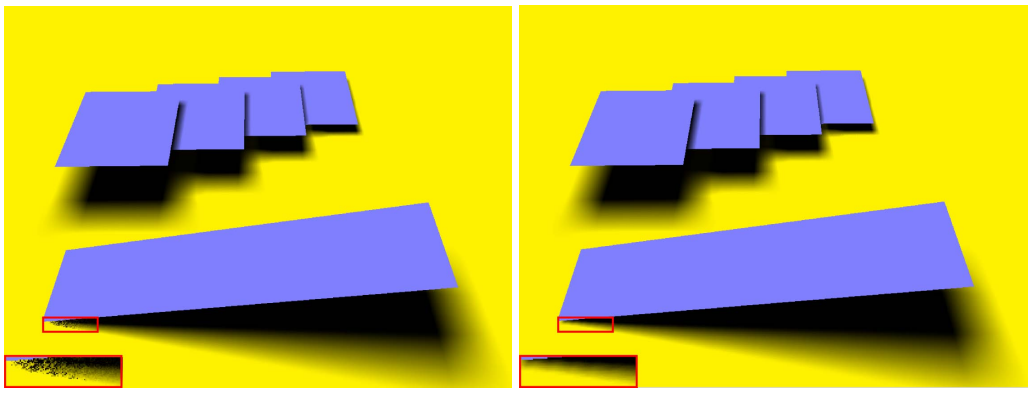
7.5.3 Combining Different Subdivision Schemes

The performance of adaptive subdivision is a balance between the hierarchical culling gain and traversal cost. When the number of sub-kernels is small (like $m = 5$), the traversal cost could hinder the performance. A better strategy is to use uniform subdivision for evaluating average occluder depth ($m = 5$) and adaptive subdivision for the soft shadow ($H = 3$) separately. As shown in Fig. 7.5(d), such a combination gives the same quality but provides the best performance.

7.5.4 SAT Precision and Contact shadow

We adopt summed-area tables (SAT) to pre-filter the shadow map. However, it is well known that SAT suffers from numerical precision loss for large filter kernel. Following [Lauritzen07], the 32-bit integer format is used for SAT generation to achieve stable shadow quality. However, in contact shadow areas, where the blocker and receiver are placed very closely, the precision of integer SAT is still not enough and can introduce small errors (Fig. 7.6(a)) for the average blocker depth z_{Avg}. We observe that in contact shadow areas, the difference between z_{Avg} and d is very small [Annen08a], and hence the corresponding penumbra size is

also very small and applying several PCF samplings for shadow is usually sufficient. In our experiments, the contact shadow sub-kernels are detected by checking the difference between z_{Avg} and d. If the difference is smaller than a threshold value , a 3×3 jittered bilinear PCF sampling [Bavoid08] is applied for evaluating soft shadow. In our experiments, $= 0.01 \cdot r$ and r is the bounding sphere radius of input scene. The experimental results demonstrate such a strategy can avoid precision artifacts and generate convincing contact shadows (Fig. 7.6(b)).



(a) Contact Noise (b) Noise Fixed

Figure 7.6: Fixing contact shadow noise.

7.5.5 Threshold Selection

In Algorithm 7.4.3, there is a *Threshold* value which is used to identify nearly-planar regions that can be safely marked as fully lit. In all of our tests, $Threshold = 0.0001 \cdot r$ works well and r is the bounding sphere radius of input scene.

7.6 Results

Our experiments were run on a PC with a quad-core 2.83GHz Intel Q9550 CPU, an NVIDIA GeForce GTX 285, and 4GB of physical memory. Except for comparison in Fig. 7.5, all the result images are using mixed kernel subdivision scheme: 5×5 uniform subdivision for estimating average blocker depth, and $H = 3$ adaptive subdivision for computing soft shadow. The screen resolution for rendering is always 1024×768.

Table 7.1 provides the performance breakdown for different scenes: Balls (55k faces) in Fig. 7.5, Sponza (72k faces) in Fig. 1.4 (a) and Soldier (9700k faces with 100 instances) in Fig. 1.4 (d). Each row contains timings for: generate G-buffer data (GBuf), render shadow map (SM), generate mip-map HSM (HSM),

Scene	GBuf	SM	HSM	SAT	Shadow	Total
Balls	2.0	0.1	0.1	2.5	3	7.7
Sponza	2.5	0.6	0.3	4.4	2.0	9.8
Soldier	13.8	10.6	0.3	4.3	3.8	32.8

Table 7.1: Performance (milliseconds) breakdown for different scenes (SM:1024×1024**).**

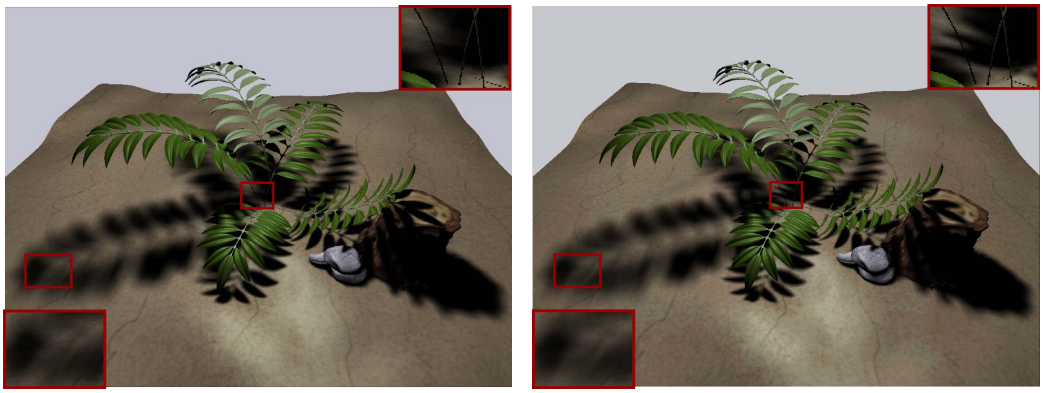
generate summed-area table (SAT), and soft shadow pass (Shadow). The final column (Total) is the sum of each step's timing. From the data, we can see that SAT usually takes a significant ratio of the running time. The running time of soft shadow pass also depends on how many screen pixels locate in penumbra. In the soldier scene, penumbras appear in many areas, so the soft shadow pass takes more time. Also since the geometric burden in the soldier scene is very high, the GBuf and SM become the bottleneck of rendering. Table 7.2 provides the performance breakdown for the Plant (142k faces) in Fig. 7.7 using different SM resolution. With increasing resolution, the SAT becomes the bottleneck and also increases the sampling cost in the soft shadow pass.

SMRes	GBuf	SM	HSM	SAT	Shadow	Total
512	1.7	0.17	0.18	2.1	2.5	6.65
1024	2.0	0.25	0.3	4.8	3.5	10.85
2048	2.0	0.5	1.0	17.9	4.4	25.8

Table 7.2: Performance (milliseconds) breakdown using different SM resolution for plant scene (141k faces).

The result images shown in Fig. 7.7 compare the shadow quality of several different algorithms including a ray-traced reference image. We analyze two situations in particular, large penumbrae and multiple blockers shadows (close-ups in red squares). Overall Shadows rendered with VSSM are very close to the reference. For the large penumbrae, the results of all methods are close to the reference and just a little bit of banding can be noticed in PCSS case. In the case of multiple blockers, the difference between our method and the reference becomes more obvious. It is because VSSM is based on planar assumption of PCSS and will average the blocker depth so that the umbra is underestimated. The rendering result of PCSS exhibits the same effect as VSSM. Backprojection method can achieves more physically correct result, but its performance is slow for real-time applications. We further compare VSSM with SAVSM [Lauritzen07], see Fig. 7.8. All the three close-up regions contain multiple depth layers. Hence, for SAVSM the "non-planarity" lit case happens. The side-by-side comparisons clearly show that our kernel subdivision scheme successfully removes incorrectly lit areas at very reasonable performance cost.

(a) Ground Truth (Ray-Traced) (b) VSSM (148 fps)

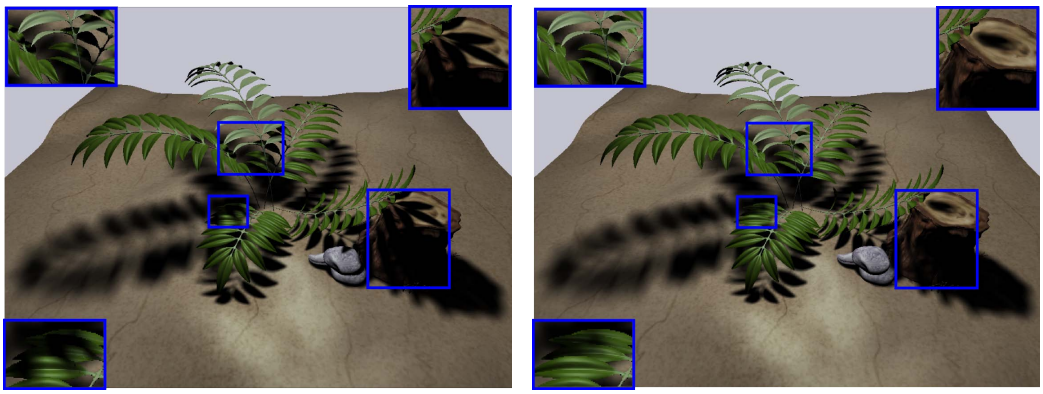


(c) PCSS (10 fps) (d) Backprojection (19 fps)

Figure 7.7: Shadow quality comparison of several methods (SM size 512×512**, scene has 212K faces): ray-tracing (a), our VSSM method using mixed subdivision scheme (b), percentage closer soft shadows [Fernando05a] (c), backprojection [Guennebaud07] (d).**

7.6.1 Limitations

Our VSSM method shares the same failure cases as PCSS. The PCSS method assumes that all blockers have the same depth within the filter kernel. Such a "single blocker depth assumption" essentially flattens blockers. When the light size becomes bigger, this assumption is more likely to be violated and umbrae tend to be underestimated. Furthermore, PCSS only generates one depth map from the center of the light source. When using a single depth map to deal with blockers of a high depth range, single silhouette artifacts [Assarsson03] may appear. Actually, all the existing PCSS-based soft shadow methods [Annen08a] [Lauritzen07] share these problems. Nevertheless, in most cases, the soft shadow generated by VSSM is visually plausible and looks very similar to the ray-traced reference.



(a) VSSM (148 fps) (b) SAVSM (183 fps)

Figure 7.8: Shadow quality comparison of between VSSM (a) and SAT-based variance shadow map (SAVSM) [Lauritzen07] (b).

7.7 Summary

In this chapter, we have presented *variance soft shadow mapping* (VSSM) for rendering plausible soft shadow. VSSM is based on the theoretical framework of *percentage-closer soft shadows*. In order to estimate the average blocker depth for each scene point, a novel formula is derived for its efficient computation based on the VSM theory. We solve the classical "non-planarity" lit problem by subdividing the filtering kernel, which removes artifacts. As future work, we would like to apply our kernel subdivision method to exponential shadow mapping [Annen08b], which is also a *single-bounded* pre-filterable shadow mapping method.

Part IV

Interactive Global Illumination in Participating Media

Chapter 8

Interactive Volume Caustics in Single-Scattering Media

8.1 Introduction

The GI effects increase the realism of computer generated scenes significantly. Caustics caused by specular or refractive objects are stunning visual effects, even more so in participating media, where *volumetric caustics* can be observed, see Fig. 1.5.

However, most existing methods for computing volumetric caustics are computationally expensive, preventing interactive applications from including this appealing effect. In this paper we propose a novel interactive volume caustics rendering method for single-scattering participating media. We derive a simplified physics-based model enabling the efficient rendering of volumetric caustics in participating media exhibiting variations in scattering and absorption coefficients as well as the, potentially anisotropic, scattering phase function. We describe a practical GPU-based implementation and evaluate the technique in detail.

Our method avoids all pre-computations, enabling the interactive simulation of light interaction with fully dynamic refractive and reflective objects while maintaining good temporal coherence in animated renderings. Since large and complex scenes can be handled efficiently by our technique, the rendering of volumetric caustics in interactive applications like computer games is becoming an option. Additionally, our method is applicable for fast preview generation of a more complex lighting simulation like volumetric photon mapping [Jensen98], as required e.g. for feature films and in commercial rendering packages.

In brief, we present the following contributions in this chapter:

- We derive a theoretically grounded simplified image formation model for

volume caustics in single-scattering media, and,

- Based on a reformulation of the proposed model, we develop a screen-based interactive rendering technique that uses line primitives [Krüger06, Sun08] to efficiently splat radiance contributions to image pixels.

The remainder of this chapter is organized as follows: First, Section 8.2 gives a short overview of our method. We then derive a simplified image formation model for volume caustics in single-scattering media, Section 8.3, the implementation of which on graphics hardware is covered in Section 8.4. Section 8.5 presents experimental results and comparison against existing techniques. We then conclude the paper in Section 8.6 and discuss avenues for future research.

8.2 Overview

Image formation for volumetric caustics is an involved process. We thus start with an overview of our algorithm, Fig. 8.1. The radiance estimate for an image pixel in the presence of a participating medium and specular objects can be split into three separate components:

- radiance scattered into the viewing direction by the participating medium for incident rays *not* having interacted with specular objects (A),
- radiance scattered into the viewing direction from incident rays that experienced specular reflection or refraction events prior to the scattering event (B), and
- surface radiance, possibly illuminated by a caustic (C).

Since superposition of light is linear, the final image (D) can be computed by summing the three components. For steps (A) and (C) we employ previously developed algorithms. Our focus in this paper is on the efficient computation of component (B).

On a coarse level, our technique employs the following steps: First, we render the airlight and volumetric shadow contributions (A) using the anisotropic version of Sun et al.'s model [Sun05] computed with the algorithm of Wyman and Ramsey [Wyman08a]. Second, the image with the volume caustics (B) is generated by computing the light paths from the light source that are reflected or refracted at a specular object. We draw these paths as *lines* and directly compute radiance contributions for each of the affected pixels which are summed up over all line segments representing light rays. The algorithm can be seen as a radiance splatting operation that emulates GPU *ray marching*. This rendering pass is our

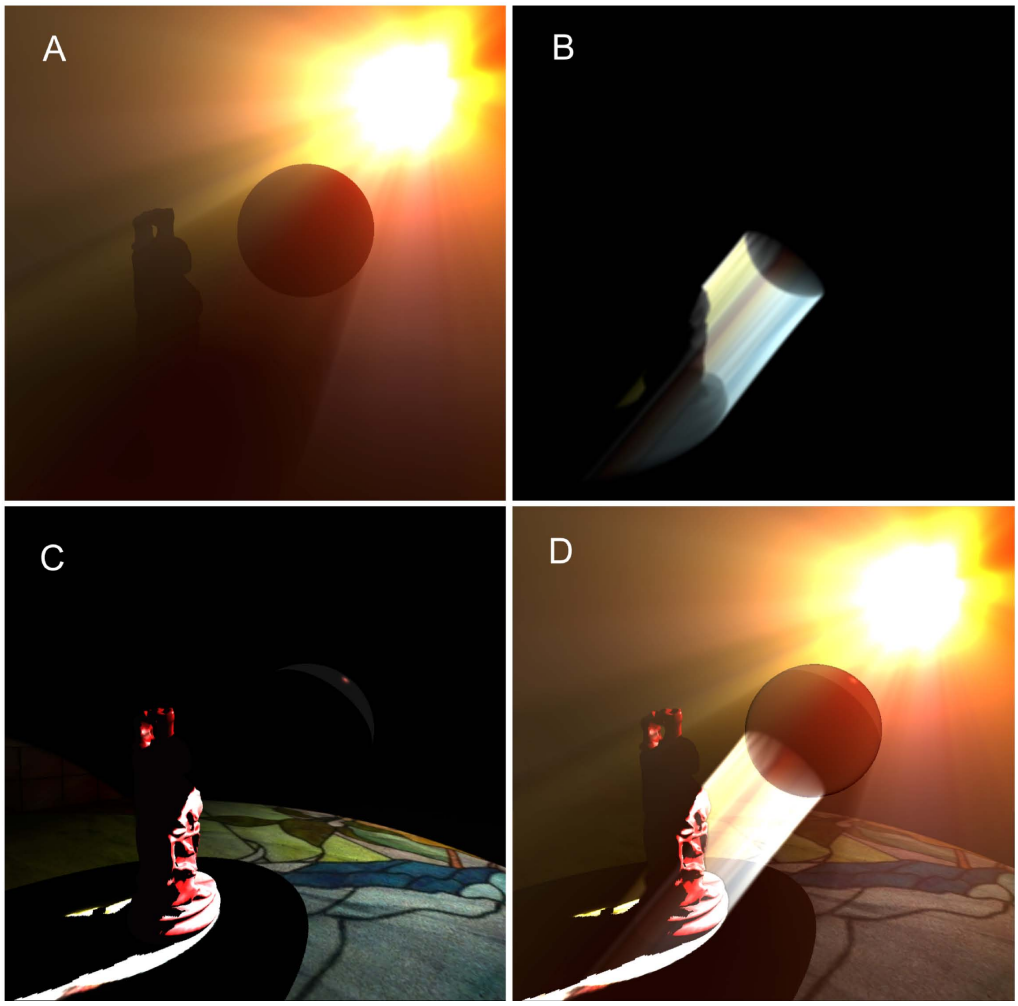


Figure 8.1: The final image (D) is composed of an airlight image with a shadow volume (A), a volume caustic image (B) and the illumination of the surface (C).

L_{in}	incoming radiance in caustic volume
L_{ls}	light source radiance
$_s(\mathbf{x})$	scattering cross-section
$_a(\mathbf{x})$	absorption cross-section
$(\mathbf{x})$	extinction coefficient $= _a + _s$
$\Omega_0(\mathbf{x})$	albedo of participating medium $\Omega_0(\mathbf{x}) = \frac{_s}{}$
$(\mathbf{a},\mathbf{b})$	transmittance from **a** to **b**: $\exp(-\int_{\mathbf{a}}^{\mathbf{b}} (\mathbf{x})dx)$
p	scattering phase function
$d_{\mathbf{ab}}$	distance between points **a** and **b**
T	Fresnel transmittance (or reflectance)

Figure 8.2: A summary of the notation used in this paper.

main contribution and it is described in detail in the following sections. Third, we generate an image of the surface illumination and the surface caustics (C) using Wyman's hierarchical caustic map (HCM) algorithm [Wyman08b]. The final image is then the sum of these three images.

8.3 Line-Based Volume Caustics

Our goal in this section is to derive a physically accurate model for volume caustics in single scattering media. We concentrate on the radiance contributed to the image by single scattering in the caustic volume, i.e. step (B) in Fig. 8.1.

In the presence of a participating medium the ray integral for a viewer at position **v** looking into direction Ω is given by

$$L(\mathbf{v},\Omega) = \int_{Ray} (\mathbf{v},\mathbf{x}) \ _s(\mathbf{x}) \int_{\Omega} p(\mathbf{x},\Omega,\Omega')L_{in}(\mathbf{x},\Omega')d\Omega' dx, \tag{8.1}$$

where denotes transmittance, $_s$ the scattering cross section, and p the scattering phase function (see also Fig. 8.3). Eq. 8.1 requires the computation of the incident radiance distribution L_{in} at every point **x** along a viewing ray.

Since radiance, in the absence of ray attenuation, is an optical invariant along the ray even when passing refractive objects [Veach98], L_{in} can be determined efficiently by texture look-ups for a light source texture or by simply setting radiance values. Eq. 8.1 can be discretized in a straight-forward manner as

$$L(\mathbf{v},\Omega) \approx \sum_i (\mathbf{v},\mathbf{x}_i) \ _s(\mathbf{x}_i) \sum_j p(\mathbf{x}_i,\Omega,\Omega'_j)L_{in}(\mathbf{x}_i,\Omega'_j)\Delta\Omega\Delta x, \tag{8.2}$$

where Δx is the constant step size of the ray marching integral and $\Delta\Omega$ is the size of the discretized solid angle. Radiance is summed over spatial positions along

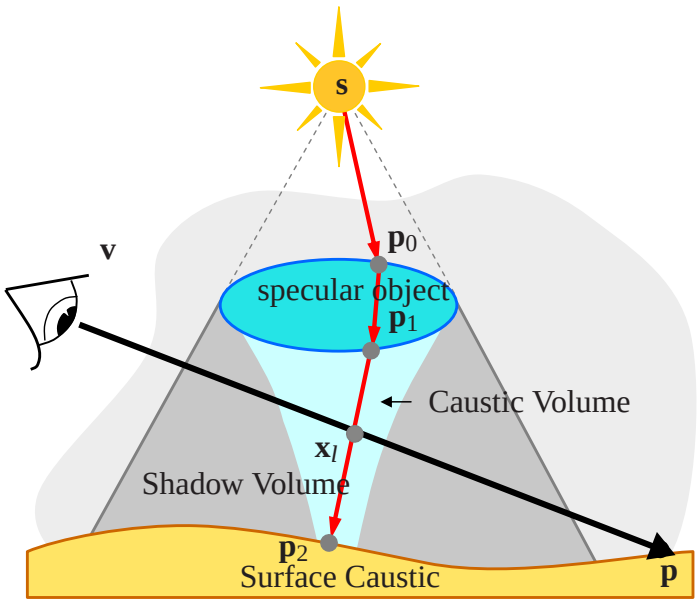


Figure 8.3: A volume caustic can be generated by computing all rays intersecting a specular object. The caustic can be rendered by drawing lines from exit points $\mathbf{p}_1$ to surface points $\mathbf{p}_2$ for all light paths interacting with the object.

the viewing ray $\mathbf{x}_i$ in all directions Ω'_j. Since we are rendering line segments to connect light paths between viewing rays and light source, see Fig. 8.3, the double sum in Eq. 8.2 can be converted into a single sum over these line segments:

$$L(\mathbf{v},\Omega) \approx \sum_l \quad (\mathbf{v},\mathbf{x}_l) \quad {}_s(\mathbf{x}_l)p(\mathbf{x}_l,\Omega,\Omega'_l)L_{in}(\mathbf{x}_l,\Omega'_l)\Delta\Omega\Delta x_l. \tag{8.3}$$

Each line segment intersecting the viewing ray introduces a light path between **s** and **v**. The connection is made at point $\mathbf{x}_l$ where the line segment passing in direction $\Omega'_l = \mathbf{p}_2 - \mathbf{p}_1$ intersects the viewing ray in direction Ω. The radiance reaching point $\mathbf{x}_l$ from **s** is equal to the original radiance at the light source attenuated by absorption and out-scatter along the ray. Thus we have an additional factor of $(\mathbf{s},\mathbf{x}_l)$ that includes the Fresnel factors T for specular reflection or refraction, e.g. $L_{in} = L_{ls}\ (\mathbf{s},\mathbf{p}_0)T_0\ (\mathbf{p}_0,\mathbf{p}_1)T_1\ (\mathbf{p}_1,\mathbf{x}_l)$ in the case of two-bounce refraction. Note that the conversion of Eq. 8.2 into Eq. 8.3 introduces the implicit assumption that the radiance distribution L_{in} in the caustic volume is zero for light rays not being represented by line segments, i.e. all light leaving the light source at **s** is reaching $\mathbf{x}_l$ via a light ray that can be represented by a line segment.

We assume the discretized solid angle $\Delta\Omega$ to be constant. This is the case if the area of the light source is small with respect to the distance $d_{\mathbf{sx}_l}$ since the size of the solid angle under which the light source appears at $\mathbf{x}_l$ is approximately

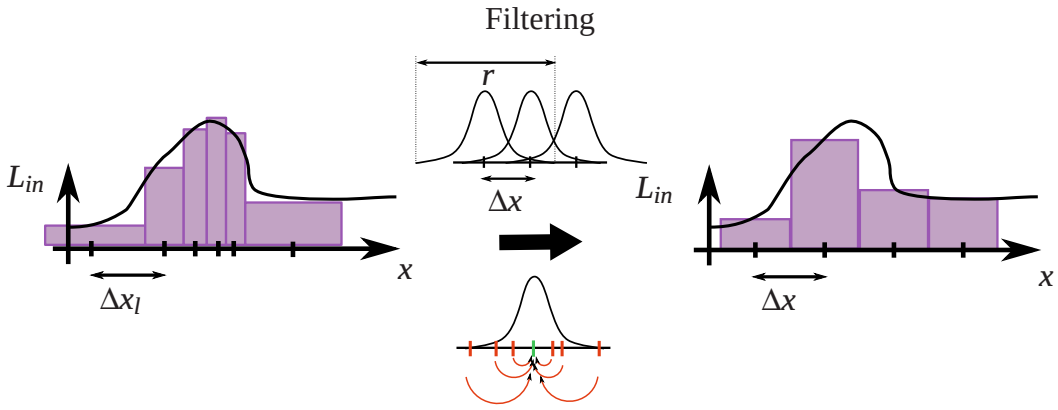


Figure 8.4: We simulate regular interval spatial integration along the viewing ray. For this, the irregular radiance samples (with a variable step size of Δx_l) caused by line rendering (left diagram) are converted via filtering into a regular representation with fixed step size Δx (right). Note that this fixed step size representation is implicit. In practice, we directly accumulate the filtered values to compute the pixel integral. The small figure above the arrow shows that the Gaussian filters with support r are centered at the regular sample points to be emulated. The lower figure indicates how the irregular samples obtained from line rendering (red) contribute to a simulated regular sample point (green). All irregular samples within the support of the filter function contribute to the corresponding regular sample point.

constant for comparatively minor path length differences between $\mathbf{s}$ and $\mathbf{x}_l$ for different incident directions.

Note that using lines as rendering primitives, we cannot ensure an even sampling of the radiance function L_{in} along the viewing ray. We cannot predict the intersection points $\mathbf{x}_l$ without rendering all line segments and storing the intersection points with all viewing rays. This would require an intermediate storage of the irradiance as in [Ihrke07, Sun08]. Thus, we cannot compute an appropriate variable spatial step size Δx_l, Fig. 8.4 (left).

Therefore, in our algorithm, we simulate a constant step size $\Delta x_l = \text{const.} = \Delta x$ for the ray marching integral, where Δx is a user parameter emulating a fixed step size as found in explicit ray marching algorithms. We re-sample the radiance function L_{in} on-the-fly while rendering the line segments. Using a filtering operation, we re-distribute the radiance function to the closest regular step points, see Fig. 8.4 (right). We employ a normalized Gaussian filter [Jensen01], the support of which can be modified as a user parameter r. We thus have derived a simplified image formation model based on physical assumptions. Our model intrinsically supports anisotropic scattering phase functions as well as inhomogeneously scattering participating media via spatially varying scattering phase functions, absorption, and

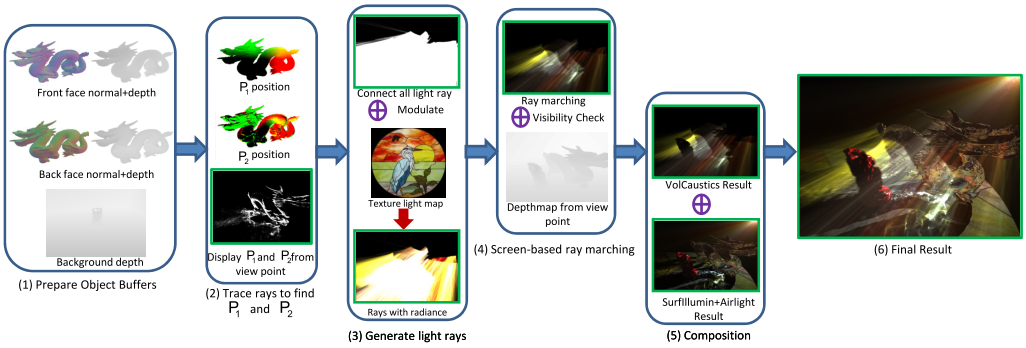


Figure 8.5: Algorithmic steps of our algorithm.

scattering coefficents. Note that the case of homogeneous media is typically more efficient in terms of implementation since the evaluation of $(\mathbf{x}_i, \mathbf{v})$ does require the sampling of a line integral in the case of inhomogeneous media.

To summarize, our simplifying assumptions are

- The image formation is dominated by the effect of single scattering, and
- The distance between light source and volume caustic points is large compared to the size of the light source.

In the following section we describe how this simplified model can be efficiently implemented on the GPU.

8.4 Implementation

In this section we discuss the implementation of our simplified image formation model. During the discussion we refer to Fig. 8.5 which shows an outline of our algorithm. We will refer to illuminating rays, such as $\mathbf{p}_1$,$\mathbf{p}_2$ in Fig. 8.3, as *light rays* whereas rays from the camera are called *viewing rays*. Our algorithm proceeds in two main stages. The first stage is the computation of three-dimensional light ray segments. The second stage then draws all those segments to the screen buffer and blends the radiance contributions according to Eq. 8.3 while emulating a proper ray marching implementation via filtered re-distribution of radiance values to neighboring regular step points, Fig. 8.4.

8.4.1 Generating Line Primitives

In this stage we compute segments of light rays after their interaction with a specular object and before hitting a diffuse surface. These line segments, if passing

through a participating medium, will generate an indirect light path between light and viewing rays by means of a scattering interaction.

In practice, we simulate one- or two-bounce reflection or refraction, respectively. For this purpose we generate a depth map of the scene excluding specular objects as seen by the light source (step 1, bottom). Additionally, we generate front- and back-facing depth and normal maps of the specular objects (step 1, upper rows). The specular object depth and normal buffers are then used to compute the light rays that are reflected or refracted from the object [Hu07]. This step results in a position $\mathbf{p}_1$ on the back-face of the object as well as a light ray direction after specular interaction. An additional intersection of these rays with the scene depth map results in $\mathbf{p}_2$. If no intersection is found, we intersect the ray with a large bounding sphere surrounding the scene. In this way, missing geometry in the depth map does not invalidate the computed light rays.

This procedure results in the two points $\mathbf{p}_1$ and $\mathbf{p}_2$, Fig. 8.3, and thus determines the light ray segment in camera space. Fig. 8.5, step 2, visualizes this *volume caustic buffer*. The image below the buffers shows a 3D rendering of the end-point positions as seen from the camera. In our implementation we store $\mathbf{p}_1$ and $\mathbf{p}_2$ into the same texel position of two different textures. In a subsequent pass we render point primitives to achieve a read back of $\mathbf{p}_1$ and $\mathbf{p}_2$ from the textures into the geometry shader stage. This way, a line primitive can be constructed. Initially, the coordinate values of the end points are in light space; we transform them into camera space in this stage. After the geometry shader, the newly created line primitive will be automatically rasterized before entering the pixel shader stage.

8.4.2 Light Ray Blending

In step 3, we modulate all light rays with a light source texture. Note that a rendering of all lines with a radiance value picked directly from the texture would result in over-exposed images as shown in the image at the bottom of step 3.

To properly compute the correct radiance contributions we employ a fragment shader implementing Eq. 8.3. This step requires the use of the front- and back-facing object buffers from step 1, the volume caustic buffer, step 2, and the modulation texture from step 3. They all share the same resolution and corresponding pixels in the buffers describe different properties of the light ray $\mathbf{s}, \mathbf{p}_2$. The fragment shader first computes the point of intersection $\mathbf{x}_l$ between the viewing ray and the light ray. Next, to properly emulate the regularly discretized ray marching integral, Fig. 8.4 , we compute the regular step points along the ray that are within the filter support.

This results in a set $\{\mathbf{x}_i | i = 1 \ldots N\}$ of regularly spaced points in the vicinity of $\mathbf{x}_l$. N is typically in the range of $2-4$. For each point in this set we determine the light path described by the vertices $(\mathbf{v}, \mathbf{x}_i, \mathbf{p}_1, \mathbf{p}_0, \mathbf{s})$ and compute per segment

attenuation and Fresnel transmission factors. Combining the attenuation factors and the light source radiance from the modulation texture, step 3, weighted by the kernel value, we obtain the incident radiance value L_{in} at $\mathbf{x}_i$. The ray direction remains $\Omega'_l = \mathbf{p}_2 - \mathbf{p}_1$. This way, we can directly evaluate parts of the sum in Eq. 8.3 and add the appropriate radiance contribution to the pixel value.

8.4.3 Visibility and Remaining Illumination Components

In step 4 we compute visibility of the light rays as seen from the camera. This step requires a scene depth map in camera coordinates. The depth map includes the specular objects. We simply cull light ray fragments if they are behind objects. This step obviously introduces artifacts; refractive objects appear opaque and do not transmit light from volume caustics. This limitation is inherent in our screen-based approach. Since we do not store the irradiance distribution in the caustic volume we cannot perform volume rendering along refracted viewing rays.

Finally, we add the images containing the airlight and volumetric shadows, Fig. 8.1 (A), and the surface illumination, Fig. 8.1 (C). Note that in order to compute the radiance contribution due to the surface properly, the transmittances $(\mathbf{s},\mathbf{p})$ and $(\mathbf{v},\mathbf{p})$ have to be computed and multiplied to the surface radiance in absence of a participating medium.

8.4.4 Inhomogeneous Media

The previous description applies to the case of homogeneous media. The case of inhomogeneous media differs only slightly. Instead of computing transmittance values $(\mathbf{a},\mathbf{b})$ analytically, we now have to perform sampling and numerical integration in order to retrieve the inhomogeneous information. It might seem that this excludes the simulation of inhomogeneous media at interactive frame rates.

Note, however, that sampling is only required to compute the transmittance. The accumulation of varying albedo and phase function is not affected by the quality of the numerical integration since it is computed implicitly by summing line segment contributions. Thus, in practice, we can choose a very low sampling rate for the numerical integration. Although this assumes no high-frequency changes inside the medium, the poor approximation will only be visible in the shadow cast by the inhomogeneous medium and in slight low-frequency intensity variations within the caustic. The latter effect is usually masked by the inhomogeneous appearance and the brightness of the caustic regions, see Fig 8.10.

8.5 Results and Discussion

In this section we report on the quality and performance of our method. Our technique was implemented in OpenGL and all results were rendered on a Intel Core2 Quad Processor Q9550 with 2.83GHz using an NVIDIA GeForce 280 GTX graphics card. Except where explicitly noted, all the rendering results in the paper are generated using a volume caustics buffer resolution of 1024×1024 pixels. Similarly, the size of the screen buffer is usually 1024×1024, except in the test exploring the performance impact of the screen buffer size. We use a volume shadow buffer resolution [Wyman09] of 256×256 pixels; the size of the surface caustics buffer [Wyman08b] is 1024×1024.

8.5.1 Ground Truth Comparison

To verify the accuracy of our method, we compare volume caustics rendered with our model to both ground truth images generated with Mental Ray and images generated using the previous techniques by [Krüger06] and [Papadopoulos09]. Since the latter two techniques are very similar, and [Krüger06] does not contain explicit formulas we implement attenuation along the ray as in [Papadopoulos09] but refer to the technique itself as [Krüger06]. The results of this comparison are shown in Fig. 8.6. Since [Krüger06] does not support anisotropic and inhomogeneous media, the test scene is isotropic and homogeneous.

As can be seen from the figure, our results closely match the images rendered with photon mapping (5 min. computation time). Our technique , in comparison, runs at more than 25 fps. For the simple case of isotropic and homogeneous scenes, the technique of Krüger et al. [Krüger06] is also able to generate a similar image with slight advantages in rendering speed. The performance penalty associated with our more general and physically accurate model is about 5% for the volume caustics stage compared to Krüger et al. [Krüger06]. The overall performance drop in the full algorithm is not noticable. Note, that it is necessary to scale the output of [Krüger06] linearly to match the ground truth rendering and that the scale factor is scene dependent and cannot be easily estimated. Our method, on the other hand, runs with fixed parameter settings of $\Delta x = 0.2$.

The differences between our results and the ground truth rendering can mainly be attributed to the approximate computation of the light path segments in our algorithm. Note, however, that future improvements of GPU ray tracing directly increase the accuracy of our method since the computation of the light rays is an independent module. A second difference is that photon mapping can determine light contributions seen *through* refractive objects, which is impossible for our algorithm as discussed previously. A third effect is a slight multiple scattering component in the ground truth image whereas our result shows only single

scattering according to our model. Finally, differences can appear since we only follow the most important light paths (eg. two refractions inside the object).

A comparison between ground truth and our algorithm for the anisotropic case is shown in Fig. 8.7. While there are differences in the images, mainly concerning brightness and sharpness of the caustics, the overall characteristic of the medium is captured by our technique. The smoother look of the ground truth result can be attributed to multiple scattering effects.

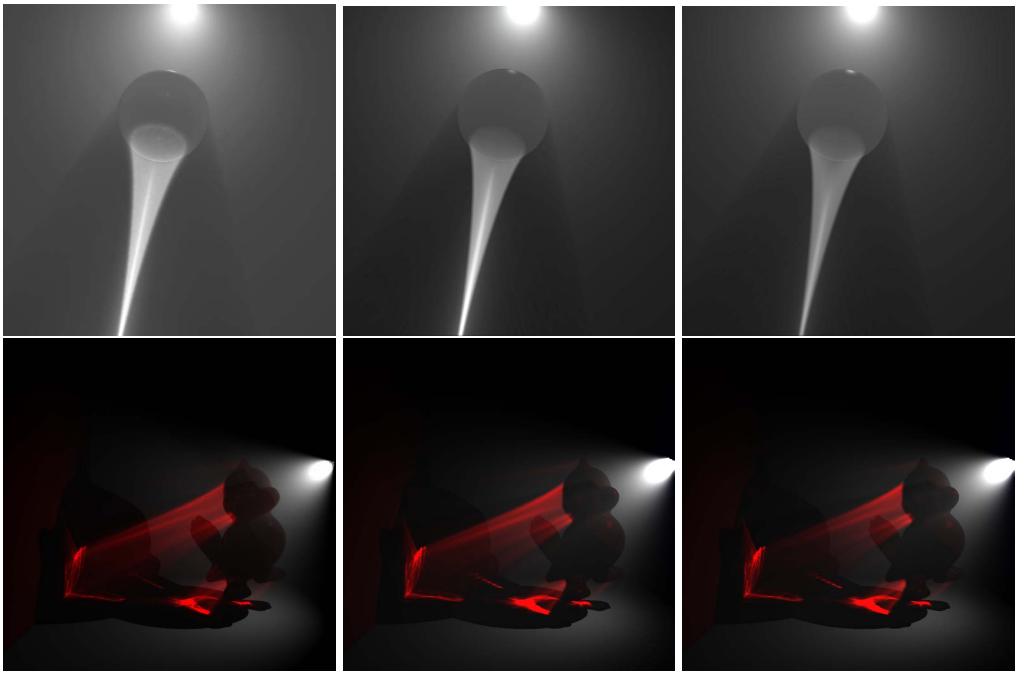


Figure 8.6: Comparison of volumetric photon mapping as implemented in MentalRay (left), our method (middle), Krüger et al. [2006] linearly scaled with a manually determined value to match ground truth (right, scale factors are 0.06 and 0.02 for the top and bottom results). The ground truth results are generated with Mental Ray, using 2 million photons and take around 5 minutes to compute. All results generated by our method use a 1024×1024 **volume caustics buffer and can be rendered with more than 25 fps.**

8.5.2 Performance Analysis

To assess the performance characteristics of our technique we experimented with three test scenes shown in Fig. 8.8. The scenes are increasingly complex in terms of volume caustic computation. The ring scene demonstrates one-bounce reflection, the Buddha scene is rendered with two-bounce refraction, and the gemstone

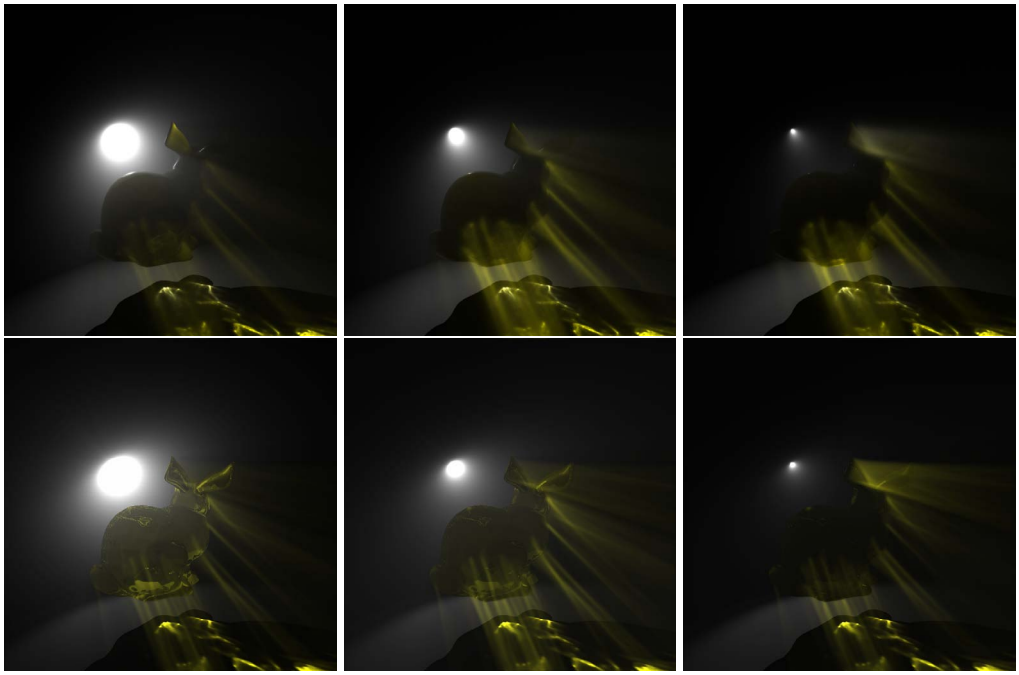


Figure 8.7: Comparison of volume photon mapping in anisotropic participating media (top row) with results of our method (bottom row). The phase function uses the Henyey-Greenstein model as implemented by MentalRay. The anisotropy parameters are $g = 0.7$ (left), $g = 0.0$ (middle), and $g = -0.7$ (right) for forward, isotropic and backward scattering, respectively.

scene includes both effects. The timing data in Table 8.1 shows the rendering

	Rendering Stage				
Scene	AL	SC	VC	Comp.	FPS
Ring	4.5 ms	15 ms	10 ms	4 ms	28
Buddha	5 ms	13 ms	9 ms	4 ms	31
Gemstone	5 ms	28 ms	21 ms	4 ms	17

Table 8.1: Timing data of different rendering stages in Fig. 8.8. AL = airlight and volume shadows , Fig. 8.1 (A), SC = surface caustics (B), VC = volume caustics (C), Comp. = composition (D), FPS - overall performance.

time for each rendering stage. We can see that only rendering the volumetric caustics runs at over 80 fps on average when using a caustics buffer resolution of 1024×1024. For the gemstone scene, we compute the volume caustics generated by both two-bounce refraction and one-bounce reflection. The total number of caustic buffers doubles in this case. From the timing data in the table, we see that

the time required for the surface and volume caustics stages is also approximately doubled compared to the scenes with a single caustic buffer. This suggests that our method scales linearly with the number of light rays involved in the caustics computation. Since HCM [Wyman08b] contains an additional overhead due to hierarchical processing compared to the volume caustics stage, for usual scene settings with a low number of pixels affected by volume caustics the surface caustics stage dominates rendering time. As a second test, we evaluated the dependence of

	Screen Buffer Resolution		
CB Res.	256^2	512^2	1024^2
256^2	193	173	171
512^2	163	140	115
1024^2	93	74	62

Table 8.2: Timing data for different caustic buffer (CB Res.) and screen buffer resolutions. Numbers are for the Buddha scene and are given in frames per second.

our algorithm on screen size and caustic buffer resolution. Table 8.2 shows that an increasing caustics buffer size has more severe consequences than increasing the screen resolution. Of course, this finding depends on the number of pixels covered by lines. The other two test scenes show similar performance characteristics. Note however, that the timings are for the volume caustics rendering step only. Since this part of our algorithm only consumes about 30% of the overall computation time, the performance gains for lower resolution buffers are less dramatic in a realistic scenario.

8.5.3 Influence of User Parameters

Several parameters affect the image quality and rendering performance of our method. The first is the volume caustics buffer resolution. All results shown in this paper use 1024×1024 as the resolution for the volume caustics buffer. In Fig. 8.9 (left) we show results with different volume caustics buffer resolutions but keep the surface caustics buffer resolution at 1024×1024 pixels. A higher resolution results in better image quality, however, the volume caustics quality is still reasonable even though details appear slightly blurred in the low resolution case. Another user parameter is the size of the filter kernel r. We typically choose the support of the filter equal to the step size Δx. In Fig. 8.9 (right) we show results for drastically different settings. If the support of the filter is chosen too low, noise is being generated in the images. Finally, the number of light sources influences the performance of our method. Each light source requires separate

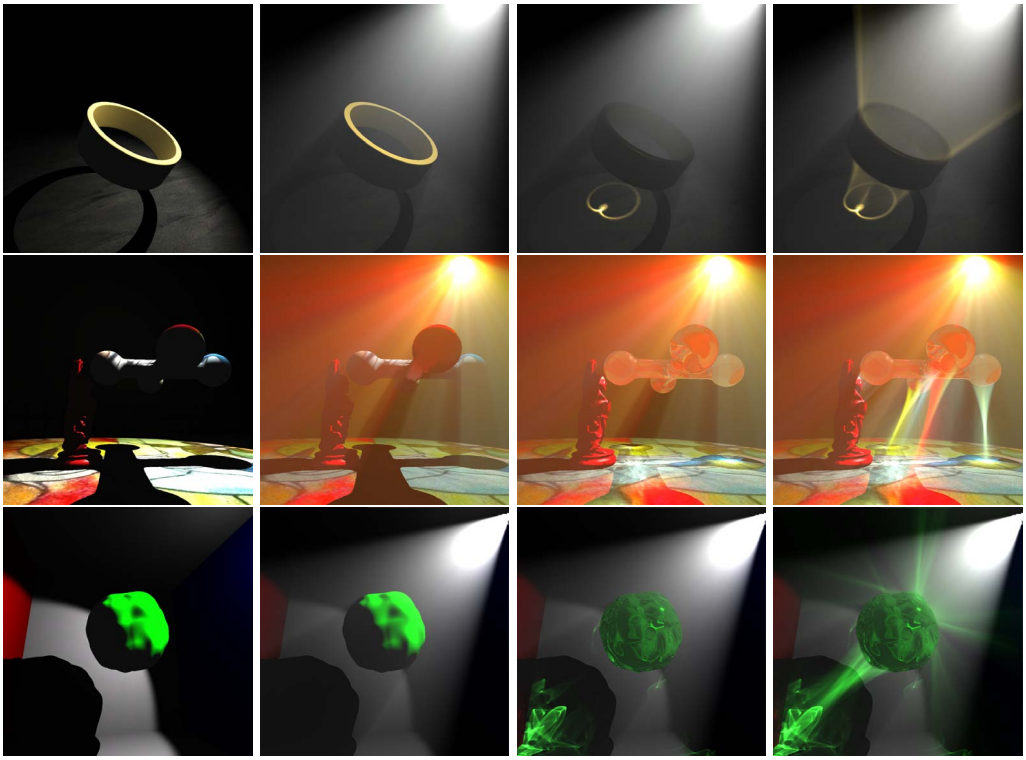
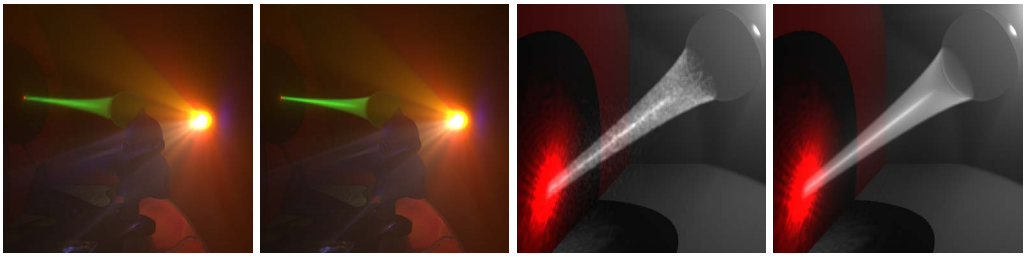


Figure 8.8: Different combinations of illumination methods for our test scenes. From left to right, the following type of illumination is included: Direct illumination, airlight and volume shadows, surface caustics, and volume caustics. The ring scene (top row) is rendered with one-bounce reflection, the Buddha scene (middle row) illustrates two-bounce refraction and the gemstone scene (bottom row) includes both one-bounce reflection and two-bounce refraction.

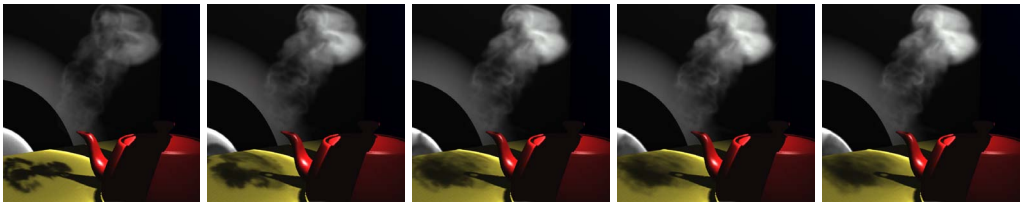


(a) 256^2, 39 fps (b) 1024^2, 29 fps (c) $r = 0.1 \cdot \Delta x$ (d) $r = \Delta x$

Figure 8.9: Left: Different volume caustics buffer resolution. Right: Different filter size.

caustic buffers. As we have seen in Table 8.1, the performance drops approximately linearly with the number of caustic buffers. We also observe this behavior

in the case of multiple light sources; tests with one to four light sources resulted in the following numbers: 37 fps(1), 21.5 fps(2), 9.25 fps(3), 2.9 fps(4). An addi-



(a) 2 steps, 41 fps (b) 5 steps, 27 fps (c) 10 steps, 18 fps (d) 20 steps, 10.5 fps (e) 30 steps, 8 fps

Figure 8.10: Inhomogeneous smoke rendering using different number of discrete integral steps. Note that a low number of step points only affects the brightness of the volume caustic and the accuracy of the shadow. 10 integration steps already suffice to obtain a visually appealing result with only minor differences to the final solution.

tional strength of our method is the temporal coherence: animated and deforming objects can be displayed correctly without flickering. We can deal with arbitrary deformations without any pre-computation. Since images capture this feature of our method inadequately we recommend to watch the accompanying video.

8.5.4 Limitations

Our screen-based volume caustic technique achieves high performance for highly complex scene settings. However, because of its screen-based nature, the computation cost increases linearly with the effective number of screen pixels that require volume caustics computations. As an example, in the ocean scene, Fig. 1.5, almost all screen pixels (1024×1024) are involved in volume caustics computations. This is the reason for the low frame rate compared to the other examples.

Another issue is the effective resolution of the caustics buffer. Consider a large field of view for a spot light source illuminating a comparatively small specular object. Since the object occupies very few pixels in the caustic buffer, the rendering quality will suffer in this case. Another, related case is the rendering of extreme close-ups of the volume caustics. Individual lines might become visible in this case. The problem is increased for objects with very high refractive indices since light rays diverge more strongly under these circumstances.

8.6 Summary

We have presented a real-time method for rendering volume caustics in single scattering participating media. Our technique generates physically plausible results and enables the rendering of completely dynamic scenes with good temporal coherence. Anisotropic and inhomogeneous media are naturally supported by our algorithm and interactive performance can be achieved in the latter case. Our method simulates the most important light paths for volumetric caustics and introduces a theoretically founded approximation for single scattering media. Since it efficiently renders large and dynamic scenes we believe that it can be applied in computer games and as a preview for more sophisticated lighting simulations.

An interesting direction for future work are hierarchical representations of the caustic buffer. Currently we use separate buffers for surface and volume caustics. However, the two buffers share common information and speed-ups can potentially be gained by combining them into a single buffer. A hierarchical approach similar in spirit to [Wyman08b] could potentially increase the performance significantly. Another research direction is the approximation of multiple scattering effects in participating media. Since line primitives are performing very well in the case of single scattering media an interesting question is whether this power can be harnessed for multiple scattering approximations.

Chapter 9

Conclusions and Future Work

The driving motivation of this thesis is to generate photorealistic global illumination rendering of arbitrary scenes. To achieve this goal, some reasonable approximations have to be developed to find a visually plausible compromise between quality and performance. This thesis focuses on real-time visually pleasing global illumination rendering for fully-dynamic scenes using graphics hardware.

We have identified several problems which need to be solved. First, the explicit visibility check in radiosity-like methods is expensive, and we present an *implicit visibility* scheme to tackle this problem efficiently. Then, the most important GI effect: realistic soft shadow is difficult to be achieved with real-time performance because of its visibility determination. We successively propose the *convolution soft shadow map* (CSSM) and *variance soft shadow map* (VSSM) methods to render visually plausible soft shadows with real-time frame rates. Further, we apply the CSSM method to approximately solve the visibility problem of real-time indirect lighting. Finally, the volume caustics in participating media is a kind of non-trivial and time-consuming GI effect. Our novel volume caustics rendering method achieves high-quality results at real-time/interactive frame rates for complex dynamic scenes containing homogeneous/inhomogeneous participating media.

9.1 Summary

We will quickly summarize our algorithms, what kind of approximations they are involved with and how they constitute advancements over existing techniques.

9.1.1 Implicit Visibility

We present a new global illumination method that builds on and extends the traditional hierarchical radiosity [Hanrahan91] approach. Compared with explicitly evaluating visibility using ray casting, our method tackles the visibility problem by implicitly evaluating mutual visibility while constructing a hierarchical link structure between scene elements. This new concept circumvents time-consuming explicit visibility queries, which is the main performance bottleneck in traditional approaches. Our method allows for rendering of full global illumination solutions for moderately complex and arbitrarily deforming dynamic scenes at near-real-time frame rates on a single PC. It faithfully reproduces a variety of complex lighting effects including diffuse and glossy interreflections, and handles scenes featuring environment map and area light sources.

9.1.2 Pre-filtering Soft Shadow Maps and their applications

We successively propose CSSM and VSSM for rendering visually plausible soft shadows in real-time. Both methods are based on the pre-filtering theory of shadow mapping and implemented in the percentage closer soft shadow (PCSS) [Fernando05a] framework.

CSSM is based on the convolution theory [Annen07] and can achieve several hundred frames per second for a single area light source if using hardware supported mipmapping for filtering. Therefore, it is fast enough to render many area light sources simultaneously. We have shown that environment map lighting for dynamic objects can be incorporated by decomposing the lighting into a collection of area lights, which are then rendered using the CSSM technique. Furthermore, we apply the CSSM technique in computing indirect lighting. We demonstrate that indirect visibility can be approximated with a small number of area lights in combination with our CSSM technique. Due to the fast computation time of the CSSM, we can display visually plausible approximated indirect illumination at real-time performance.

In order to reduce memory consumption and improve the performance of CSSM, we propose the VSSM method which is based on a one-tailed version of Chebyshev's inequality [Donnelly06a]. We introduce new formulation for achieving efficient computation of (average) blocker distances based on pre-filtering, a common bottleneck in PCSS-based methods [Lauritzen07]. Furthermore, we avoid incorrectly lit pixels by appropriately subdividing the filter kernel. We demonstrate that VSSM renders high quality soft shadows efficiently (usually over 100 fps) for complex scene settings. Its speed is at least one order of magnitude faster than PCSS [Fernando05a] for large penumbra. Such a great performance of VSSM makes it possible to be applied in game development.

9.1.3 Volume Caustics

We have presented a real-time method for rendering volume caustics in single scattering participating media. Our method is based on the observation that line rendering of illumination rays into the screen buffer establishes a direct light path between the viewer and the light source. The radiance contributions of these light paths to each of the pixels can be computed and accumulated independently in GPU. Our technique generates physically plausible results and enables the rendering of completely dynamic scenes with good temporal coherence. Anisotropic and inhomogeneous media are naturally supported by our algorithm and interactive performance can be achieved in the latter case. Our method simulates the most important light paths for volumetric caustics and introduces a theoretically founded approximation for single scattering media. Since it efficiently renders large and dynamic scenes we believe that it can be applied in video games and as a preview for more sophisticated lighting simulations [Jensen98]. Furthermore, our "line rendering of illumination rays" concept can also be applied in ray tracing, and a recent research paper [Sun10] demonstrates its reasonability.

9.2 Conclusions and Future Works

We introduce a set of novel algorithms and techniques using graphics hardware to achieve real-time visually pleasing rendering for GI effects, even with participating media. The rendering results are are visually comparable to offline rendering but are achieved at high frame rates. This significant speed-up is achieved by introducing reasonable approximations in the theory of GI rendering. Our approximations save lots of computation cost but ensure the rendering quality is visually plausible. Furthermore, all of our methods impose no limitations for the input scenes, so that it could be applied in real interactive applications.

All of our current algorithms are tested on reasonably large-scale scenes. However, in real film or game applications, the input scene is usually in extremely complex and the data amount is in billions of triangles [Pantaleoni10]. In such a scenario, some out-of-core stream-based geometry processing algorithm will get involved with the design of global illumination rendering. Our next step in real-time GI rendering would be investigating the real-time visually plausible GI for such kind of super-scale fully-dynamic scenes. Future graphics hardware will unify CPU and GPU into a many-core platform, like Larrabee [Seiler08], and hence the programmability of our graphics pipeline will become extremely flexible. In a next step, we will probably develop new real-time GI rendering algorithms based on such a many-core platform.

Bibliography

[Agarwal03] SAMEER AGARWAL, RAVI RAMAMOORTHI, SERGE BELONGIE, AND HENRIK WANN JENSEN. Structured Importance Sampling of Environment Maps. *ACM Trans. Graph.*, 22(3):605–612, 2003. 44

[Aila04] TIMO AILA AND SAMULI LAINE. Alias-Free Shadow Maps. In *Proc. of EGSR*, pages 161–166, 2004. 24

[Akerlund07] OSKAR AKERLUND, MATTIAS UNGER, AND RUI WANG. Precomputed Visibility Cuts for Interactive Relighting with Dynamic BRDFs. In *Proc. of Pacific Graphics*, pages 161–170, 2007. 49

[Annen07] THOMAS ANNEN, TOM MERTENS, PHILIPPE BEKAERT, HANS-PETER SEIDEL, AND JAN KAUTZ. Convolution Shadow Maps. In *Proc. of EGSR*, volume 18, pages 51–60, 2007. 5, 24, 26, 44, 46, 48, 76, 77, 78, 79, 86, 142

[Annen08a] THOMAS ANNEN, ZHAO DONG, TOM MERTENS, PHILIPPE BEKAERT, HANS-PETER SEIDEL, AND JAN KAUTZ. Real-time, all-frequency shadows in dynamic scenes. *ACM Trans. Graph. (Proc. of SIGGRAPH 2008)*, 27(3):1–8, 2008. 4, 49, 94, 102, 107, 117, 120

[Annen08b] THOMAS ANNEN, TOM MERTENS, HANS-PETER SEIDEL, EDDY FLERACKERS, AND JAN KAUTZ. Exponential shadow maps. In *GI '08: Proceedings of Graphics Interface 2008*, pages 155–161, 2008. 24, 26, 44, 46, 121

[Arbree05] ADAM ARBREE, BRUCE WALTER, AND KAVITA BALA. Pre-Processing Environment Maps for Dynamic Hardware Shadows. Technical report, Dept. of Computer Science, Cornell University, 2005. 44

[Ashikhmin00] M. ASHIKHMIN AND P. SHIRLEY. An Anisotropic Phong BRDF Model. *Journal of Graphics Tools*, 5(2):25–32, 2000. 15

[Assarsson03] U. ASSARSSON AND T. AKENINE-MÖLLER. A Geometry-Based Soft Shadow Volume Algorithm Using Graphics Hardware. *ACM Trans. Graph.*, 22(3):511–520, July 2003. 28, 47, 73, 82, 120

[Atty06] LIONEL ATTY, NICOLAS HOLZSCHUCH, MARC LAPIERRE, JEAN-MARC HASENFRATZ, CHUCK HANSEN, AND FRANÇOIS SILLION. Soft shadow maps: Efficient sampling of light source visibility. *Computer Graphics Forum*, 25(4), 2006. 47

[Bavoid08] LOUIS BAVOID. Advanced soft shadow mapping techniques. In *GDC 2008*, 2008. 118

[Bavoil09] LOUIS BAVOIL AND MIGUEL SAINZ. Image-Space Horizon-Based Ambient Occlusion . In *ShaderX 7: Advanced Rendering Techniques*. Charles River Media, 2009. 43

[Beckmann63] P. BECKMANN AND A. SPIZZICHINO. *The Scettering of Electromagnetic Waves from Rough Surfaces*. McMillan, 1963. 14

[Ben-Artzi08] ANER BEN-ARTZI, KEVIN EGAN, FRÉDO DURAND, AND RAVI RAMAMOORTHI. A precomputed polynomial representation for interactive BRDF editing with global illumination. *ACM Trans. Graph.*, 27:13:1–13:13, 2008. 42

[Blinn77] J. BLINN. Models of Light Reflection For Computer Synthesized Pictures. In *Proc. of ACM SIGGRAPH*, pages 192–198, 1977. 15

[Blythe06] DAVID BLYTHE. The Direct3D 10 system. *ACM Trans. Graph.*, 25:724–734, 2006. 34

[Born64] M. BORN AND E. WOLF. *Principles of Optics*. Pergamon Press, 2nd edition, 1964. 11, 16, 17

[Boudet05] ANTOINE BOUDET, PAUL PITOT, DAVID PRATMARTY, AND MATHIAS PAULIN. Photon Splatting for Participating Media. In *Proc. of GRAPHITE*, pages 375–384, 2005. 50

[Brabec02] STEFAN BRABEC, THOMAS ANNEN, AND HANS-PETER SEIDEL. Shadow Mapping for Hemispherical and Omnidirectional Light Sources. In *Proc. of CGI*, pages 397–408, 2002. 58, 94

[Bunnell05] M. BUNNELL. Dynamic Ambient Occlusion and Indirect Lighting. In Matt Pharr, editor, *GPU Gems 2*, chapter 2, pages 223–233. Addison Wesley, March 2005. 42, 59, 60, 114, 115

[Carr03] N. CARR, J. HALL, AND J. HART. GPU Algorithms for Radiosity and Subsurface Scattering. In *Proc. of Graphics Hardware*, pages 51–59, July 2003. 41, 95

[Catmull74] EDWIN E. CATMULL. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. PhD thesis, Dept. of CS, U. of Utah, December 1974. 33

[Cerezo05] EVA CEREZO, FREDERIC PEREZ-CAZORLA, XAVIER PUEYO, FRANCISCO SERON, AND FRANÇOIS SILLION. A Survey on Participating Media Rendering Techniques. *the Visual Computer*, pages 303–328, 2005. 31, 50

[Chen90] S. CHEN. Incremental Radiosity: An Extension of Progressive Radiosity to an Interactive Image Synthesis System. *Computer Graphics (Proc. of SIGGRAPH'90)*, 24(4):135–144, August 1990. 41

[Cheslack-Postava08] EWEN CHESLACK-POSTAVA, RUI WANG, OSKAR AKERLUND, AND FABIO PELLACINI. Fast, Realistic Lighting and Material Design using Nonlinear Cut Approximation. *ACM Trans. Graph. (Proc. of SIGGRAPH ASIA 2008)*, 27(5), 2008. 49

[Chin92] NORMAN CHIN AND STEVEN FEINER. Fast object-precision shadow generation for area light sources using BSP trees. In *Proc. of ACM SI3D*, pages 21–30, 1992. 46

[Clarberg05] PETRIK CLARBERG, WOJCIECH JAROSZ, TOMAS AKENINE-MÖLLER, AND HENRIK WANN JENSEN. Wavelet Importance Sampling: Efficiently Evaluating Products of Complex Functions. *ACM Trans. Graph.*, 24(3):1166–1175, 2005. 44

[Cohen85] MICHAEL F. COHEN AND DONALD P. GREENBERG. The hemi-cube: a radiosity solution for complex environments. In *Proc. of ACM SIGGRAPH*, pages 31–40, 1985. 41

[Cohen88] MICHAEL F. COHEN, SHENCHANG ERIC CHEN, JOHN R. WALLACE, AND DONALD P. GREENBERG. A progressive refinement approach to fast radiosity image generation. In *Proc. of ACM SIGGRAPH*, pages 75–84, 1988. 41

[Cohen93] M. COHEN AND J. WALLACE. *Radiosity and Realistic Image Synthesis*. Academic Press Professional, Cambridge, MA, 1993. 20, 41, 55, 63

[Cook84] R. L. COOK, T. PORTER, AND L. CARPENTER. Distributed Ray Tracing. In *Proc. of ACM SIGGRAPH*, pages 137–145, Minneapolis, Minnesota, July 1984. 40

[Coombe04] G. COOMBE, M. HARRIS, AND A. LASTRA. Radiosity on Graphics Hardware. In *Proc. of Graphics Interface*, pages 161–168, 2004. 41

[Crow77] F. CROW. Shadow Algorithms for Computer Graphics. In *Proc. of ACM SIGGRAPH*, pages 242–248, July 1977. 22, 46

[Crow84] FRANKLIN C. CROW. Summed-area tables for texture mapping. In *Proc. of SIGGRAPH '84*, pages 207–212, 1984. 26, 44, 76

[Dachsbacher05] C. DACHSBACHER AND M. STAMMINGER. Reflective Shadow Maps. In *Proc. of ACM I3D*, pages 203–213, 2005. 43, 92, 99, 102

[Dachsbacher06] C. DACHSBACHER AND M. STAMMINGER. Splatting Indirect Illumination. In *Proc. of ACM I3D*, pages 93–100, 2006. 43, 92

[Dachsbacher07] C. DACHSBACHER, M. STAMMINGER, G. DRETTAKIS, AND F. DURAND. Implicit Visibility and Antiradiance for Interactive Global Illumination. *ACM Trans. Graph.*, 26(3), August 2007. 48

[Davidovič10] TOMÁŠ DAVIDOVIČ, JAROSLAV KŘIVÁNEK, MILOŠ HAŠAN, PHILIPP SLUSALLEK, AND KAVITA BALA. Combining global and local virtual lights for detailed glossy illumination. *ACM Trans. Graph.*, 29:143:1–143:8, 2010. 52

[Davis07] SCOTT T DAVIS AND CHRIS WYMAN. Interactive Refractions with Total Internal Reflection. In *Proc. of Graphics Interface*, pages 185–190, 2007. 49

[Décoret05] XAVIER DÉCORET. N-Buffers for efficient depth map query. *Computer Graphics Forum*, 24(3), 2005. 116

[Deussen02] O. DEUSSEN, C. COLDITZ, M. STAMMINGER, AND G. DRETTAKIS. Interactive Visualization of Complex Plant Ecosystems. In *Proc. of IEEE Visualization*, pages 219–226, 2002. 51

[Dobashi00] YOSHINORI DOBASHI, KAZUFUMI KANEDA, HIDEO YAMASHITA, TSUYOSHI OKITA, AND TOMOYUKI NISHITA. A Simple, Efficient Method for Realistic Animation of Clouds. In *Proc. of ACM SIGGRAPH*, pages 19–28, 2000. 50

[Dobashi02] YOSHINORI DOBASHI, TSUYOSHI YAMAMOTO, AND TOMOYUKI NISHITA. Interactive Rendering of Atmospheric Scattering Effects using Graphics Hardware. In *Proc. of Graphics Hardware*, pages 99–107, 2002. 50

[Dong07] ZHAO DONG, JAN KAUTZ, CHRISTIAN THEOBALT, AND HANS-PETER SEIDEL. Interactive Global Illumination Using Implicit Visibility. In *Proc. of Pacific Graphics*, pages 77–86, 2007. 4

[Dong09] ZHAO DONG, THORSTEN GROSCH, TOBIAS RITSCHEL, JAN KAUTZ, AND HANS-PETER SEIDEL. Real-time Indirect Illumination with Clustered Visibility. In *Proc. of Vision, Modeling, and Visualization Workshop*, 2009. 4

[Donnelly06a] WILLIAM DONNELLY AND ANDREW LAURITZEN. Variance shadow maps. In *Proc. of ACM I3D '06*, pages 161–165, 2006. 6, 24, 26, 44, 45, 48, 108, 142

[Donnelly06b] WILLIAM DONNELLY AND ANDREW LAURITZEN. Variance Shadow Maps. In *Proc. of ACM I3D*, pages 161–165, 2006. 77

[Drettakis94] GEORGE DRETTAKIS AND EUGENE FIUME. A Fast Shadow Algorithm for Area Light Sources Using Backprojection. In *SIGGRAPH '94*, pages 223–230, 1994. 47

[Eisemann09] ELMAR EISEMANN, ULF ASSARSSON, MICHAEL SCHWARZ, AND MICHAEL WIMMER. Casting Shadows in Real Time. In *ACM SIGGRAPH Asia 2009 Courses*, 2009. 23, 44

[Ernst05] MANFRED ERNST, TOMAS AKENINE-MÖLLER, AND HENRIK WANN JENSEN. Interactive Rendering of Caustics using Interpolated Warped Volumes. In *Proc. of Graphics Interface*, pages 87–96, 2005. 51

[Estalella06] PAU ESTALELLA, IGNACIO MARTIN, GEORGE DRETTAKIS, AND DANI TOST. A GPU-driven Algorithm for Accurate Interactive Reflections on Curved Objects. In *Proc. of EGSR*, pages 313–318, June 2006. 49

[Fernando05a] RANDIMA FERNANDO. Percentage-closer soft shadows. In *ACM SIGGRAPH 2005 Sketches*, page 35, 2005. viii, xii, 5, 11, 25, 26, 47, 107, 120, 142

[Fernando05b] RANDIMA FERNANDO. Percentage-Closer Soft Shadows. In *ACM SIGGRAPH 2005 Sketches*, page 35, 2005. 74, 76, 77, 82, 85

[Garland01] M. GARLAND, A. WILLMOTT, , AND P. HECKBERT. Hierarchical Face Clustering on Polygonal Surfaces. In *Proc. of ACM I3D*, pages 49–58, 2001. 60

[George90] D. GEORGE, F. SILLION, AND D. GREENBERG. Radiosity Redistribution for Dynamic Environments. *IEEE CG&A*, 10(4):26–34, 1990. 41

[Glassner91] A. GLASSNER. *An introduction to ray tracing*. Academic Press, 1991. 19

[Glassner95] A. GLASSNER. *Principles of Digital Image Synthesis*. Morgan Kaufmann, 1995. 16

[Goral84] CINDY M. GORAL, KENNETH E. TORRANCE, DONALD P. GREENBERG, AND BENNETT BATTAILE. Modeling the interaction of light between diffuse surfaces. *SIGGRAPH Comput. Graph.*, 18:213–222, January 1984. 20, 41

[Guennebaud06] G. GUENNEBAUD, L. BARTHE, AND M. PAULIN. Real-time Soft Shadow Mapping by Backprojection. In *Proc. of EGSR*, pages 227–234, 2006. 47, 73, 82, 85

[Guennebaud07] GAEL GUENNEBAUD, LOIC BARTHE, AND MATHIAS PAULIN. High-quality adaptive soft shadow mapping. *Computer Graphics Forum*, 26(3), 2007. 47, 73, 120

[Hanrahan91] P. HANRAHAN, D. SALZMAN, AND L. AUPPERLE. A Rapid Hierarchical Radiosity Algorithm. *Proc. of ACM SIGGRAPH*, 25(4):197–206, 1991. 4, 41, 142

[Harris01] MARK J. HARRIS AND ANSELMO LASTRA. Real-Time Cloud Rendering. *Computer Graphics Forum*, 20(3):76–84, 2001. 50

[Hasenfratz03a] J.-M. HASENFRATZ, M. LAPIERRE, N. HOLZSCHUCH, AND F. SILLION. A Survey of Real-Time Soft Shadows Algorithms. *Computer Graphics Forum*, 22(4):753–774, dec 2003. 22, 94

[Hasenfratz03b] JEAN-MARC HASENFRATZ, MARC LAPIERRE, NICOLAS HOLZSCHUCH, AND FRANÇOIS SILLION. A survey of Real-Time Soft Shadows Algorithms. *Computer Graphics Forum*, 22(4):753–774, 2003. 23, 25, 44

[Hašan07] MILOŠ HAŠAN, FABIO PELLACINI, AND KAVITA BALA. Matrix Row-Column Sampling for the Many-Light Problem. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 26(3):26, 2007. 49

[Hensley05] JUSTIN HENSLEY, THORSTEN SCHEUERMANN, MONTEK SINGH, AND ANSELMO LASTRA. Interactive Summed-Area Table Generation for Glossy Environmental Reflections. In *ACM SIGGRAPH 2005 Sketches*, page 34, 2005. 83

[Hoberock07] JARED HOBEROCK AND YUNTAO JIA. High-quality Ambient Occlusion . In *GPU Gems 3*, chapter 12, pages 239–274. Addison Wesley, 2007. 42

[Hu07] WEI HU AND KAIHUAI QIN. Interactive Approximate Rendering of Reflections, Refractions, and Caustics. *IEEE Transactions on Visualization and Computer Graphics*, 13(1):46–57, 2007. 50, 132

[Hu10] WEI HU, ZHAO DONG, IVO IHRKE, THORSTEN GROSCH, GUODONG YUAN, AND HANS-PETER SEIDEL. Interactive Volume Caustics in Single-Scattering Media. In *Proc. of ACM I3D*, pages 109–117. ACM, 2010. 4

[Ihrke07] IVO IHRKE, GERNOT ZIEGLER, ART TEVS, CHRISTIAN THEOBALT, MARCUS MAGNOR, AND HANS-PETER SEIDEL. Eikonal Rendering: Efficient Light Transport in Refractive Objects. *ACM Trans. Graph.*, 26(3):article 59, 2007. 51, 130

[Immel86] D. IMMEL, M. COHEN, AND D. GREENBERG. A Radiosity Method for Non-Diffuse Environments. In *Proc. of ACM SIGGRAPH*, pages 133–142, 1986. 17, 20, 56, 57, 58

[Ingle88] JAMES D. INGLE AND STANLEY R. CROUCH. *Spectrochemical Analysis* . Prentice Hall, 1988. 29

[Iwasaki02] K. IWASAKI, Y. DOBASHI, AND T. NISHITA. Efficient Rendering of Optical Effects within Water using Graphics Hardware. *Computer Graphics Forum*, 21(4):701–711, 2002. 50

[Iwasaki07] K. IWASAKI, Y. DOBASHI, F. YOSHIMOTO, AND T. NISHITA. Precomputed Radiance Transfer for Dynamic Scenes Taking into Account Light Interreflection. In *Proc. of EGSR*, pages 35–44, June 2007. 42

[Jansen10] JON JANSEN AND LOUIS BAVOIL. Fourier opacity mapping. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, pages 165–172, 2010. 52

[Jarosz08] WOJCIECH JAROSZ, MATTHIAS ZWICKER, AND HENRIK WANN JENSEN. The Beam Radiance Estimate for Volumetric Photon Mapping. In *Eurographics*, pages 557–566, 2008. 50

[Jensen96] H. W. JENSEN. Global Illumination using Photon Maps. In *Proc. EG Rendering Workshop*, pages 21–30, June 1996. 32, 40, 55

[Jensen98] HENRIK WANN JENSEN AND PER H. CHRISTENSEN. Efficient Simulation of Light Transport in Scences with Participating Media using Photon Maps. In *Proc. of SIGGRAPH*, pages 311–320. ACM, 1998. 3, 32, 50, 51, 125, 143

[Jensen01] HENRIK WANN JENSEN. *Realistic Image Synthesis Using Photon Mapping*. AK Peters, 2001. 49, 50, 130

[Kajiya84] JAMES T. KAJIYA AND BRIAN P VON HERZEN. Ray Tracing Volume Densities. *Proc. of SIGGRAPH*, 18(3):165–174, 1984. 50

[Kajiya86] J. KAJIYA. The Rendering Equation. In *Proc. of ACM SIGGRAPH*, pages 143–150, August 1986. 17, 56

[Kautz04] J. KAUTZ, J. LEHTINEN, AND T. AILA. Hemispherical Rasterization for Self-Shadowing of Dynamic Objects. In *Proc. of EGSR*, pages 179–184, June 2004. 42

[Kautz05] JAN KAUTZ, JAAKKO LEHTINEN, AND SLOAN PETER-PIKE. Precomputed Radiance Transfer: Theory and Practice. In *SIGGRAPH Course Notes*, 2005. 42

[Keller97] A. KELLER. Instant Radiosity. In *Proc. of ACM SIGGRAPH*, pages 49–56, August 1997. 5, 43, 48, 91, 92

[Khronos08] GROUP KHRONOS. OpenCL specification. http://www.khronos.org/registry/cl/specs/, 2008. 36

[Kontkanen05] J. KONTKANEN AND S. LAINE. Ambient Occlusion Fields. In *Proc. of ACM I3D*, pages 41–48, 2005. 43

[Kontkanen06] JANNE KONTKANEN AND TIMO AILA. Ambient Occlusion for Animated Characters. In *Proc. of EGSR*, June 2006. 43

[Kristensen05a] ANDERS WANG KRISTENSEN, TOMAS AKENINE-MÖLLER, AND HENRIK WANN JENSEN. Precomputed local radiance transfer for real-time lighting design. *ACM Trans. Graph.*, 24:1208–1215, 2005. 42

[Kristensen05b] ANDERS WANG KRISTENSEN, TOMAS AKENINE-MÖLLER, AND HENRIK WANN JENSEN. Precomputed Local Radiance Transfer for Real-time Lighting Design. In *ACM Trans. Graph. (Proc. of ACM SIGGRAPH)*, pages 1208–1215, 2005. 49

[Krüger06] JENS KRÜGER, KAI BÜRGER, AND RÜDIGER WESTERMANN. Interactive Screen-Space Accurate Photon Tracing on GPUs. In *Proc. of EGSR*, pages 319–329, June 2006. 51, 126, 134

[Lafortune93] E. LAFORTUNE AND Y. WILLEMS. Bidirectional Path Tracing. In *Proc. of Compugraphics*, pages 95–104, 1993. 40, 55

[Lafortune97] E. LAFORTUNE, S.-C. FOO, K. TORRANCE, AND D. GREENBERG. Non-Linear Approximation of Reflectance Functions. In *Proc. of ACM SIGGRAPH*, pages 117–126, 1997. 15

[Laine05a] SAMULI LAINE AND TIMO AILA. Hierarchical Penumbra Casting. *Computer Graphics Forum*, 24(3):313–322, 2005. 47

[Laine05b] SAMULI LAINE, TIMO AILA, ULF ASSARSSON, JAAKKO LEHTINEN, AND TOMAS AKENINE-MÖLLER. Soft Shadow Volumes for Ray Tracing. *ACM Trans. Graph.*, 24(3):1156–1165, 2005. 47

[Laine07] SAMULI LAINE, HANNU SARANSAARI, JANNE KONTKANEN, JAAKKO LEHTINEN, AND TIMO AILA. Incremental Instant Radiosity for Real-Time Indirect Illumination. In *Proc. of EGSR*, pages 277–286, 2007. 92, 101

[Langer00] M.S. LANGER AND H.H. BLTHOFF. Depth discrimination from shading under diffuse lighting. *Perception*, 29:649–660, 2000. 21

[Lauritzen07] ANDREW LAURITZEN. Summed-Area Variance Shadow Maps. In Hubert Nguyen, editor, *GPU Gems 3*. Addison-Wesley Professional, 2007. 48, 74, 77, 107, 108, 117, 119, 120, 121, 142

[Lauritzen08] ANDREW LAURITZEN AND MICHAEL MCCOOL. Layered variance shadow maps. In *Proc. of GI*, pages 139–146, 2008. 46

[Lehtinen06] JAAKKO LEHTINEN, SAMULI LAINE, AND TIMO AILA. An Improved Physically-Based Soft Shadow Volume Algorithm. *Computer Graphics Forum*, 25(3):303–312, 2006. 47

[Lensch05] HENDRIK P. A. LENSCH, MICHAEL GOESELE, YUNG-YU CHUANG, TIM HAWKINS, STEVE MARSCHNER, WOJCIECH MATUSIK, AND GERO MUELLER. Realistic materials in computer graphics. In *ACM SIGGRAPH 2005 Courses*, SIGGRAPH '05, 2005. 15

[Lischinski92] DANI LISCHINSKI, FILIPPO TAMPIERI, AND DONALD P. GREENBERG. Discontinuity Meshing for Accurate Radiosity. *IEEE Comput. Graph. Appl.*, 12:25–39, November 1992. 20

[Liu04] X. LIU, P.-P. SLOAN, H.-Y. SHUM, AND J. SNYDER. All-Frequency Precomputed Radiance Transfer for Glossy Objects. In *Proc. of EGSR*, pages 337–344, 2004. 41

[Liu07] X. LIU, M.-H. PAN, R. WANG, AND H.-J. BAO. Efficient Rendering of Interreflections for Dynamic Scenes. In *Proc. of Eurographics 2007*, 2007. 42

[Mallo05] O. MALLO, R. PEIKERT, C. SIGG, AND F. SADLO. Illuminated Lines Revisited. In *Proc. of IEEE Visualization*, pages 19–26, 2005. 51

[Martin04] TOBIAS MARTIN AND TIOW SENG TAN. Anti-aliasing and Continuity with Trapezoidal Shadow Maps. In *Rendering Techniques 2004 (Proc. of EGSR)*, pages 153–160, 2004. 24

[Meyer09] QUIRIN MEYER, CHRISTIAN EISENACHER, MARC STAMMINGER, AND CARSTEN DACHSBACHER. Data-Parallel Hierarchical Link Creation for Radiosity. In *Prof. of EGPGV09*, pages 65–69, 2009. 52

[Mic00] Microsoft Corporation. *DirectX 8.0 SDK*, November 2000. Available from http://www.microsoft.com/directx. 34

[Miller94] G. MILLER. Efficient Algorithms for Local and Global Accessibility Shading. In *Proc. of ACM SIGGRAPH*, pages 319–326, July 1994. 21

[Mittring07] MARTIN MITTRING. Finding next gen: Cryengine 2. In *SIGGRAPH Course Notes*, 2007. 43

[Myszkowski01] KAROL MYSZKOWSKI, TAKEHIRO TAWARA, HIROYUKI AKAMINE, AND HANS-PETER SEIDEL. Perception-guided global illumination solution for animation rendering. In *Proc. of ACM SIGGRAPH*, pages 221–230, 2001. 2, 22

[Ng03] R. NG, R. RAMAMOORTHI, AND P. HANRAHAN. All-Frequency Shadows Using Non-linear Wavelet Lighting Approximation. *ACM Trans. Graph.*, 22(3):376–381, July 2003. 20, 41

[Nielsen02] KASPER HØY NIELSEN AND NIELS JØRGEN CHRISTENSEN. Fast texture-based form factor calculations for radiosity using graphics hardware. *J. Graph. Tools*, 6:1–12, 2002. 41

[Nijasure05] M. NIJASURE, S. PATTANAIK, AND V. GOEL. Real-Time Global Illumination on GPUs. *Journal of Graphics Tools*, 10(2):55–71, 2005. 43

[Nishita94] TOMOYUKI NISHITA AND EIHACHIRO NAKAMAE. Method of Displaying Optical Effects within Water using Accumulation Buffer. In *Proc. of SIGGRAPH*, volume 28, pages 373–379, 1994. 51

[NVI08] NVIDIA Corporation. *NVIDIA CUDA specification*, 2008. Available from http://www.nvidia.com/CUDA/. 36

[NVIDIA05] NVIDIA. *Hardware Shadow Mapping*, 2005. http://http.download.nvidia.com/developer/SDK/. 24

[NVIDIA08] NVIDIA. *Cascaded shadow maps*, 2008. http://developer.download.nvidia.com/SDK/. 24

[Oliveira07] MANUEL M. OLIVEIRA AND MAICON BRAUWERS. Real-time Refraction through Deformable Objects. In *Proc. of ACM I3D*, pages 89–96, 2007. 49

[Ostromoukhov04] VICTOR OSTROMOUKHOV, CHARLES DONOHUE, AND PIERRE-MARC JODOIN. Fast Hierarchical Importance Sampling with Blue Noise Properties. *ACM Trans. Graph.*, 23(3):488–495, 2004. 44, 85

[Pantaleoni10] JACOPO PANTALEONI, LUCA FASCIONE, MARTIN HILL, AND TIMO AILA. PantaRay: fast ray-traced occlusion caching of massive scenes. *ACM Trans. Graph.*, 29:37:1–37:10, 2010. 143

[Papadopoulos09] CHARILAOS PAPADOPOULOS AND GEORGIOS PAPAIOANNOU. Realistic Real-time Underwater Caustics and Godrays. In *Proc. of GraphiCon '09*, pages 89–95, 2009. 51, 134

[Parker10] 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 Trans. Graph.*, 29:66:1–66:13, 2010. 40

[Pharr04] MATT PHARR AND SIMON GREEN. Ambient Occlusion . In *GPU Gems*, chapter 17, pages 279–292. Addison Wesley, 2004. 42

[Policarpo05] FABIO POLICARPO AND FRANCISCO FONSECA. Deferred Shading Tutorial. In *SBGAMES*, 2005. 37

[Puech90] C. PUECH, F. SILLION, AND C. VEDEL. Improving Interaction with Radiosity-based Lighting Simulation Programs. In *Proc. of ACM I3D*, pages 51–57, March 1990. 41

[Purcell02] TIMOTHY J. PURCELL, IAN BUCK, WILLIAM R. MARK, AND PAT HANRAHAN. Ray Tracing on Programmable Graphics Hardware. *ACM Trans. Graph.*, 21(3):703–712, 2002. 40

[Purcell03] T. PURCELL, C. DONNER, M. CAMMARANO, H. JENSEN, AND P. HANRAHAN. Photon Mapping on Programmable Graphics Hardware. In *Proc. of Graphics Hardware*, pages 41–50, 2003. 50

[Ramasubramanian99] MAHESH RAMASUBRAMANIAN, SUMANTA N. PATTANAIK, AND DONALD P. GREENBERG. A Perceptually Based Physical Error Metric for Realistic Image Synthesis. In *Proc. of ACM SIGGRAPH*, pages 73–82, 1999. 2, 22

[Reeves87] W. REEVES, D. SALESIN, AND R. COOK. Rendering Antialiased Shadows with Depth Maps. In *Proc. of ACM SIGGRAPH*, pages 283–291, July 1987. 24, 25, 44

[Reinbothe09] CHRISTOPH REINBOTHE, TAMY BOUBEKEUR, AND MARC ALEXA. Hybrid Ambient Occlusion. *EUROGRAPHICS 2009 Areas Papers*, 2009. 43

[Ren06] 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 Trans. Graph.*, 25(3):977–986, 2006. 42, 73

[Ren08] ZHONG REN, KUN ZHOU, STEPHEN LIN, AND BAINING GUO. Gradient-based Interpolation and Sampling for Real-time Rendering of Inhomogeneous, Single-scattering Media. *Computer Graphics Forum*, 27(7):1945–1953, 2008. 50

[Ritschel08a] TOBIAS RITSCHEL, THORSTEN GROSCH, JAN KAUTZ, AND HANS-PETER SEIDEL. Interactive Global Illumination Based on Coherent Surface Shadow Maps. In *Proc. of Graphics Interface*, pages 185–192, 2008. 49

[Ritschel08b] 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 Trans. Graph. (Proc. of SIGGRAPH ASIA 2008)*, 27(5), 2008. 48, 92, 94, 102, 103

[Ritschel09a] TOBIAS RITSCHEL, THOMAS ENGELHARDT, THORSTEN GROSCH, HANS-PETER SEIDEL, JAN KAUTZ, AND CARSTEN DACHSBACHER. Micro-Rendering for Scalable, Parallel Final Gathering. *ACM Trans. Graph. (Proc. of SIGGRAPH Asia 2009)*, 28(5), 2009. 49

[Ritschel09b] TOBIAS RITSCHEL, THORSTEN GROSCH, AND HANS-PETER SEIDEL. Approximating dynamic global illumination in image space. In *Proc. of ACM I3D*, pages 75–82, 2009. 43, 102

[Roger07] DAVID ROGER, ULF ASSARSSON, AND NICOLAS HOLZSCHUCH. Whitted Ray-Tracing for Dynamic Scenes using a Ray-Space Hierarchy on the GPU. In *Rendering Techniques 2007 (Proc. of EGSR)*, pages 99–110, 2007. 40

[Roth82] SCOTT D. ROTH. Ray Casting for Modeling Solids. *Journal of Computer Graphics and Image Processing*, 18(2):109–144, February 1982. 20

[Salvi08] MARCO SALVI. Rendering filtered shadows with exponential shadow maps. In *ShaderX 6.0 - Advanced Rendering Techniques*. Charles River Media, 2008. 44, 46, 108

[Scheuermann07] THORSTEN SCHEUERMANN AND JUSTIN HENSLEY. Efficient histogram generation using scattering on GPUs. In *Proc. of ACM I3D*, pages 33–37, 2007. 99

[Schwarz07] MICHAEL SCHWARZ AND MARC STAMMINGER. Bitmask soft shadows. *Computer Graphics Forum*, 26(3):515–524, 2007. 47, 73, 85

[Segal99] M. SEGAL AND K. AKELEY. *The OpenGL Graphics System: A Specification (Version 1.2.1)*, 1999. 34

[Segovia06] BENJAMIN SEGOVIA, JEAN-CLAUDE IEHL, RICHARD MITANCHEY, AND BERNARD PÉROCHE. Non-interleaved Deferred Shading of Interleaved Sample Patterns. In *Proc. of Graphics Hardware*, pages 53–60, 2006. 101

[Seiler08] LARRY SEILER, DOUG CARMEAN, ERIC SPRANGLE, TOM FORSYTH, MICHAEL ABRASH, PRADEEP DUBEY, STEPHEN JUNKINS, ADAM LAKE, JEREMY SUGERMAN, ROBERT CAVIN, ROGER ESPASA, ED GROCHOWSKI, TONI JUAN, AND PAT HANRAHAN. Larrabee: a many-core x86 architecture for visual computing. *ACM Trans. Graph.*, 27:18:1–18:15, 2008. 143

[Shah07] MUSAWIR A. SHAH, JAAKKO KONTTINEN, AND SUMANTA PATTANAIK. Caustics Mapping: An Image-Space Technique for Real-Time Caustics. *IEEE Transactions on Visualization and Computer Graphics*, 13(2):272–280, 2007. 50

[Sloan02] PETER-PIKE SLOAN, JAN KAUTZ, AND JOHN SNYDER. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. *ACM Trans. Graph.*, 21:527–536, 2002. 1, 20, 41, 73

[Sloan07] PETER-PIKE SLOAN, NAGA K. GOVINDARAJU, DEREK NOWROUZEZAHRAI, AND JOHN SNYDER. Image-Based Proxy Accumulation for Real-Time Soft Global Illumination. In *Proc. of Pacific Graphics*, pages 97–105, 2007. 42

[Smits94] B. SMITS, J. ARVO, AND D. GREENBERG. A Clustering Algorithm for Radiosity in Complex Environments. In *Proc. of SIGGRAPH*, pages 435–442, 1994. 60

[Soler98] C. SOLER AND F. SILLION. Fast Calculation of Soft Shadow Textures Using Convolution. In *Proc. of ACM SIGGRAPH*, pages 321–332, July 1998. 48, 74, 76, 82

[Stewart94] A. JAMES STEWART AND SHERIF GHALI. Fast Computation of Shadow Boundaries Using Spatial Coherence and Backprojections. In *Proc. of SIGGRAPH '94*, pages 231–238, 1994. 47

[Sun05] BO SUN, RAVI RAMAMOORTHI, SRINIVASA G. NARASIMHAN, AND SHREE K. NAYAR. A Practical Analytic Single Scattering Model for Real-time Rendering. *ACM Trans. Graph.*, 24(3):1040–1049, 2005. 50, 126

[Sun06] W. SUN AND A. MUKHERJEE. Generalized Wavelet Product Integral for Rendering Dynamic Glossy Objects. *ACM Trans. Graph.*, 25(3):955–966, 2006. 42

[Sun07] XIN SUN, KUN ZHOU, YANYUN CHEN, STEPHEN LIN, JIAOYING SHI, AND BAINING GUO. Interactive relighting with dynamic BRDFs. *ACM Trans. Graph.*, 26(3), 2007. 42

[Sun08] XIN SUN, KUN ZHOU, ERIC STOLLNITZ, JIAOYING SHI, AND BAINING GUO. Interactive Relighting of Dynamic Refractive Objects. *ACM Trans. Graph.*, 27(3):article 35, 2008. 51, 126, 130

[Sun10] XIN SUN, KUN ZHOU, STEPHEN LIN, AND BAINING GUO. Line space gathering for single scattering in large scenes. *ACM Trans. Graph.*, 29:54:1–54:8, 2010. 52, 143

[Szirmay-Kalos05] LÁSZLÓ SZIRMAY-KALOS, BARNABÁS ASZÓDI, ISTVÁN LAZÁNYI, AND MÁTYÁS PREMECZ. Approximate Ray-Tracing on the GPU with Distance Impostors. *Computer Graphics Forum*, 24(3):695–704, 2005. 50

[Tanaka97] TOSHIMITSU TANAKA AND TOKIICHIRO TAKAHASHI. Fast Analytic Shading and Shadowing for Area Light Sources. *Comput. Graph. Forum*, 16(3):231–240, 1997. 47

[Umenhoffer07] T. UMENHOFFER, G. PATOW, AND L. SZIRMAY-KALOS. *GPU Gems 3*, chapter Robust Multiple Specular

Reflections and Refractions, pages 387–407. Addison-Wesley, 2007. 49

[Veach97] E. VEACH AND L. GUIBAS. Metropolis Light Transport. In *Proc. of ACM SIGGRAPH*, pages 65–76, August 1997. 55

[Veach98] ERIC VEACH. *Robust Monte Carlo Methods for Light Transport Simulation*. PhD thesis, Stanford University, Department of Computer Science, 1998. 128

[Wald02] I. WALD, T. KOLLIG, C. BENTHIN, A. KELLER, AND P. SLUSALLEK. Interactive Global Illumination. In *Proc. of EG Workshop on Rendering*, pages 9–20, 2002. 40

[Wald03] I. WALD, C. BENTHIN, AND P. SLUSALLEK. Interactive Global Illumination in Complex and Highly Occluded Environments. In *Proc. of EGSR*, pages 74–81, 2003. 40, 93

[Wald07] I. WALD, S. BOULOS, AND P. SHIRLEY. Ray Tracing Deformable Scenes using Dynamic Bounding Volume Hierarchies. *ACM Trans. Graph.*, 26(1), 2007. 61

[Wald09] INGO WALD, WILLIAM R MARK, JOHANNES GÜNTHER, SOLOMON BOULOS, THIAGO IZE, WARREN HUNT, STEVEN G PARKER, AND PETER SHIRLEY. State of the Art in Ray Tracing Animated Scenes. *Computer Graphics Forum*, 28(6):1691–1722, 2009. 40

[Wallner09] GÜNTER WALLNER. An extended GPU radiosity solver. *The Visual Computer*, 25(5-7):529–537, 2009. 41

[Walter05] BRUCE WALTER, SEBASTIAN FERNANDEZ, ADAM ARBREE, KAVITA BALA, MICHAEL DONIKIAN, AND DONALD P. GREENBERG. Lightcuts: A Scalable Approach to Illumination. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 24(3):1098–1107, 2005. 49

[Wanger92] LEONARD WANGER. The effect of shadow quality on the perception of spatial relationships in computer generated imagery. In *Proc. of ACM SI3D*, pages 39–42, 1992. 22

[Whitted80] T. WHITTED. An improved illumination model for shaded display. *Communications of ACM*, 23(6):343–349, 1980. 39

[Williams78] L. WILLIAMS. Casting Curved Shadows on Curved Surfaces. In *Proc. of ACM SIGGRAPH*, pages 270–274, August 1978. 22, 44, 74, 92, 107

[Wimmer04] MICHAEL WIMMER, DANIEL SCHERZER, AND WERNER PURGATHOFER. Light Space Perspective Shadow Maps. In *Rendering Techniques 2004 (Proceedings Eurographics Symposium on Rendering)*, pages 143–151, 2004. 24

[Wong05] KEEN-HON WONG, XIN OUYANG, CHI-WAN LIM, TIOW-SENG TAN, AND JÜRG NIEVERGELT. Rendering Anti-Aliased Line Segments. In *Proc. of CGI*, pages 198–205, 2005. 51

[Woo90] ANDREW WOO, PIERRE POULIN, AND ALAIN FOURNIER. A Survey of Shadow Algorithms. *IEEE Computer Graphics & Applications*, 10(6):13–32, 1990. 23, 44

[Wyman05] CHRIS WYMAN. An Approximate Image-Space Approach for Interactive Refraction. *ACM Trans. Graph.*, 24(3):1050–1053, 2005. 49

[Wyman06] CHRIS WYMAN AND SCOTT DAVIS. Interactive Image-Space Techniques for Approximating Caustics. In *Proc. of ACM I3D*, pages 153–160, 2006. 50

[Wyman08a] C. WYMAN AND S. RAMSEY. Interactive Volumetric Shadows in Participating Media with Single-Scattering. In *Proc. of IEEE Symposium on Interactive Ray Tracing*, pages 87–92, 2008. 50, 126

[Wyman08b] CHRIS WYMAN. Hierarchical Caustic Maps. In *Proc. of ACM I3D*, pages 163–171, 2008. 50, 128, 134, 137, 140

[Wyman09] CHRIS WYMAN AND GREG NICHOLS. Adaptive Caustic Maps Using Deferred Shading. *Computer Graphics Forum*, 28(2):309–318, 2009. 50, 134

[Yang09] BAOGUANG YANG, JIEQING FENG, GAEL GUENNEBAUD, AND XINGUO LIU. Packet-based Hierarchal Soft Shadow Mapping. *Computer Graphics Forum*, 2009. 47

[Yang10] BAOGUANG YANG, ZHAO DONG, JIEQING FENG, HANS-PETER SEIDEL, AND JAN KAUTZ. Variance Soft Shadow Mapping. *Computer Graphics Forum (Proc. of Pacific Graphics 2010)*, 29(7):2127–2134, 2010. 4

[Yu05] JINGYI YU, JASON YANG, AND LEONARD MCMILLAN. Real-time Reflection Mapping with Parallax. In *Proc. of ACM I3D*, pages 133–138, 2005. 49

[Yu07] XUAN YU, FENG LI, AND JINGYI YU. Image-Space Caustics and Curvatures. In *Proc. of Pacific Graphics*, pages 181–188, 2007. 50

[Yu09] INSU YU, ANDREW COX, MIN H. KIM, TOBIAS RITSCHEL, THORSTEN GROSCH, CARSTEN DACHSBACHER, AND JAN KAUTZ. Perceptual influence of approximate visibility in indirect illumination. *ACM Trans. Appl. Percept.*, 6(4):1–14, 2009. 2, 5

[Zhang06] FAN ZHANG, HANQIU SUN, LEILEI XU, AND LEE KIT LUN. Parallel-split shadow maps for large-scale virtual environments. In *Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications*, pages 311–318, 2006. 24

[Zhou05] K. ZHOU, Y. HU, S. LIN, B. GUO, AND H.-Y. SHUM. Precomputed Shadow Fields for Dynamic Scenes. *ACM Trans. Graph.*, 24(3):1196–1201, 2005. 42

[Zhou08a] KUN ZHOU, QIMING HOU, RUI WANG, AND BAINING GUO. Real-time KD-tree construction on graphics hardware. *ACM Trans. Graph.*, 27:126:1–126:11, 2008. 40

[Zhou08b] KUN ZHOU, ZHONG REN, STEPHEN LIN, HUJUN BAO, BAINING GUO, AND HEUNG-YEUNG SHUM. Real-time Smoke Rendering using Compensated Ray Marching. *ACM Trans. Graph.*, 27(3):article 36, 2008. 50

[Zhukov98] S. ZHUKOV, A. IONES, AND G. KRONIN. An Ambient Light Illumination Model. In *Proc. of EG Rendering Workshop*, pages 45–56, 1998. 1, 2, 21, 42

[Zöckler96] M. ZÖCKLER, D. STALLING, AND H.-C. HEGE. Interactive Visualization of 3D-Vector Fields Using Illuminated Stream Lines. In *Proc. of IEEE Visualization*, pages 107–113, 1996. 51

Appendix A

List of Publications

[A] Baoguang Yang, **Zhao Dong**, Jieqing Feng, Hans-Peter Seidel, Jan Kautz,: *Variance Soft Shadow Mapping*. Computer Graphics Forum (Pacific Graphics 2010) (The first two authors contributed equally.)

[B] Chunxia xiao, Meng Liu, Yongwei Nie, **Zhao Dong**: *Fast Exact Nearest Patch Matching for Patch-based Image Editing and Processing*. To appear in IEEE Transactions on Computer Graphics and Visualization.

[C] Wei Hu, **Zhao Dong**, Ivo Ihrke, Throsten Grosch, Hans-Peter Seidel: *Interactive Volume Caustics in Single-Scattering Media*. In Proc. of ACM I3D 2010, Washington DC, USA.(The first two authors contributed equally.)

[D] **Zhao Dong**, Baoguang Yang: *Variance Soft Shadow Mapping*. In Proc. of ACM I3D 2010, Poster, Washington DC, USA.

[E] **Zhao Dong**, Throsten Grosch, Tobias Ritschel, Jan Kautz, Hans-Peter Seidel: *Real-time Indirect Illumination with Clustered Visibility. In Proc. of Vision, Modeling, and Visualization Workshop 2009 (VMV 2009), Braunschweig, Germany.*

[F] Thomas Annen, **Zhao Dong**, Tom mertens, Philippe Bekaert, Hans-Peter Seidel, Jan Kautz: *Real-Time, All-Frequency Shadows in Dynamic Scenes*. In ACM Transactions on Graphics (SIGGRAPH 2008), los angeles, USA.

[G] Xinguo Liu, **Zhao Dong**, Qunsheng Peng, Hujun Bao: *Caustics Spot Light for Rendering Caustics*. In the Visual Computer (Best ranked paper accepted by CGI 2008).

[H] **Zhao Dong**, Jan Kautz, Chrisitan Theobalt, Hans-Peter Seidel: *Interactive Global Illumination Using Implicit Visibility*. In Proc. of Pacific Graphics 2007 (oral paper), Hawaii, USA.

[I] Mingli Song, **Zhao Dong**, Chrisitan Theobalt, Huiqiong Wang, Zicheng Liu, Hans-Peter Seidel: *A General Framework for Efficient 2D and 3D Facial Expression Analogy*. In IEEE Transactions on Multimedia, Volume: 9, Issue: 7.

[J] **Zhao Dong**, Wei Chen, Hujun Bao, Hongxin Zhang, Qunsheng Peng: *Real-time Voxelization for Complex Polygonal Models. In Proc. of Pacific Graphics 2004 (oral paper), Seoul, Korea.*

[K] **Zhao Dong**, Wei Chen, Long Zhang, Qunsheng Peng: *Balancing CPU and GPU: Real-time Visualization of Large Scale 3D Scene. In Proc. of GCC 2004, VVS workshop (Visualization and Visual Steering), Lecture Notes in Computer Science, Springer Verlag.*

Appendix B

Curriculum Vitae – Lebenslauf

Curriculum Vitae

October, 1980	Born in Loudi, Hunan Province, China
September 1991 - June 1994	Middle School, Liangang Middle School, Loudi, China
September 1994 - June 1997	High School, Liangang High School, Loudi, China
September 1997 - June 2001	B.S. in Computer Science and Polymer Engineering, Zhejiang University, China
September 2002 - April 2005	M.Sc. in Computer Science, Zhejiang University, China
October 2004 - January 2005	Research Intern, Microsoft Research Asia, Beijing, China.
September 2005 -	Ph.D. Student at the Max-Planck-Institut für Informatik, Saarbrücken, Germany
September 2008 - December 2008	Research Intern, Microsoft Research Redmond, Redmond, USA.

Lebenslauf

Oktober, 1980	Geboren in Loudi, Hunan Province, China
September 1991 - Juni 1994	Mittelschule, Liangang Middle School, Loudi, China
September 1994 - Juni 1997	Gymnasium, Liangang High School, Loudi, China
September 1997 - Juni 2001	B.S. in Informatik und Polymer Engineering, Zhejiang University, China
September 2002 - April 2005	M.Sc. in Informatik, Zhejiang University, China
Oktober 2004 - Januar 2005	Forschungspraktikum, Microsoft Research Asia, Beijing, China.
September 2005 -	Promotion am Max-Planck-Institut für Informatik, Saarbrücken, Deutschland
September 2008 - Dezember 2008	Forschungspraktikum, Microsoft Research Redmond, Redmond, USA.