FAKULTÄT FÜR !NFORMATIK

Real-Time Global Illumination Using Temporal Coherence

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Computergraphik/Digitale Bildverarbeitung

eingereicht von

Martin Knecht
Matrikelnummer 0326294

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung:
Betreuer/Betreuerin: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Mitwirkung: Dipl.-Ing. Oliver Mattausch

Wien, 28.07.2009

(Unterschrift Verfasser/in) (Unterschrift Betreuer/in)

Erklärung zur Verfassung der Arbeit

Martin Knecht
Mottaweg 72, 6822 Röns

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

Ort, Datum:_____ Unterschrift: _____

Abstract

The goal of this thesis is to produce plausible global illumination in real time using temporal coherence. While direct illumination combined with precomputed static global illumination is widely used in today's computer games and 3D applications, real-time global illumination that supports arbitrary dynamic scenes and light setups is still an active area of research.

This master thesis gives an introduction to global illumination and discusses various methods that have been developed. However, since most of the existing methods need some kind of precomputation to calculate global illumination in real time, they also introduce limitations like static light, scenes or view points. Furthermore other algorithms are not suitable for the capabilities of current graphics hardware or are simply fake approaches.

The core of this thesis is an improved version of the instant radiosity and imperfect shadow maps algorithm that reuses illumination information from previous frames. The previous approaches needed a high number of so called virtual point lights to get convincing results, whereas our method achieves visually pleasing results with only a few virtual point lights. As a second extension to the base algorithms we introduce a new method to compute multiple light bounces. In this method the fill rate is drastically reduced and therefore computation time is much lower than in previous aproaches.

Kurzfassung

Das Ziel dieser Diplomarbeit ist es, glaubwürdige globale Beleuchtung, mit Hilfe von temporaler Kohärenz, in Echtzeit zu berechnen. Direkte Beleuchtung verbunden mit vorausberechneter globaler Beleuchtung wird bereits häufig in Computerspielen oder anderen 3D Programmen eingesetzt. Die Echtzeitberechnung von globaler Beleuchtung für dynamische Szenen und dynamischem Licht stellt dabei aber immer noch eine große Herausforderung dar und ist deshalb ein aktives Forschungsfeld.

Diese Diplomarbeit gibt eine Einführung in die globale Beleuchtung und beschreibt einige Methoden die bereits dafür entwickelt wurden. Da aber die meisten Methoden Vorberechnungen benötigen, unterliegen sie bestimmten Einschränkungen wie statischen Szenen oder statischer Licht- beziehungsweise Kameraposition. Des weiteren, sind manche Algorithmen nicht auf heutige Graphikhardware angepasst oder globale Beleuchtung wird nur vorgetäuscht.

Der Kern dieser Diplomarbeit ist eine verbesserte Version der Instant Radiosity und Imperfect Shadow Maps Algorithmen, welche die Beleuchtungsinformationen aus dem letzten Bild wiederverwendet. Die vorher genannten Methoden benötigen eine hohe Anzahl an sogenannte virtuellen Lichtquellen um ansprechende Resultate zu erzielen. Unsere Methode hingegen, erreicht bereits mit wenigen virtuellen Lichtquellen ansprechende Ergebnisse. In einer zweiten Erweiterung zu den Algorithmen präsentieren wir einen neuen Algorithmus um mehrere Lichtreflexionen auf der Oberfläche zu berechnen. Unsere Methode reduziert die Anzahl der zu zeichnenden Pixel, wodurch die Berechnungszeit im Vergleich zu den vorhergehenden Methoden kleiner ist.

Acknowledgements

Many thanks go to all the people at *The Institute of Computer Graphics* at the *Vienna University of Technology* for their amazing support during my studies and while writing this thesis. Particular I want to thank Michael Wimmer and Oliver Mattausch, who both supervised my work on this master thesis and gave me lots of useful hints and comments.

Many thanks also to the members of the *Computer Graphics Club* for the interesting discussions, the comments and help, the CGC lunches and the fun at the CESCG conferences. :-)

Special thanks to all the university colleagues who helped me on my way through all the exams and laboratory courses. In particular I want to thank Michael Schwärzler and my brother Wolfgang Knecht for the great discussions, collaboration and fun! :-)

Many thanks go to my friends and flatmates, for the great time during my studies, the parties, all the hilarious discussions and the legendary "Shisha hocks"! :-)

I want to thank the *Faculty of Informatics* and its professors at the *Vienna University of Technology* for their financial support, which made it possible to acquire the needed hardware.

Finally, I thank my family for their invaluable support, help and patience. :-)

Contents

1. Introduction 6
- 1.1 Global Illumination 8
  - 1.1.1 The Light Transport Notation 8
  - 1.1.2 The Rendering Equation 9
  - 1.1.3 BRDFs 11
- 1.2 Goals of this Thesis 12

2. State of the Art 14
- 2.1 Introduction 14
- 2.2 Lightmaps 14
- 2.3 Precomputed Radiance Transfer 16
  - 2.3.1 Spherical Harmonics 18
  - 2.3.2 Diffuse BRDFs 20
  - 2.3.3 Glossy BRDFs 21
- 2.4 Ambient Occlusion 22
  - 2.4.1 Dynamic Ambient Occlusion and Indirect Lighting 23
  - 2.4.2 Approximating Dynamic Global Illumination in Image Space 26
- 2.5 Implicit Visibility and Antiradiance for Interactive Global Illumination 29
  - 2.5.1 Reformulation of the Rendering Equation 30
  - 2.5.2 Hierarchical Radiosity 32
- 2.6 Instant Radiosity 33
  - 2.6.1 Introduction 33
  - 2.6.2 Incremental Instant Radiosity 35
  - 2.6.3 Splatting Indirect Illumination 40
  - 2.6.4 Interactive Global Illumination Based on Coherent Surface Shadow Maps 43
  - 2.6.5 Imperfect Shadow Maps 50
- 2.7 Temporal Coherence 52
  - 2.7.1 The Real-Time Reprojection Cache 52
  - 2.7.2 Pixel-Correct Shadow Maps with Temporal Reprojection and Shadow Test Confidence 53

3. Real-Time Global Illumination Using Temporal Coherence . . 56
3.1 Introduction 56
3.2 The Rendering Framework 57
3.3 Generating Virtual Point Lights 58
3.4 Virtual Point Light Shading 60
3.4.1 Interleaved Sampling 60
3.4.2 Tiled Mesh Geometry 61
3.4.3 Glossy Surfaces 61
3.5 Temporal Coherence 63
3.5.1 The Movement Buffer 64
3.5.2 Confidence Computation 65
3.6 Multiple Light Bounces 66
3.6.1 Algorithm outline 67
3.7 Summary 69

4. Implementation 70
4.1 Introduction 70
4.2 Camera G-Buffer 72
4.3 Light G-Buffer 73
4.4 Light G-Buffer Sampling 74
4.4.1 Virtual Point Light Data Structure 75
4.4.2 Spotlight Sampler 75
4.4.3 Pointlight Sampler 75
4.5 Virtual Point Light Shading 76
4.6 Final Composition 78
4.7 Summary 78

5. Results 80
5.1 Introduction 80
5.2 Temporal Coherence 81
5.3 Multiple Light Bounces 87
5.4 Limitations 88
5.5 Summary 89

6. Conclusion 90
6.1 Future Work 92

Chapter 1

Introduction

In the right light, at the right time, everything is extraordinary.

Aaron Rose

The human visual system is the most important sensory device, as the perception of the surrounding world heavily relies on it. With its help we are able to recognize the world in a very astonishing way. We can *see* how objects are positioned to each other, have an impression of depth, color, shape and surface of objects and we can also *see* where the light comes from. All this information is extracted from the incoming light, hitting the retina of the eyes, which is nothing more than a bunch of electromagnetic energy. Beside the very interesting question on how the brain manages to extract these kinds of information about the world, it is also very interesting how the incoming light has to be *set up*, so that the visual system can extract as much information as possible out of it.

Thousands of years ago, cavemen already started to paint at walls of caves and so produced the first images. Although the images had only two-dimensional representations of the objects, they could still be recognized. But only a fraction of visual information was available for the human brain. For example, perspective and depth information was completely missing. In the renaissance epoch painters discovered how to draw perspectively correct images and so they were able to give an impression of the depth in the scenes.

In computer science the process of producing images is called image synthesis. The history of realistic image synthesis is somehow similar. The first produced images were only two dimensional and later on, when the computer had more performance, three-dimensional scenes could be rendered. But in opposite to painted images, computer generated images still did not look realistic. They did not contain correctly shaded surfaces or shadows and therefore, some information for the human visual system was still missing. Further development made

it finally possible to render images that look almost realistic, by using global illumination methods (see Chapter 1.1). Global illumination describes how light propagates in a scene and since light can bounce off from several surfaces, it is a mathematically complex task. However, to calculate absolutely realistic looking images, it still takes time from a couple of hours up to several days.

The image synthesis process described in the previous paragraph is called offline rendering. Offline rendering means that the position of objects, light or the view point cannot be changed interactively, as the rendering time is way too long. With the increase of computation power and the introduction of graphics hardware it is possible to render several images (called frames) per second. This way, we are able to create interactive applications that synthesize images at a rate of 20 to 60 or more Hertz. However, these images are far away from looking realistic. There are several reasons for non-realistic looking images, but a main reason is that the illumination calculation is only a local process instead of a global one. Figure 1.1 shows the difference between a locally and globally illuminated scene.

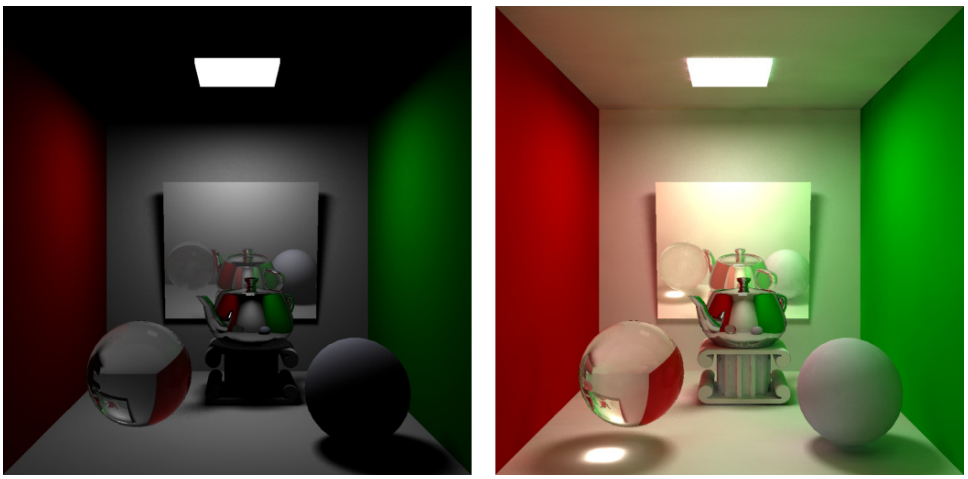


Fig. 1.1: Image on the left is rendered with local illumination. Image on the right is rendered using a global illumination approach. Note the color bleeding and caustics as well as the more natural look of the image. (Image courtesy of Guerrero [Gue07])

In the last few years, researchers started to develop algorithms that are able to perform global illumination in real time. However, most of these methods have some restrictions or limitations. Sometimes only static scenes are allowed or the view or light has to be fixed. In other methods a huge precomputation step is needed to achieve interactive frame rates. In 1997 Alexander Keller introduced Instant Radiosity [Kel97], the method our thesis is based on (see Section 2.6). It achieves global illumination by placing so called virtual point lights (VPL) in a scene. However, visibility calculation for each VPL was a very time consum-

ing task and therefore not usable for real time, until Ritschel et al. introduced the concept of imperfect shadow maps [RGK+08] (see Section 2.6.5). Imperfect shadow maps enable fast visibility calculation and therefore are suitable for real time global illumination. This concept is the main basis of this thesis and we will extend it by exploiting temporal coherence.

1.1 Global Illumination

Global Illumination describes how light propagates in a scene. Light starts from one or several light sources and gets reflected, scattered or transmitted at surface points. It is important to know how light hitting the surface gets reflected, and this behavior is described by the *bidirectional scattering distribution function* (BSDF) (see Section 1.1.3). After several possible surface hits the light finally reaches the eye/view point. Actually, there are an infinite number of possible paths that light can take from the light source to the view point. Thus, illumination computation for a given pixel on the screen is a global, complex and time consuming task that has to be solved as fast as possible for real-time applications.

1.1.1 The Light Transport Notation

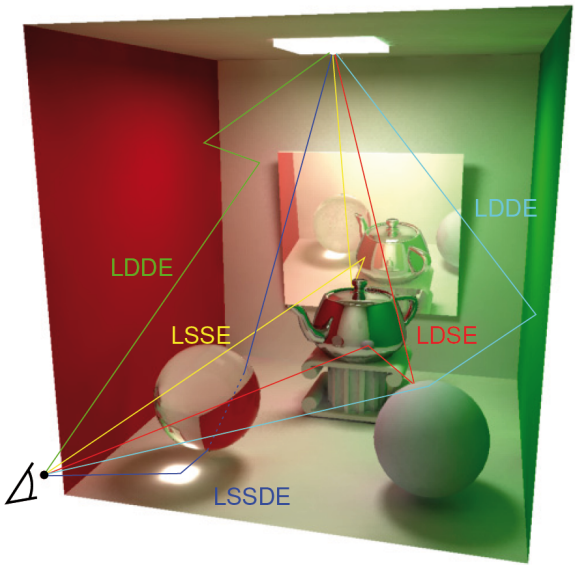


Fig. 1.2: Image rendered using global illumination. Some possible light paths are shown with its corresponding light transport notation. (Image courtesy of Guerrero [Gue07])

As mentioned before, light can take various paths from the light source to the

eye. In 1990 Heckbert [Hec90] therefore introduced the so called *light transport notation*. It allows describing a light path using the type of surfaces (diffuse or specular) that were hit. But it also gives us a tool to describe the capabilities of rendering methods. The notation consists of the elements **L**, **S**, **D** and **E**.

L - is used for a ray starting from a light source

S - describes a specular light bounce

D - describes a diffuse light bounce

E - is used for the eye/view point

Figure 1.2 shows several ray paths starting from the light source and ending at the eye/view point. Furthermore, regular expressions can be used to describe groups of paths. **S*** describes zero or more -, **S+** one or more - and **S?** zero or one specular reflections. **(S|D)** stands for a specular or diffuse reflection.

A global illumination renderer that supports all possible light paths must be able to calculate paths with notation **L(S|D)*E**. As pointed out by Heckbert [Hec90], classic raytracing can only handle **LD?S*E** paths and classic radiosity supports only diffuse bounces and therefore only calculates **LD*E** paths.

1.1.2 The Rendering Equation

The rendering equation was introduced by James Kajiya [Kaj86] in 1986. It completely describes the global illumination process. The original form looked as follows:

$$I(x, x') = g(x, x')[\epsilon(x, x') + \int_{\Omega} \rho(x, x', x'')I(x', x'')dx''] \tag{1.1}$$

where

$I(x, x')$ is the light intensity that passes from point x' to x.

$g(x, x')$ is the geometry term that describes how the layout of the geometry influences the light transfer.

$\epsilon(x, x')$ is the emitted light from x' to x.

$\rho(x, x', x'')$ is the BRDF function. It describes the amount of light that gets transferred from point x'', which illuminates point x', to point x.

Note that in this form of the rendering equation, point locations are used instead of ray directions that are used in the alternative form. In the remainder of this thesis we will use the alternative representation because it is more common today.

$$L_o(p, \omega_o) = L_e(p, \omega_o) + \int_\Omega \rho(p, \omega_i, \omega_o) L_i(p, \omega_i) cos\theta d\omega_i \tag{1.2}$$

where p is the point at which incoming light gets scattered in direction ω_o. $L_e(p, \omega_o)$ is the light emitted from point p in direction ω_o. The integral samples over the upper hemisphere Ω or the whole sphere, but then a BSDF has to be used. The BRDF $\rho(p, \omega_i, \omega_o)$ takes directions instead of point locations and $L_i(p, \omega_i)$ is the light, that arrives point p from direction ω_i. θ represents the angle between the incoming direction ω_i and the normal at point p. Thus $cos\theta$ forms the geometry term in this form of the rendering equation. A geometric explanation of the rendering equation is shown in Figure 1.3

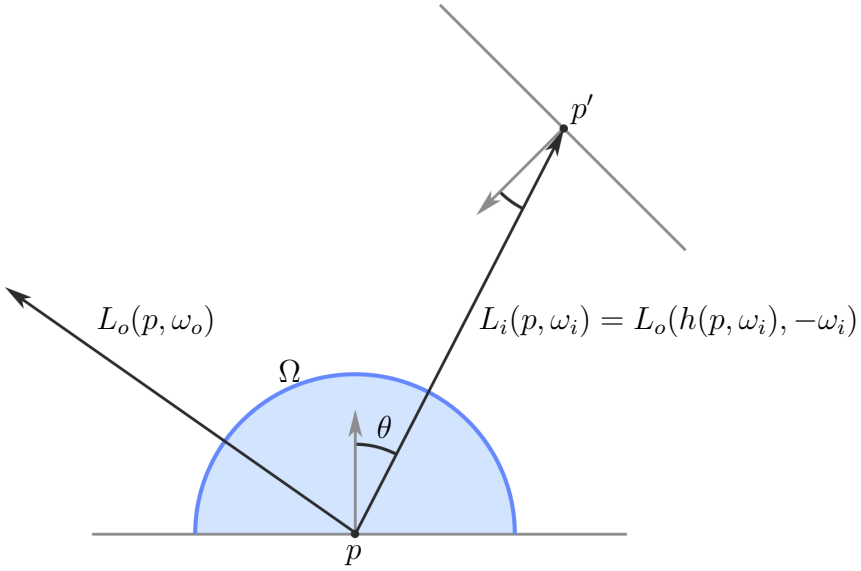


Fig. 1.3: Graphical illustration of the rendering equation.

It is very hard to solve the rendering equation analytically because it is a recursive equation. The incoming light L_i from direction ω_i is the outgoing light from a point p' (p' is the first intersection point when a ray is shot from p in direction ω_i). We can introduce an integral operator T as

$$(TL_o)(p, \omega_o) = \int_\Omega \rho(p, \omega_i, \omega_o) L_o(h(p, \omega_i), -\omega_i) cos\theta d\omega_i \tag{1.3}$$

where $h(p, \omega_i)$ is a function that returns the first ray intersection starting from point p into direction ω_i, the previously mentioned intersection point p', and thus the short form of the rendering equation looks like

$$L_o = L_e + TL_o \tag{1.4}$$

It is easy to see that the rendering equation is recursive and that it can be expanded to

$$L_o = L_e + T(L_e + T(L_e + T(... \tag{1.5}$$

Note that for physically based BRDFs the reflected amount of energy is always smaller or equal to the incoming radiant energy. Thus only the first terms really influence the final result of L_o and are therefore more important.

1.1.3 BRDFs

A *bidirectional reflectance distribution function* (BRDF) describes how incident light is scattered at a surface point over a hemisphere that is oriented with respect to its normal. It was first specified by Nicodemus [Nic70] in 1970. The function returns the reflected amount of light $L_i(\omega_i)$ for a given incoming direction ω_i and an outgoing light $L_o(\omega_o)$ in direction ω_o. The BRDF is a four-dimensional function, with two angular values for incoming and outgoing direction. Furthermore we can add dependence to the surface location. Thus, if the surface is parameterized using a two dimensional function a BRDF is a six-dimensional function.

A BRDF only describes the behavior of a surface on the upper hemisphere with respect to the surface normal. The *bidirectional transmittance distribution function* (BTDF) on the other hand describes transmittance of light through materials. For a BRDF the incoming direction ω_i and the outgoing direction ω_o are in the same hemisphere. For a BTDF they are on opposite hemispheres. With BTDFs translucent materials can be supported in rendering systems. The combination of a BRDF and BTDF is called the *bidirectional scattering distribution function* (BSDF).

BRDFs can be represented through measured data or through analytical models. The problem with measured BRDFs is, that they provide a huge amount of data because of its multidimensionality. Furthermore for off line physically based rendering the wavelength of the light is also taken into account, thus the BRDF becomes a seven-dimensional function.

To omit huge storage costs for BRDFs, several analytical models have been developed [Pho75, CT81, ON94].

BRDFs or their corresponding materials have a couple of properties that will be described here [PH04].

1. *Diffuse, glossy specular, perfect specular and retro-reflective*: Surface materials can be categorized into four main groups: Diffuse, glossy specular, perfect specular and retro-reflective surfaces. Diffuse materials have the characteristic that the outgoing radiance is independent of the outgoing direction ω_o and therefore the BRDF is only a four-dimensional function. Figure 1.4 shows polar plots for all four types of surfaces. Note that most of the incoming light is scattered in direction of the reflection vector for specular surfaces. A perfect mirror or glass has no diffuse part and only reflects incident light in direction of the reflection vector. The moon is an example for a retro-reflective surface, as it reflects most of the incident light back in the same direction.

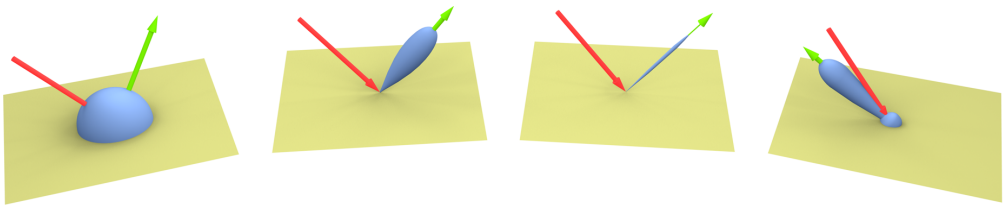


Fig. 1.4: Graphical illustration of the different material types. The red ray illustrates ω_i, while the green ray shows ω_o. From left to right: diffuse, glossy specular, perfect specular and retro-reflective surfaces.

2. *Helmholtz reciprocity*: Physically based BRDFs are reciprocal. That means that the BRDF function returns the same result, if the incoming and outgoing directions are interchanged.

$$\rho(p, \omega_i, \omega_o) = \rho(p, \omega_o, \omega_i)$$

3. *Energy conservation*: Physically based BRDFs are energy conserving. The reflected amount of energy is always less or equal to the incoming amount of energy and thus

$$\int_\Omega \rho(p, \omega_i, \omega_o) cos\theta d\omega_i \leq 1 \tag{1.6}$$

4. *Isotropic or Anisotropic*: Isotropic surfaces do not change the amount of reflected light, if they are rotated around their normals. In opposite, anisotropic materials, like compact discs, do change under different rotations.

1.2 Goals of this Thesis

In this thesis we will first give an overview on real-time global illumination methods that are related to this work in Chapter 2. Note that the instant radiosity

algorithm is also explained in this chapter, along with the imperfect shadow maps method.

In Chapter 3 we will present our new method to achieve global illumination in real-time. We introduce two main extensions to the imperfect shadow maps approach. The first extension reduces time for shading, while maintaining good quality global illumination. The idea is to exploit temporal coherence between consecutive frames and reuse as much information from the previous frame as possible. The second extension introduces a new method to calculate multiple light bounces. The new method reduces fill-rate drastically in contrast to the method proposed by Ritschel et al. [RGK$^+$08].

Chapter 4 will outline details of the implementation and Chapter 5 will present the results of our work. Finally, Chapter 6 gives a conclusion to the thesis.

Chapter 2

State of the Art

In the beginning there was nothing. God said, "Let there be light!" And there was light. There was still nothing, but you could see it a whole lot better.

Ellen DeGeneres

2.1 Introduction

Several methods and algorithms exist to simulate or approximate global illumination in real-time environments. This chapter is a collection of some of these approaches. Section 2.2 describes the lightmaps approach and Section 2.3 will explain the Precomputed Radiance Transfer method [SKS02]. In Section 2.4 we describe a method for approximating global illumination called ambient occlusion. Section 2.5 introduces a method that avoids explicit visibility queries by reformulating the rendering equation. Section 2.6 explains the idea of Instant Radiosity, the technique on which our proposed method is based on. Section 2.7 introduces real-time approaches that exploit temporal coherence to get better results.

2.2 Lightmaps

Lightmaps are a quite old but still useful technique to get global illumination effects in real-time environments. The basic idea is to map so called lightmaps onto the scene geometry as it is done with standard texture maps. With the difference that texture maps store surface color information whereas lightmaps store surface illumination information. A similar method was first introduced by Segal

et al. [SKvW+92] where they used projective texture mapping to render images with a spotlight where the area of illumination by the spotlight was defined with a lightmap.

Lightmaps have values between 0.0 and 1.0 corresponding to the illumination at a given surface point. The resulting color c for a pixel with a texture color tc is calculated with the trivial equation

$$c = tc * l \tag{2.1}$$

where l is the illumination read out from the lightmap. Figure 2.1 shows an example with a simple lightmap. Although the concept of lightmaps is pretty simple, there are some issues which will be addressed in the next paragraphs.

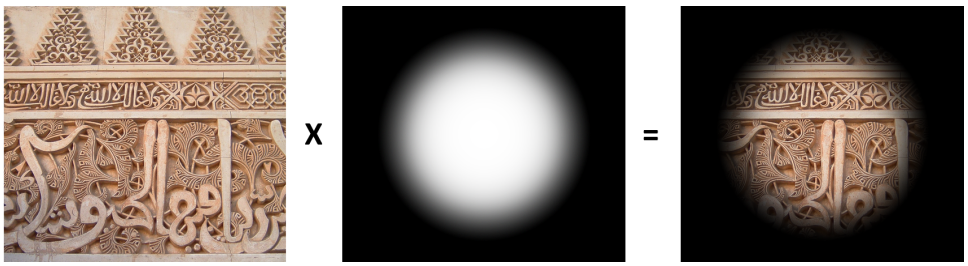


Fig. 2.1: The images illustrate the light map technique. A texture gets multiplied by a light map to get the final illuminated surface.

Lightmap precalculation In a precalculation step all the lightmaps for every polygon in the scene are calculated. This is done by performing a light simulation on the scene and storing the illumination in the lightmaps. Since this can result in a huge amount of data, lightmaps have normally only low resolutions. This saves memory but also reduces the possibility to include high frequency lighting effects such as caustics or sharp shadow edges.

Another problem is the mapping of lightmaps. If the alignment of the lightmaps for two adjacent polygons does not match exactly, visible artifacts will appear at the shared edge because of the very low resolution of the lightmaps. Planar mapping helps to overcome this issue.

Precomputed lightmaps work only with static scenes, however it is still possible to combine lightmaps with dynamic lighting methods. Therefore they can be used to get a coarse global illumination approximation which works properly for all static objects and all dynamic objects can be shaded using standard real-time lighting.

Lightmap baking As there is one lightmap for every polygon it is impractical to load and create thousands of lightmaps into video memory. A more common method is to bake several lightmaps into one big map.

2.3 Precomputed Radiance Transfer

Fig. 2.2: Head rendered using precomputed radiance transfer with self-shadowing and self-interreflection. (Image courtesy of Sloan et al. [SKS02])

Precomputed Radiance Transfer was first introduced by Sloan et al. [SKS02] in 2002. It is an elegant approach which associates a set of transfer functions between incident lighting and outgoing radiance to scene geometry. In other words, the transfer functions describe how an object will look like, for a given incident lighting situation.

They use low-order spherical harmonics to represent the transfer functions and the incident lighting (there also exist similar methods that use for example wavelets as representation [NRH04, NRH03]). Low-order spherical harmonics are an adequate representation method because they avoid aliasing and can be efficiently evaluated on the graphics hardware. With the proposed method the complexity of light interaction/transport can be split into two steps. The calculation of the transfer functions and the calculation of the outgoing radiance at runtime.

In the first precomputation step a global transport simulator creates these transfer functions for a static scene. The simulator is able to bake self-shadowing and self-interreflections into these functions.

The second step performs the mapping from incident lighting to outgoing radiance. It is important to distinguish between objects with diffuse BRDFs and objects with glossy BRDFs. When rendering objects with diffuse BRDFs, the shading integral gets reduced to a simple dot product of 9 to 25 elements between

the incident lighting and the transfer functions. For glossy BRDFs matrices are used instead of vectors. Figure 2.2 shows a head rendered with the proposed method.

Definition Assuming the light source L_{env} to be infinitely distant, the incident light L_i for a given point p can be described as a product of the radiance from the light source and the local transfer function T_p

$$L_i(p, \omega_o) = L_{env}(\omega_i)T_p(\omega_i) \tag{2.2}$$

The local transfer function T_p at point p can be further split into the following form:

$$T_p(\omega_i) = V_p(\omega_i) + R_p(\omega_i) \tag{2.3}$$

where $V_p(\omega_i)$ is the visibility function. The visibility function returns 1 if there is no geometry intersection between p and the infinitely distant light source in direction ω_i and 0 otherwise. Figure 2.3 shows how the visibility function looks like for a given point p. The reflection term $R_p(\omega_i)$ describes how much light from direction ω_i arrives at point p through light reflections on the objects surface. Using this this kind of light transport representation the rendering equation (see Section 1.1.2) can be written as follows:

$$L_o(p, \omega_o) = \int_\Omega \rho(p, \omega_i, \omega_o)L_{env}(\omega_i)T_p(\omega_i)cos\theta \tag{2.4}$$

where $\rho(p, \omega_i, \omega_o)$ is the BRDF and θ the angle between incoming direction ω_i and the normal of point p.

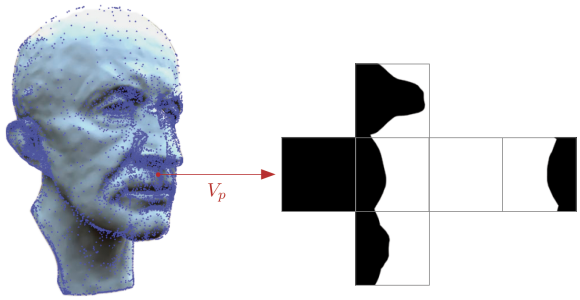


Fig. 2.3: Result of the visibility function V_p for a given point p. (Image courtesy of Guerrero [Gue07])

Equation 2.4 has no emittance term since it is assumed that the only light emitting source is the infinitely distant light L_{env}. For a given point p the next step would be to sample over the whole sphere (although the BRDF will zero out samples that are on the lower hemisphere) and store the results. Doing this over the entire surface of an object results in an huge amount of data. This is the point were Spherical Harmonics come into play because they allow a very compact way to represent these transfer functions.

2.3.1 Spherical Harmonics

Spherical Harmonics (SH) are a set of functions defined over a sphere **S**. They are very similar to the Fourier transformation. The basis functions of Spherical Harmonics are ordered into bands, indexed by parameter l, which group the functions by their frequency. Figure 2.4 shows the basis functions for the first four bands. Each band has two more basis functions in the group than the former one. The basis functions in one band are indexed through m. As we will see later there is also a representation where only one index variable is necessary to access all the basis functions.

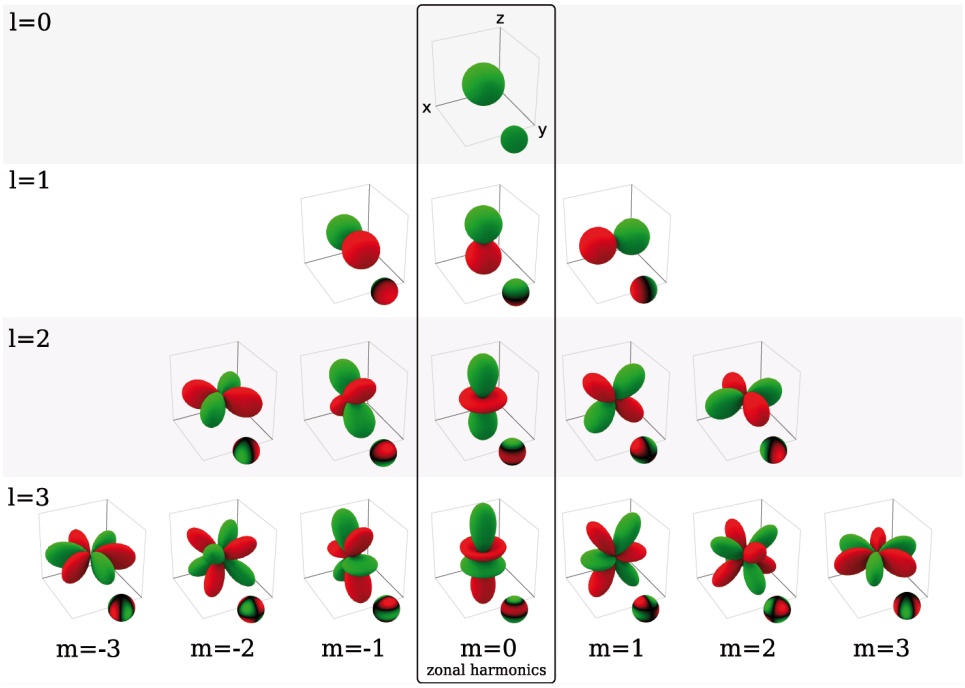


Fig. 2.4: This figure shows the first four bands of the SH basis functions. (Images courtesy of Guerrero [Gue07])

The SH basis functions can furthermore be classified by the way the surface of the sphere gets divided into zones. Figure 2.5 shows the three possible spherical SH function types. Basis functions that have latitudinal divisions are called *zonal harmonics*. *Sectoral harmonics* are basis functions that divide the sphere along its meridians and all the remaining basis functions are called *tesseral harmonics* [Gue07].

Definition Spherical harmonics define an orthonormal basis over the sphere **S** [SKS02]. Using the parameterization

$$s = (x, y, z) = (sin\theta cos\phi, sin\theta sin\phi, cos\theta),$$

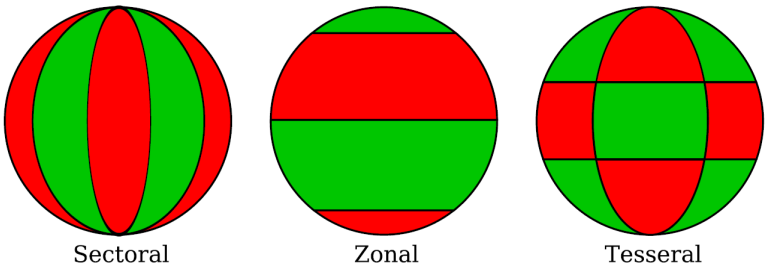


Fig. 2.5: This figure shows the three different basis function types. (Images courtesy of Guerrero [Gue07])

the basis functions are defined as

$$Y_l^m(\theta,\phi) = K_l^m \exp^{im\phi} P_l^{|m|} cos\theta, l \in N, -l \leq m \leq l \tag{2.5}$$

where l is the index of the SH band, m the index in band l, P_l^m are the associated Legendre polynomials and K_l^m are the normalization constants

$$K_l^m = \sqrt{\frac{(2l+1)(l-|m|)!}{4\pi(l+|m|)!}} \tag{2.6}$$

The next paragraphs introduce some important properties of Spherical Harmonics but we will not go into further details.

Projection and Reconstruction Because Spherical Harmonics are an orthonormal basis over **S**, a function f defined over **S** can be projected into its coefficients by the following integral:

$$f_l^m = \int_S f(s) y_l^m(s) ds \tag{2.7}$$

The n-th order reconstruction function is represented via

$$\bar{f}(s) = \sum_{l=0}^{n-1} \sum_{m=-l}^{l} f_l^m y_l^m(s) \tag{2.8}$$

If function f is a low-frequency signal only a few spherical harmonic bands are necessary to approximate f correctly. If it is a high-frequency signal, the SH projection acts like a low pass filter and $\bar{f}$ will be a band limited projection. To have an n-th order representation n^2 coefficients are necessary [SKS02].

A different representation which uses a single-indexed coefficient vector looks as followed

$$\bar{f}(s) = \sum_{i=1}^{n^2} f_i y_i(s) \tag{2.9}$$

where $i = l(l+1)+m+1$. Sloan et al. [SKS02] use Equation 2.9 to calculate the transferred radiance, where f_i is the n^2-component coefficient vector describing the objects radiance transfer function. Figure 2.6 shows three spherical functions that are projected and reconstructed with increasing order of approximation.

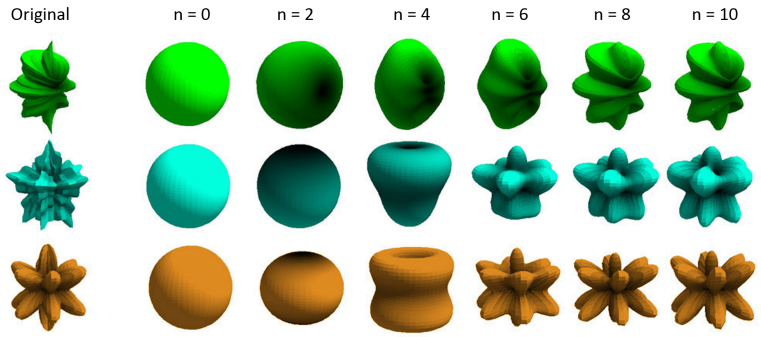


Fig. 2.6: Reconstruction of three spherical functions with increasing order of approximation. (Images courtesy of Green [Gre03])

Rotational invariance Spherical Harmonics are rotational invariant, which is an important property because it ensures that there are no noticeable artifacts like light intensity fluctuations when objects are rotated [Gre03].

Product of coefficients Due to orthonormality of Spherical Harmonics the projections of two functions f and g over **S** satisfy

$$\int_S \bar{f}(s)\bar{g}(s)ds = \sum_{i=1}^{n^2} f_i g_i \tag{2.10}$$

where $\bar{f}$ and $\bar{g}$ are the reconstructed functions of f and g and f_i and g_i the corresponding SH coefficients vectors.

This is of special interest when using diffuse BRDFs because the outgoing radiance can be calculated using a single dot product between the incident lighting and the transfer functions [Gue07].

SH products and the triple product tensor SH products and the triple product tensor are useful for visibility calculation in the SH basis when multiple blockers have to be taken into account [Gue07].

2.3.2 Diffuse BRDFs

The characteristic of diffuse BRDFs is that the exiting radiance is independent from the outgoing direction ω_o. Therefore diffuse BRDFs can be included in the precomputed transfer function coefficient vectors. To speed up calculation of the

outgoing radiance at runtime it is also useful to represent the infinitely distant light source as a set of Spherical Harmonics. This way the runtime calculation reduces to a simple dot product between the incident light SH coefficients vector l and the transfer function SH coefficients vector f_p including the diffuse BRDF

$$l = \int_\Omega L_{env}(\omega_i) y(\omega_i) d\omega_i \tag{2.11}$$

$$f_p = \int_\Omega \rho_p(\omega_i) T_p(\omega_i) y(\omega_i) d\omega_i \tag{2.12}$$

$$L_o(p) = l \cdot f_p \tag{2.13}$$

Note that Equations 2.11 and 2.12 represent the first step in the precomputed radiance transfer approach, the precomputation of the coefficient vectors for the incident light and the transfer functions of the geometry in the scene. Equation 2.13 shows the second step, the simple dot product calculated every frame (see Figure 2.7 for the calculation pipeline).

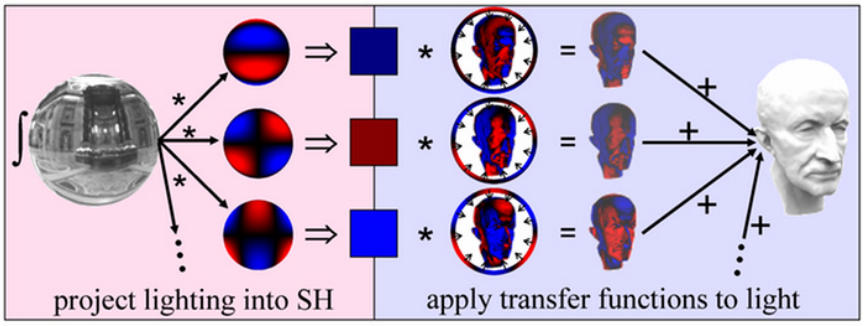


Fig. 2.7: Figure showing the process of outgoing radiance calculation for objects with diffuse BRDFs (Images courtesy of Sloan et al. [SKS02])

2.3.3 Glossy BRDFs

Glossy BRDFs change their incident light transfer behavior dependent on the outgoing direction ω_o. Therefore calculation using Spherical Harmonics is not as trivial as it was for diffuse BRDFs. Sloan et al. [SKS02] were only able to calculate radiance for isotropic BRDFs. This restriction was first eliminated by Kautz et al. [KSS02] in 2002. Later Sloan et al. [SHHS03] added clustering methods to minimize error and calculation time. The main idea behind the approach from Kautz et al. [KSS02] is to store an arbitrary BRDF using a table of SH coefficient vectors for all view directions. Hence every coefficient vector corresponds to one view direction. A second SH projection can now be performed for every single

component over the coefficient vectors. This results in a n^2 x n^2 matrix for glossy BRDFs, where n is the number of SH bands.

$$B_{ij} = \int_{\omega_o} \int_{\omega_i} y_i(\omega_o) y_j(\omega_j) \rho(\omega_i, \omega_o) d\omega_i d\omega_o \tag{2.14}$$

where ρ is an arbitrary glossy BRDF, ω_i the incident and ω_o the outgoing light direction.

The outgoing radiance for an object with glossy BRDF can then be calculated as followed

$$L_o(p, \omega_o) = y(\omega_o)(BR_pT_p)l \tag{2.15}$$

where l is the SH coefficients vector for the incident distant light source. T_p is the transfer matrix that transforms the distant light to incident light at point p. The incident light at point p is rotated into the BRDF's local coordinate system using the SH rotation R_p. B is the matrix that represents the glossy BRDF and y are the SH basis functions [Gue07].

Since evaluation of Equation 2.15 is time consuming Kautz et al. [KSS02] suppose to either fix the view- or the light position. With fixed light the incident light at point p can be precalculated. Using a fixed viewpoint allows precalculation of BRDF coefficients rotated to the lights coordinate system. Therefore no rotation of the incident light is necessary anymore.

Sloan et al. [SHHS03] propose a *clustered principal component analysis* (CPCA) to get higher frame rates. The idea is to partition per-point transfer matrices into fewer clusters that approximate the transfer matrices. Therefore the per-point matrix/vector multiplications are converted into a weighted combination of a few precomputed vectors resulting in a more flexible approach where the restrictions of fixed light- or fixed view position are not necessary anymore to achieve real time frame rates.

2.4 Ambient Occlusion

Ambient Occlusion (AO) is a shading method that takes occlusion by geometry into account. It is an approximation to Global Illumination but normally does not support indirect illumination. However, some of these methods can be easily extended to support indirect illumination.

Definition Ambient Occlusion was introduced by Zhukov et al. [ZIK98] in 1998. It uses the inverted principal as surface exposure and is defined as the percentage of light blocked by geometry close to point p

$$ao(p, n) = \frac{1}{\pi} \int_{\Omega} V_p(\omega) \max(n \cdot \omega, 0) d\omega \tag{2.16}$$

where n is the normal of point p and $V_p(\omega)$ is the visibility function at point p, which returns 0 if no geometry is visible in direction ω and 1 otherwise. The importance of a particular direction is weighted by the cosine between the normal n and direction ω. $\int_{\Omega}$ refers to the integration over the entire hemisphere with respect to n.

Over the last years several AO methods, usable for real time applications, have been introduced. They can be categorized by their way they calculate AO:

1. **Object Based Methods** These methods calculate ambient occlusion on the level of objects. Normally this is done by attaching a texture to each object in the scene. For further reading we refer to the publications *Ambient Occlusion Fields* by Kontkanen and Laine [KL05] and *Fast Precomputed Ambient Occlusion for Proximity Shadows* by Malmer et al. [MMAH06].

2. **Vertex Based Methods** The second group of AO techniques works on vertex level. The methods in this group calculate AO for each vertex and therefore have the problem that they normally need a very high tessellation. The method introduced by Bunnell [Bun05], *Dynamic Ambient Occlusion and Indirect Lighting*, will be described in more detail in Section 2.4.1 as it supports indirect illumination. *Hemispherical Rasterization for Self-Shadowing of Dynamic Objects* was developed by Kautz et al. [KLA04] and uses a rasterizer to calculate AO. *Hardware-accelerated Ambient Occlusion computation* [SSZK04] uses hardware occlusion queries to perform AO computation.

3. **Image Based Methods** The last group are image based methods. Here ambient occlusion is performed in screen space. This way, AO calculation is completely independent from scene complexity, which is a very important feature. The first screen space method was presented by Shanmugam and Arikan [SA07] in 2007. However, it separates AO into low- and high frequency ambient occlusion and is therefore not completely independent of the scene complexity. The screen space method introduced by Ritschel et al. [RGS09], *Approximating Dynamic Global Illumination in Image Space*, will be explained in further details in Section 2.4.2 as it also supports indirect illumination handling.

2.4.1 Dynamic Ambient Occlusion and Indirect Lighting

The presented algorithm was introduced by Bunnell [Bun05] and is able to calculate AO nearly on the fly. Furthermore, it is also possible to use deformable meshes. The idea is to convert vertices into *surface elements*. With a simple extension it is also possible to calculate bounces of indirect light. Figure 2.8 shows

a scene with and without this ambient occlusion approach and also with indirect lighting.

Fig. 2.8: The left image uses only environment lighting without any ambient occlusion. Adding AO (middle) enhances realism but now looks partly too dark. In the image on the right side indirect illumination is also taken into account, which increases realism even more. (Images courtesy of Bunnell [Bun05])

The basic idea behind this method is to treat a geometric mesh as a bunch of surface elements. Surface elements are oriented discs having a position, a normal and an area size. For each vertex of the mesh, a surface element is created by calculating the size of the polygons surrounding it, taking the position and the average normal of the surrounding polygons. A surface element is an approximation of the surface, making it easier to calculate illumination or how much they occlude each other. Figure 2.9 shows the conversion from a mesh to surface element representation.

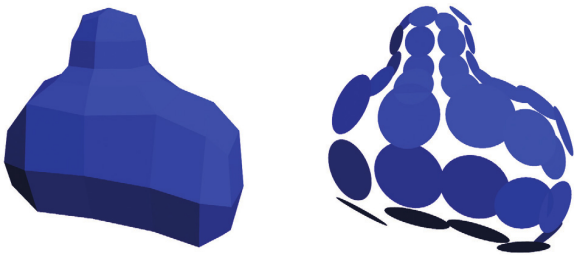


Fig. 2.9: For every vertex a surface element is calculated. (Images courtesy of Bunnell [Bun05])

In order to allow animated meshes the surface elements have to be stored in a texture map, enabling us to update them very fast. The main part of the algorithm is just calculating the accessibility from one surface (the emitter) element

to the other (the receiver). So the receiver's luminance is defined as one minus all other surface elements shadowing the receiver. Bunnell [Bun05] approximates the amount of shadow by an equation, which depends on the area size A of the emitter and the angles between the surface elements:

$$1 - \frac{r \cos \Theta_E \max(1, 4 \cos \Theta_R)}{\sqrt{\frac{A}{\pi} + r^2}} \tag{2.17}$$

As shown in Figure 2.10 Θ_E and Θ_R are the angles between the corresponding disc normals (emitter, receiver) and the connecting line between the surface elements.

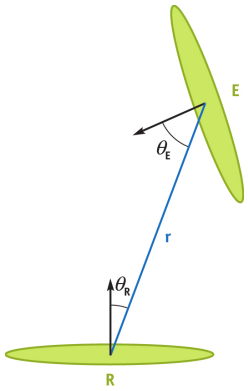


Fig. 2.10: Illustration of radiance transfer between two surface elements. (Image courtesy of Bunnell [Bun05])

Precomputation

The precomputation effort for this method is quite small. As we will show later, each surface element is applied to each other. So we have a complexity of $O(n^2)$, which scales very bad. To work against this circumstances we can build up a hierarchic representation of surface elements. This is done by clustering neighboring surface elements to a single new one. When rendering we traverse this hierarchy and if the emitter surface element is far away, we can stop earlier in the hierarchy thus, enabling better scaling with a complexity of $O(n \log n)$. However, when using animated meshes these surface elements have to be recalculated.

Algorithm outline

The algorithm is executed in two passes. In the first pass, for a receiving surface element, the occlusion of every other surface element is summed and the result is

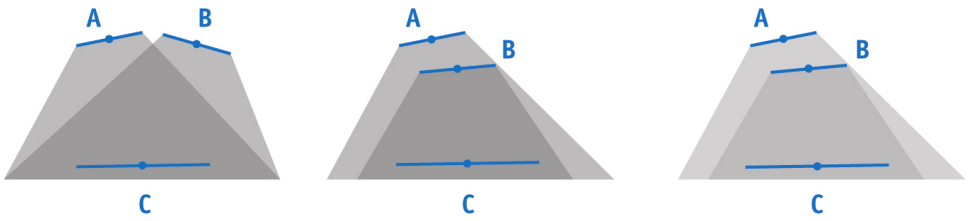


Fig. 2.11: Left: Occlusion calculation is correct. Middle: Occlusion calculation gets too dark. Right: After the second pass, darkening is reduced. (Images courtesy of Bunnell [Bun05])

subtracted from one. This pass gives a first approximation on AO, but since some surface elements cast shadows but are themselves shadowed, receiving surface elements will get too dark. See Figure 2.11 for illustration. In the second pass occlusion is calculated again for a given receiver, but this time the occlusion value for each surface element is multiplied by its own accessibility value from the first pass. With this step any double shadowed surface is corrected. However, tripled shadowed surfaces will still be too dark. Additionally to the accessibility values, the bend normal is calculated and stored. Since all of the previously described calculations are done in a fragment shader and the surface elements are stored in a texture, we have to draw a quad with a size that correlates to the number of vertices/surface elements in the scene.

Indirect Illumination

With some modifications this algorithm can also be used for calculating bounces of indirect light. This is done by replacing the solid angle function 2.17 with a disc-to-disc radiance transfer function. Figure 2.12 shows how the additional bounce improves image quality.

Artifacts

Note that this method suffers from the problem that only vertices are used for calculation. Therefore a scene needs a high tessellation to get convincing results. Otherwise under-sampling will occur as seen in Figure 2.13.

2.4.2 Approximating Dynamic Global Illumination in Image Space

This method introduced by Ritschel et al. [RGS09] belongs to the image based methods. It extends the screen spaced ambient occlusion (SSAO) approach to

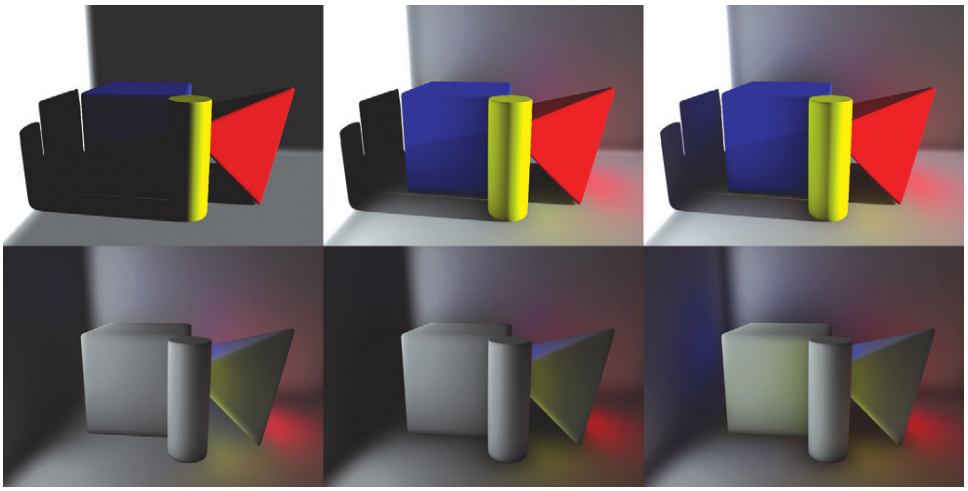


Fig. 2.12: In the top row, the scene is shown with a varying number of indirect light bounces. From left to right: zero, one and two indirect bounces. The bottom row shows the corresponding indirect illumination only. (Images courtesy of Bunnell [Bun05])

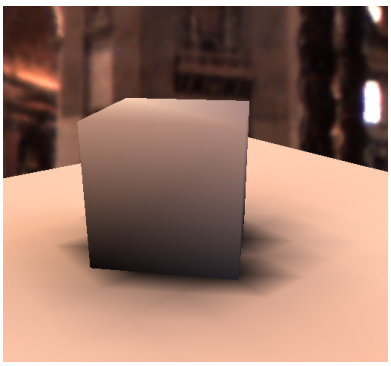


Fig. 2.13: Artifacts due to a coarse mesh. (Image courtesy of Kautz et al. [KLA04])

the so called screen spaced directional occlusion (SSDO) technique which allows one bounce of indirect illumination. Figure 2.14 shows images rendered with this method. Note that everything is done in screen space so the scene and the lighting can be completely dynamic.

The idea behind this approach is to take additional information during AO calculation into account. Ritschel et al. added two extensions to the SSAO approach. The first performs direct lighting using directional occlusion and the second indirect lighting. So the proposed technique needs two passes to calculate SSDO.

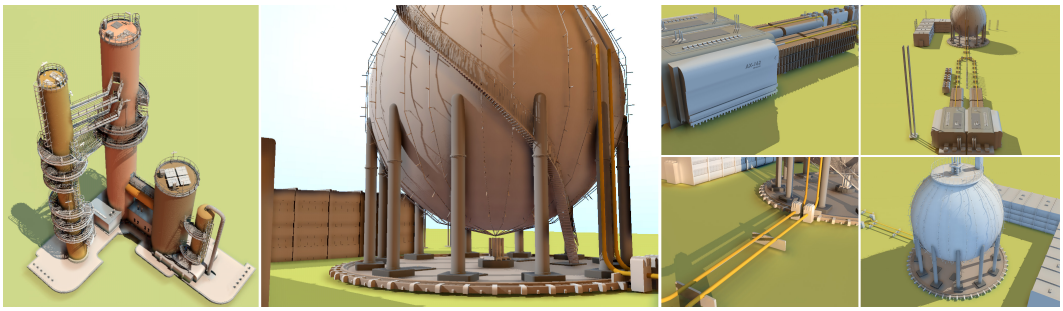


Fig. 2.14: Images rendered with the SSDO method. The scene and the lighting can be completely dynamic. (Images courtesy of Ritschel et al. [RGS09])

Direct Lighting

Standard ambient occlusion calculates a visibility value and performs lighting afterwards by multiplying the outgoing radiance with the AO value. In this case they sample around a 3D point p in screen space and calculate direct lighting only for those samples that do not occlude the current point p. Figure 2.15 (left) illustrates this process. Here sample points A, B and D lie below the surface and therefore they occlude incoming light and are not taken into lighting calculation. Instead, sample point C lies above the screen spaced depth value and the direction from point p to sample point C can be used for lighting calculation. The equation for calculation direct lighting looks as follows:

$$L_o(p) = \sum_{i=1}^{N} \frac{\rho}{\pi} L_{env}(\omega_i) V(\omega_i) max(0, n_P \cdot \omega_i) \triangle \omega \tag{2.18}$$

where N is the number of sample points, L_{env} is the incoming light and $V(\omega_i)$ is the visibility function, which is calculated using the described sampling method above. $\frac{\rho}{\pi}$ is the diffuse BRDF of the surface and $\triangle\omega = \frac{2\pi}{N}$ the solid angle covered by each sample.

Indirect Bounces

The calculation of indirect light bounces is somehow the opposite operation to directional light. Here the interesting sample points are A, B and D instead of C. Their position, normal and the calculated direct light from the previous pass can be used to calculate one light bounce with the following equation

$$L_{ind}(P) = \sum_{i=1}^{N} \frac{\rho}{\pi} L_{occ_i} (1 - V(\omega_i)) \frac{A_s cos\theta_{s_i} cos\theta_{r_i}}{d_i^2} \tag{2.19}$$

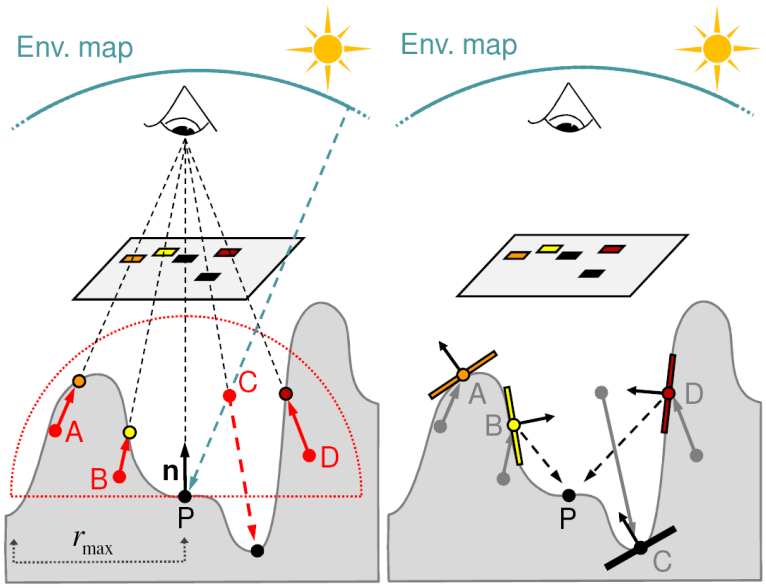


Fig. 2.15: The left graphic illustrates directional occlusion and the right indirect light calculation. (Images courtesy of Ritschel et al. [RGS09])

where d_i is the distance between point P and occluder i and L_{occ_i} is the radiance of occluder i. θ_{s_i} and θ_{r_i} are the angles between the sender/receiver normal and the transmittance direction. A_s represents the patch size of occluder i and is set to $A_s = \pi r_{max}^2/N$ with r_{max} being the maximal sampling distance. Figure 2.15 (right) illustrates the process of indirect light bounce calculation.

2.5 Implicit Visibility and Antiradiance for Interactive Global Illumination

Implicit Visibility and Antiradiance for Interactive Global Illumination was introduced by Dachsbacher et al. [DSDD07] in 2007. The basic idea is to avoid explicit visibility queries by introducing the concept of Antiradiance. Antiradiance can be understood as negative light, that is *shining* behind the surface of an object that occludes light. Normally visibility queries determine if an object blocks light or not. However, in this approach there are no radiance blocking objects. Instead Antiradiance is used to remove radiance that should not reach areas that are in shadow. Figure 2.16 shows an oriental room scene, which was rendered using Antiradiance.

Fig. 2.16: Oriental room scene rendered with Antiradiance and implicit visibility. (Image courtesy of Dachsbacher et al. [DSDD07])

2.5.1 Reformulation of the Rendering Equation

The rendering equation (see Section 1.1.2) has to be reformulated to get a form where visibility can be calculated implicitly. For better understanding we will revisit the rendering equation here. Dachsbacher et al. introduced several new operators. The first one is called the *reflection* operator **K**. This operator returns the outgoing radiance L at point x by computing the shading integral:

$$(\mathbf{K}L_i)(x,\omega_o) = \int_\Omega \rho(x,\omega_i,\omega_o)L_i(x,\omega_i)(n_x \cdot \omega_i) \tag{2.20}$$

where x is the point to be shaded, ω_i the direction of the incoming light L_i, ω_o the outgoing direction and $\rho(x,\omega_i,\omega_o)$ is the BRDF at point p.

The incoming light L_i is described through a *geometry* operator **G** which uses an explicit visibility query $ray(x,\omega)$ that returns the first hit point. The ray starts from x in direction $-\omega$. The incoming light can then be written as

$$L_i(x,\omega_i) = (\mathbf{G}L)(x,\omega_i) = L(ray(x,\omega_i),\omega_i) \tag{2.21}$$

After these steps, the rendering equation becomes

$$L(x,\omega_o) = E(x,\omega_o) + (\mathbf{KG}L)(x,\omega_i) \tag{2.22}$$

where $E(x, \omega_o)$ is the self-emission at point x.

Up to now, there is still an explicit visibility query in the **G** operator. The goal of Dachsbacher et al. [DSDD07] was to eliminate this by introducing Antiradiance. Therefore they introduced another operator **U**, that uses a modified ray function called RAY. Instead of returning the first hit point the RAY function returns a set of all hits and the new illumination is calculated as follows:

$$L_i^{unocc}(x, \omega) = (UL)(x, \omega) = \sum_{y \in RAY(x,\omega)} L(y, \omega) \tag{2.23}$$

This results in an illumination from all intersection points into direction $-\omega$, thus occlusion is not taken into account. To avoid this *overillumination*, Antiradiance is created with exactly the same amount of incident light at an object.

To connect the operators **G** and **K**, Dachsbacher et al. introduced a new operator **J**. **J** is a simple *go-through* operator that forwards incident light.

$$(\mathbf{J}L_i)(x, \omega) = L_i(x, \omega) \tag{2.24}$$

Figure 2.17 shows the function of all introduced operators. Note that operator **J** just forwards incident light and it can be used to describe $\mathbf{U}L$ by summing up all combinations of **G** and **J**.

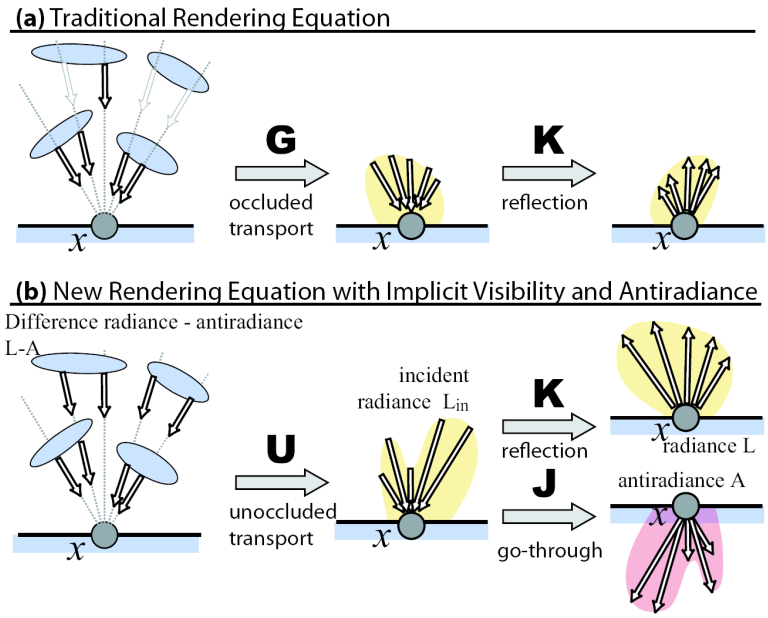


Fig. 2.17: The illustrations show the different formulations of the rendering equation. (a) Shows the traditional rendering equation using operator **G** and **K**. (b) Shows the new formulation using **U**, **K** and **J** operators. (Images courtesy of Dachsbacher et al. [DSDD07])

$$\mathbf{U}L = \mathbf{G}L + \mathbf{GJG}L + \mathbf{GJGJG}L + ... \tag{2.25}$$

This leads to the equation that relates **G** and **U**:

$$\begin{aligned} A &= \mathbf{J}(\mathbf{U}L - \mathbf{UJG}L) && (2.26)\\ &= \mathbf{JU}(L - A) && (2.27) \end{aligned}$$

where $A = \mathbf{JG}L$ is the Antiradiance. Note that Antiradiance is defined in a recursive way, so it can be calculated step wise using the previous result. Antiradiance can now be placed in the final reformulated rendering equation:

$$\begin{aligned} L &= E + \mathbf{KU}(L - A) && (2.28)\\ A &= \mathbf{JU}(L - A) && (2.29) \end{aligned}$$

2.5.2 Hierarchical Radiosity

For fast rendering, Dachsbacher et al. [DSDD07] use Hierarchical Radiosity with clustering (HRC) similar to Smits et al. [SAG94]. To setup the hierarchy they used the CPU, whereas for everything else the GPU is used. For each iteration four main steps have to be performed:

1. **Global Pass** In the global pass light is transferred from sender elements to receiver elements. Note that there is no visibility calculation used, thus this pass corresponds to the operator **U**.

2. **Push Operation** The hierarchy has to stay consistent and therefore two operations exist. The first is the push operation, where energy is pushed down the hierarchy. The second step is the pull operation.

3. **Local Pass** In the local pass reflected radiance and antiradiance is calculated, which corresponds to the operators **K** and **J**.

4. **Pull Operation** As mentioned before the pull operation ensures a consistent hierarchy. In this operation energy is pulled from the leaves to its parents. After this step, the hierarchy is in a consistent state again and the next iteration can be performed.

Note that this was a rather short outline of their algorithm. For implementation details we refer to the according publication [DSDD07].

2.6 Instant Radiosity

Instant Radiosity (IR) was first introduced by Alexander Keller [Kel97] in 1997. It is a excellent technique to calculate global illumination for diffuse or not-too-shiny materials directly from the rendering equation (see Section 1.1.2). The main idea is to place so called Virtual Point Lights (VPL) in the scene, which are then used to compute the overall illumination for a given screen pixel.

2.6.1 Introduction

Instant Radiosity splits each path $\bar{x} = \{x_0, x_1, ..., x_n\}$ that light can take from the light source to the camera into three main parts [Seg07]:

- The first element of this path $\bar{x_c} = \{x_0, x_1\}$ corresponds to a sensor element in the camera x_0 and a surface point x_1 seen by this sensor. In other words the way from the screen pixel (x_0) to its corresponding 3D point (x_1) in the scene.
- After x_1 follows x_v, which is a point that illuminates x_1 - the location of the virtual point light.
- The rest of the path called $\bar{x_s}$ beginning with x_v is the path to a light source. It can have an arbitrary length. However, note that a length of 0 is also possible and then x_v is located at the light source.

If we want to have N VPLs placed in the scene we generate N random paths $\{x_v, \bar{x_s}\}$ beginning at the light sources. These sub paths are then reused for every path that starts from the cameras screen pixel (x_0) to the surface point (x_1). Figure 2.18 illustrates this process, where first (a) the paths from the light sources are build and afterwards (b) the surface point is illuminated by all the light sources plus the placed Virtual Point Lights.

Overmodulation Problem

Instant Radiosity is a very easy method to get fast global illumination. However, it suffers from a serious overmodulation problem, which means that the variance is not bound. The visibility term decreases with $1/r^2$ and the problem gets obvious when r gets close to 0. A common solution to this is to clamp the radius. Note that this introduces a bias.

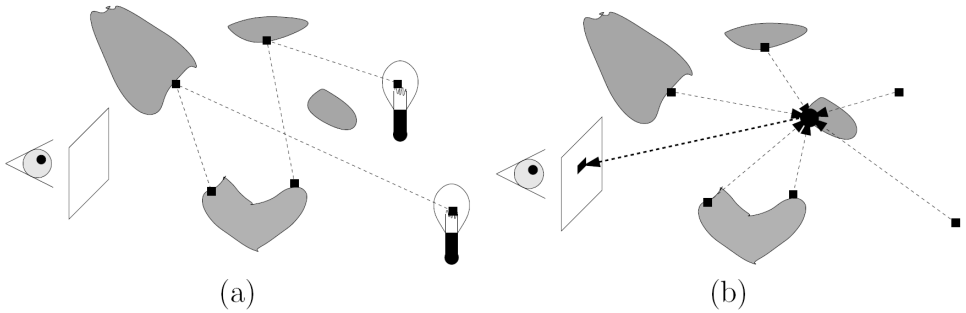


Fig. 2.18: (a) First, the light paths from the light sources are created. Each light bounce is stored as a VPL. (b) Then, the illumination for a given surface point is calculated by taking all the Virtual Point Lights and the directional light into account. (Images courtesy of Segovia [Seg07])

Variance Reduction Techniques

Instant Radiosity can be used in real-time applications. However, to get high enough frame rates it is important to use as less virtual point lights as possible to reduce shading computation costs. But at the same time the image quality should be as high as possible with the given amount of virtual point lights. This is the point, where optimal sampling gets important. Keller [Kel97] proposed to use a quasi-random walk [Kel96] on the method of quasi-Monte Carlo integration. Therefore he generates samples from Halton sequences, which have a good distribution while having low discrepancy instead of pure random samples. The problem here is, that these paths are generated regardless of the position of the camera or the properties of the surfaces. A location for a VPL is useless if it is not able to illuminate surface points, that can be seen by the camera. So what is really needed are "importance driven" methods that have sampling strategies suitable for the camera, light and scene setup.

There exist some methods that address this problem. Wald et al. [WSB01] proposed to calculate the power brought from each physical light source to the camera. The VPLs are then assigned to the physical light sources according to their contribution on the screen. Segovia et al. introduced *Bidirectional Instant Radiosity* [SIP06], where samples are also started from the camera to find good locations for virtual point lights. *Metropolis Instant Radiosity* [SIP07] is a method, which uses a Multiple-try Metropolis Hastings algorithm. The point of metropolis sampling is, to find good paths by altering existing good paths. An interesting property of Metropolis Instant Radiosity is, that each VPL provides the same amount of power to the camera. For more information we refer to the Ph. D. Thesis of Benjamin Segovia [Seg07].

Instant Radiosity in Real-Time Applications

When using Instant Radiosity in real-time applications two tasks are very time-consuming. The visibility function, that determines if a given VPL is visible from a certain surface point and the shading for all virtual point lights. The two techniques introduced in the next sections address these challenges. Section 2.6.2 addresses the problem of visibility calculation and Section 2.6.3 tries to minimize shading costs.

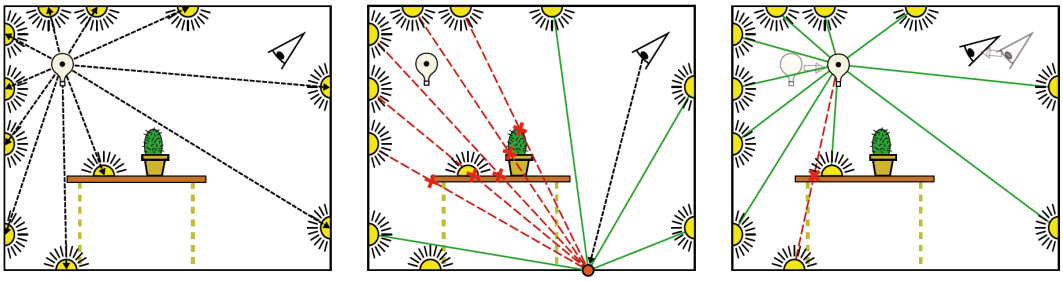
2.6.2 Incremental Instant Radiosity

Incremental Instant Radiosity is a technique introduced by Laine et al. [LSK+07] in 2007. It addresses the visibility problem in Instant Radiosity by trying to cache virtual point lights over several frames and thus, calculating visibility only once in the "lifetime" of a VPL. Laine et al. proposed a stable and fast algorithm to cache VPLs while keeping a good distribution over time to the price of dynamic scenes. Dynamic objects will not contribute to indirect illumination calculation although they can be illuminated by it. Figure 2.19 shows an image rendered with the proposed method.

Fig. 2.19: Image rendered using the incremental instant radiosity method introduced by Laine et al. (Image courtesy of Laine et al. [LSK+07])

The first step is to shoot rays from the light source and place VPLs at the hit points. Note that only static geometry is used for ray intersection computation. Then the scene is rendered illuminated with the primary light source and the VPLs. In the next frame a visibility test is performed, that finds out, which virtual point lights are not visible from the primary light source any more. This could happen

due to orientation/position change of the light source. Figure 2.20 illustrates this process.



(a) (b) (c)

Fig. 2.20: (a) Rays are shot from the primary light source. At the hit points virtual point lights are placed. (b) The scene is rendered by using the VPLs to illuminate all the visible surface points. (c) For each VPL a visibility test is performed to delete invalid VPLs (Images courtesy of Laine et al. [LSK+07])

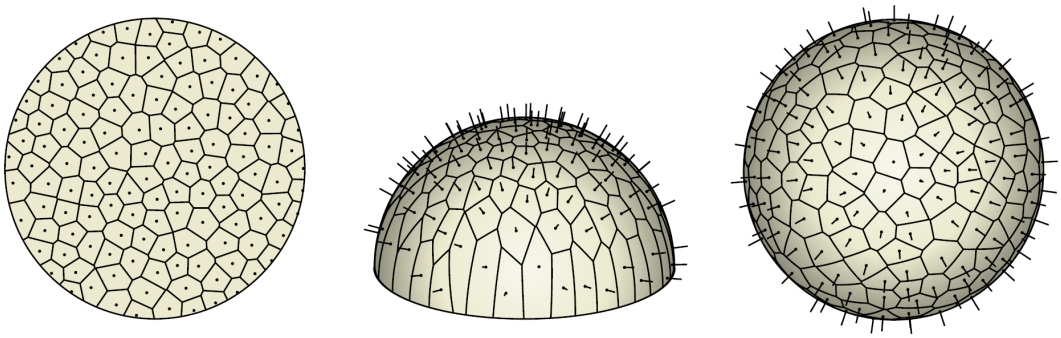
Distribution domains

The implementation presented by Laine et al. supports two types of light sources. $180°$ spot lights with a cosine falloff and omnidirectional point lights. Their key to a good VPL distribution for spot lights is to map the VPLs seen by the spot light onto the unit disc by flattening the z-coordinate of the hemisphere pointing at the spot direction. See Figure 2.21 for illustration. Note that the size of the projected areas corresponds with the cosine weighted energy distribution of the spot light (see Nusselt analog [CWH93]). For omnidirectional lights Laine et al. operate on the unit sphere.

All computations for redistributing VPLs are then performed in those 2D domains. An often used operation is the Delaunay triangulation and the associated Voronoi diagram. For those calculations they used the CGAL computational geometry library [Boa08]. For calculation of the Delaunay triangulation on the unit sphere they proposed to use a 3D tetrahedalization on the VPL sampling points. However, in their presentation slides to the publication they mention that a 3D tetrahedralization is nothing else than the convex hull, which can be calculated much faster. In the algorithm outline we will describe how these computations can be useful for VPL caching.

Quality control

To get good quality images over a longer period of time it is necessary to introduce $recalcMin$ and $recalcMax$. These two values bound the maximum and minimum



(a) (b) (c)

Fig. 2.21: These images show the different domains for the light sources. a) Shows the unit disc with the projected VPL directions. In this 2D space the redistribution and creation/deletion of VPLs is performed and afterwards backprojected to the hemisphere as seen in figure b). Figure c) shows the Delauney Triangulation on the unit sphere for an omnidirectional point light source. (Images courtesy of Laine et al. [LSK+07])

amount of recalculated VPLs per frame. Another value $maxVPLs$ defines the maximum number of virtual point lights in the scene. We will show in the next section how they influence quality.

Algorithm outline

The following steps are executed for each frame:

1. **Determine validity of each VPL** The first step is to determine if a VPL is visible. Laine et al. used a conventional CPU ray tracer. Note, that it is also possible to use the shadow map from the primary light source to determine visibility for a VPL.

2. **Remove invalid VPLs** All VPLs that are flagged to be invalid get deleted. It can also happen that $maxVPLs - numberOfVPLs < recalcMin$. That means, that it is not possible to recalculate $recalcMin$ VPLs because the maximum amount of VPLs has been reached. In this case also valid VPLs have to be deleted. Laine et al. used the Delauney triangulation to find the optimal VPL so that a more uniform distribution is reached because VPLs are deleted in areas with high densities. The first step is to find the shortest Delauney edge. For the two adjacent vertices they find the *second closest* neighbors of them. The vertex/VPL with the smaller distance to the second closest neighbor gets deleted. This is repeated until there is room for $recalcMin$ new VPLs. Note that after each deletion recalculation of the Delauney triangulation is necessary.

3. **Create new VPLs** For creating new VPLs again the Delauney triangulation and the corresponding Voronoi diagram is used. *Dispersion*, introduced by Niederreiter [Nie92], is a metric for quality measurement of point sets and is computed as the largest empty circle in the domain

$$\delta(P) = \sup_{x \in X} \min_{p \in P} d(x, p) \tag{2.30}$$

where P is a point set in the metric space (X, d). Since we want to place new virtual point lights in regions that have low density we have to find point p where the largest empty circle could be placed.

Aurenhammer et al. [AK99] pointed out, that there are three possible cases for the position of the largest empty circle in a 2D point set. Note that the point set has to be bounded with a convex polygon.

(a) In the first case the center of the circle lies on a vertex of the Voronoi diagram, that touches three points.

(b) In the second case the circle is positioned at the intersection of an infinite Voronoi edge with the bounding convex polygon. It also has to touch two points.

(c) The last case is, where the circle touches two points and the center of it lies on a vertex of the convex polygon.

For the unit sphere only case a) is important, whereas for the unit disc the bounding circle is seen as an infinite tessellated convex polygon and therefore all 3 cases may occur. With this 3 cases and the Delauney triangulation as well as the Voronoi diagram, finding the right point requires only enumerating Voronoi vertices and edges.

Once the point was found it will be added and the Delauney triangulation is performed again. A ray corresponding to the new sample point is then cast into the scene. If there is no hit point there will be no VPL created. Otherwise the VPL will be placed. The VPL color depends on the light source- and the surface color. Note that for textured surfaces it is useful to take a heavily blurred version for color lookup. For every new VPL a paraboloid shadow map is rendered [OBM06]. This step is repeated until $recalcMax$ VPLs got created or the number of VPLs reaches $maxLights$.

4. **Compute intensities** Since the light source moves from frame to frame but the bigger part of VPLs will stay at the same location their intensities have to be recalculated to guarantee high quality illumination. Since we

are working in the 2D domain the Voronoi diagram can be used again to perform this computation. As mentioned before the area where a sample point is located in directly corresponds to the light contribution from the light source. The intensity of a virtual point light can therefore be calculated by

$$I_{VPL} = \frac{A_{VPL}}{\bar{A}} I_{light} \tag{2.31}$$

where I_{VPL} is the intensity of the VPL, A_{VPL} is the area of the Voronoi diagram where the sample point lies, $\bar{A}$ is the overall area and I_{light} is the primary light source intensity.

5. **Render G-Buffer** A deferred rendering system is used to display the scene. The G-Buffer is an off-screen frame buffer that stores the world positions, normals and colors of the visible pixels.

6. **Split G-Buffer** Since illumination computation for each pixel is expensive due to the massive amount of virtual point lights it is important to shade only as few pixels as necessary. One approach therefore is to only calculate illumination by a VPL on a subset of pixels in the G-Buffer. Interleaved Sampling introduced by Keller et al. [WKH01] is a suitable method here. The G-Buffer gets splitted into $n \times m$ tiles and each tile represents the whole G-Buffer image, but uses a different subset of pixels. Thus pixel (a, b) in tile (i, j), with $i \in [0, ..., n-1]$ and $j \in 0, ..., m-1$ is read from position $(an + i, bm + j)$ in the G-Buffer.

   Segovia et al. [SIMP06] mentioned to do interleaved sampling in a multiple pass approach to get better cache coherency. However, Laine et al. [LSK+07] found out that current graphics hardware performs better using a single render pass.

7. **Accumulate illumination** In the illumination accumulation phase each virtual point light gets assigned to one tile. The shading is then performed using the paraboloid shadow map of the VPL on the assigned tile. To omit artifacts that may occur when the distance from the VPL to the current surface point is near zero, the maximal contribution of each VPL is clamped (see Section 2.6.1). The result of each shading process for a VPL is additively added to an accumulation buffer.

8. **Combine illumination** Once the illumination for every Virtual Point Light was calculated the $n \times m$ tiles have to be combined again. In this step the inverse operation to the splitting step is performed.

9. **Smooth illumination** Since the illumination of the VPLs was only applied on a subset of pixels the combined accumulation buffer has to be filtered to get good visual results. Therefore Laine et al. [LSK+07] used a geometry aware box filter. They introduce two new threshold values α and β where α is the threshold for the maximum spatial distance and β for the maximum difference between the normals. Figure 2.22 shows the combined accumulation buffer and the filtered final result.

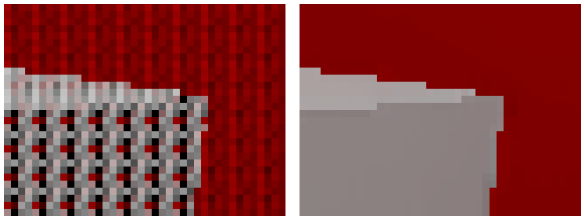


Fig. 2.22: Image on the left shows the unfiltered combined accumulation buffer. On the right the final accumulation buffer after the geometry aware box filter was applied. (Images courtesy of Laine et al. [LSK+07])

2.6.3 Splatting Indirect Illumination

As outlined in Section 2.6.1 the two computational expensive operations in the Instant Radiosity approach for real time applications is the visibility determination and the shading. The *Incremental Instant Radiosity* approach (see Section 2.6.2) addresses the problem of the expensive visibility function by caching the virtual point lights over several frames. The method *Splatting Indirect Illumination* introduced in this section addresses the problem of expensive shading. The method was developed by Dachsbacher et al. [DS06] and is also based on the Instant Radiosity approach.

The main idea in this approach is to reduce the number of illumination computations per pixel by decreasing the influence area of a virtual point light. In the Incremental Instant Radiosity method, each virtual point light illuminates one tile of the split G-Buffer. However, it is often a waste of computation power since the influence of the virtual point light is negligible at some areas of the scene. Therefore, Dachsbacher et al. use a sphere mesh geometry that is splatted on screen space, covering the influence area of a VPL. This way they minimize the number of pixels that are illuminated by a VPL. Figure 2.23 illustrates the splatting of the sphere mesh (left). Note that diffuse as well as glossy virtual point lights are supported in this approach and thus caustics can be rendered (right). To get proper caustics the number of virtual point lights has to be quite high, but at the same time the covered area of each VPL is reduced because of the high specular term.

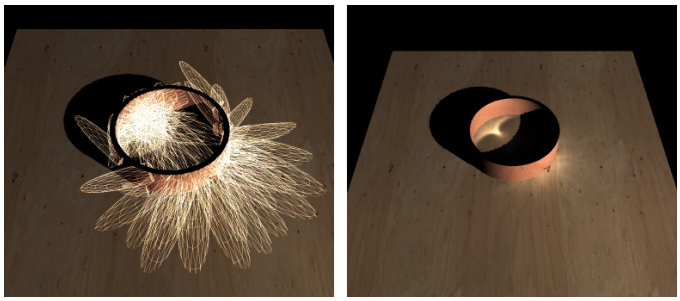


Fig. 2.23: Image on the left shows the splatted geometry for VPLs placed on the rings surface. Image on the right shows the resulting image with fine caustic effects. (Images courtesy of Dachsbacher and Stamminger [DS06])

Algorithm outline

The following section gives an overview of the method. The presented algorithm makes use of the so called *Reflective Shadow Maps* (RSM), which have been introduced by Dachsbacher and Stamminger in 2005 [DS05]. Reflective Shadow Maps extend a standard shadow map by additionally storing the surface normal, world space position and the reflected flux using multiple render targets.

1. **Reflective Shadow Maps** Each pixel of a Reflective Shadow Map can be considered as a secondary light source - in the RSM publication they are called pixel lights but we will refer to them as virtual point lights. The emitted radiant intensity for a VPL can be calculated as followed

$$I_p(\omega) = \phi_p \max(0, n_p \cdot \omega) \tag{2.32}$$

   where p is the given VPL, ϕ_p is the reflected flux, n_p the normal at p and ω the emittance direction. To calculate the irradiance at point x through VPL p they suppose the following equation

$$E_p(x, n) = I_p(\frac{x - x_p}{\|x - x_p\|}) \frac{\max(0, n \cdot (x_p - x))}{\|x_p - x\|^3} \tag{2.33}$$

   where x_p is the position of VPL p.

2. **Importance Sampling** In the publication, where Dachsbacher and Stamminger introduced Reflective Shadow Maps they sampled the RSM to gather the incident light for a given point p_g. But this method is quite inconvenient since is too expensive. Therefore they propose to sample the RSM and place VPLs at the according positions.

Since some areas in the scene, which are illuminated by the light source have a bigger influence because of a higher reflected flux it makes sense to place more VPLs in those parts. Furthermore, if there are objects with glossy surfaces, the VPLs will illuminate a smaller area but with higher intensity. Therefore it is important to place many VPLs in those areas to omit artifacts. Importance sampling is the right method to redistribute a uniform sample set accordingly. The buffer which stores the flux in the RSM gets replaced by a so called importance buffer. The importance of a given pixel in the RSM can be calculated as followed:

$$p_\phi = \phi(1 - O)P \tag{2.34}$$

where ϕ is the flux at a given RSM pixel. P is the phong exponent thus the more glossy a surface is, the higher its importance. O is the ambient occlusion term (see Section 2.4). The ambient occlusion term ensures that VPLs are not placed at edges or corners where the occlusion is very high, because those VPLs would have only less contribution to the scene.

Clarberg et al. [CJAMJ05] introduced a elegant method to efficiently build importance driven sample sets. Their approach is called *Hierarchical Warping* which performs a hierarchical recursive redistribution of a uniform sample set to get the *importance driven* distribution. Importance Sampling by Clarberg et al. will be discussed in more detail in Section 3.3.

3. **Splatting Indirect Illumination** After the sample set was redistributed according to the importance map indirect illumination is calculated in screen space. As mentioned before a sphere geometry mesh gets splatted onto the screen, which covers the illumination zone of a given VPL. Figure 2.24 shows the emission for a diffuse and glossy VPL whereas Figure 2.25 shows the according adapted bounding geometry that will be used for indirect illumination calculation.

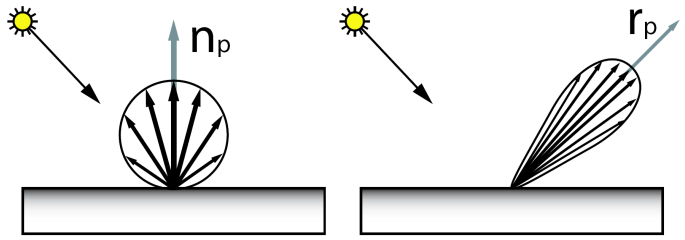


Fig. 2.24: The illustration shows emission for diffuse (left) and glossy VPLs (right) (Images courtesy of Dachsbacher and Stamminger [DS06])

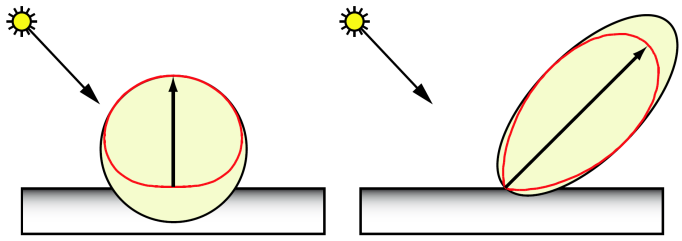


Fig. 2.25: Illustration that shows the bounding geometry for a given VPL (diffuse and glossy).(Images courtesy of Dachsbacher and Stamminger [DS06])

To calculate the correct bounding geometry each sphere mesh gets a virtual point light *id* attached. With that *id* the information (position, normal etc.) for a given VPL can be looked up from a texture in the vertex shader. The resulting bounding geometry is then the iso-surface of illumination and only covers those pixels where the illumination is higher than a given threshold.

By splatting the bounding geometry the shading calculations can be reduced to a minimum. However, note that Dachsbacher et al. did not perform any visibility calculations for the indirect illumination.

2.6.4 Interactive Global Illumination Based on Coherent Surface Shadow Maps

The method presented in this section was introduced by Ritschel et al. [RGKS08] in 2008. It is a novel method to calculate global illumination by precomputing visibility. The technique used for visibility queries is based on Coherent Shadow Maps [Rit07]. Figure 2.26 shows three scenes, rendered using coherent surface shadow maps.

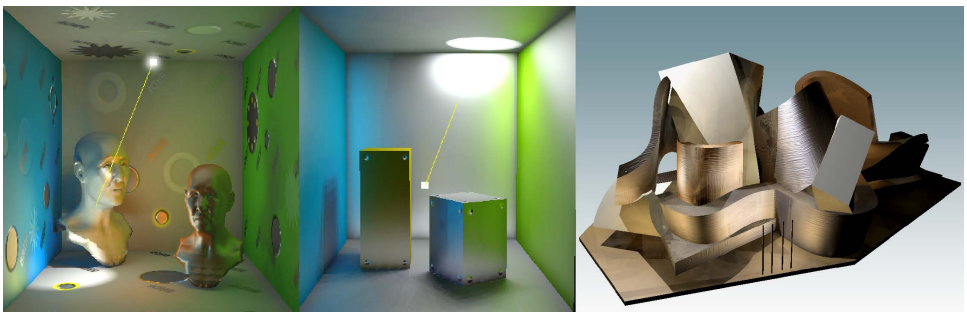


Fig. 2.26: Images rendered with coherent surface shadow maps. (Images courtesy of Ritschel et al. [RGKS08])

Coherent shadow maps are an excellent method to compress visibility data.

The idea is to take several coherent depth maps and use a lossless compression scheme to reduce memory cost. Figure 2.27 shows the Standford dragon rendered from several points of view. The resulting depth maps are then stored in a list, so that their coherence to each other is very high.

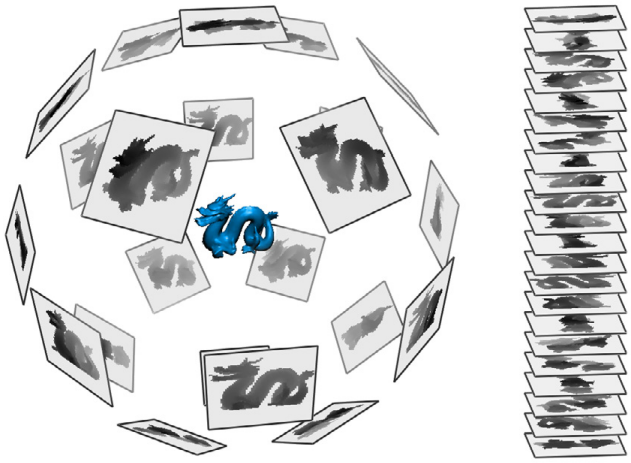


Fig. 2.27: Left: Standford dragon rendered from multiple views. Right: List of coherent depth maps. (Images courtesy of Ritschel et al. [Rit07])

Compression The compression is performed on a per pixel basis of the depth maps. This means that the following compression steps are performed on a fixed pixel location (x, y) in an arbitrary depth map i. The depth value of depth map i at pixel position (x, y) will be denoted as $z(i)$.

Weiskopf and Ertl [WE03] introduced the concept of dual depth layer shadow maps. Figure 2.28 shows an object and the resulting depth map. Note that the z values of this depth map lie between the front and the back faces of the object from the point of view of the light source. As long as the depth values stay inside the interval of the front and the back faces any depth value can be used to perform visibility tests. This fact is exploited by the compression algorithm and the idea is to find a depth value that is convenient for several coherent shadow maps.

Figure 2.29 shows multiple z values for the Standford dragon. If we now take several coherent shadow maps into account, shooting a ray always from pixel position (x, y), we get several depth values for the first (z_1) and the second (z_2) intersection.

For a high compression rate it is important to find a z_{avg} value that works for as much coherent shadow maps as possible. Mathematically speaking the following inequation has to be true:

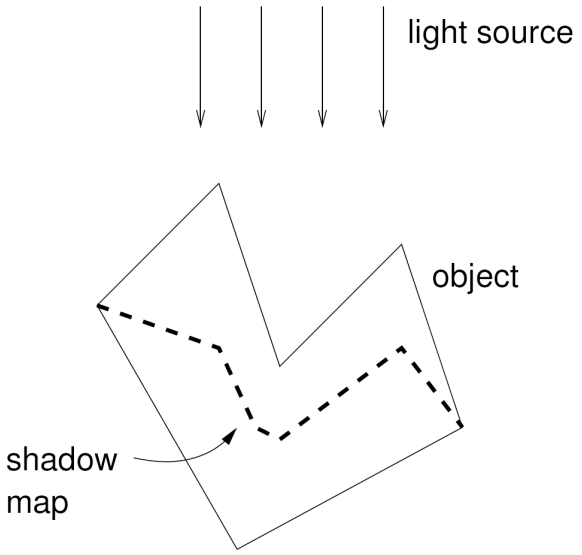


Fig. 2.28: Illustration shows the depth values of a shadow map with depth values between the front and the back faces of an object. (Image courtesy of Weiskopf and Ertl [WE03])

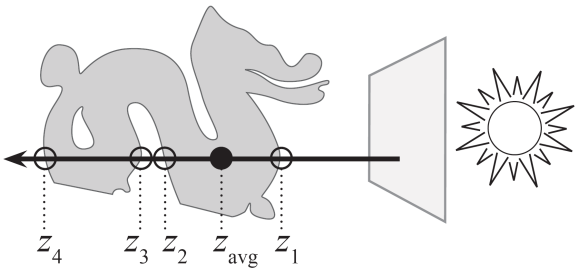


Fig. 2.29: Depth value z_{avg} which lies between z_1 and z_2 is sufficient to get correct visibility results. (Images courtesy of Ritschel et al. [Rit07])

$$\max\{z_1(1), z_1(2), ..., z_1(i_{end})\} < \min\{z_2(1), z_2(2), ..., z_2(i_{end})\} \tag{2.35}$$

where i_{end} is the largest depth map index for that the inequation is true. z_{avg} is then the average between the $\min$ and $\max$ values. With this method we end up with a list of segments for each pixel. Each segment therefore stores z_{avg} and its corresponding end index i_{end}. This process is repeated for every pixel in the shadow map, ending up with $M \times M$ segment lists, where M is the shadow map dimension.

Depth comparison Depth tests are quite similar performed as it is done for normal shadow mapping. If we want to do a shadow test for shadow map i we

have to transform point p into the shadow map space, perform the projection and afterwards do the perspective division. We end up with point $q(i)$ in the projected shadow map space. $q_{xy}(i)$ gives us the segment list and all that has to be done, is to find the right segment in the list:

$$i_{end}(j-1) < i \leq i_{end}(j) \tag{2.36}$$

where j is the index of the segment that contains the z_{avg}, which has to be used for the shadow test.

Coherent Surface Shadow Maps Coherent Surface Shadow Maps (CSSM) are an extension to the coherent shadow maps. While coherent shadow maps were used from an *outside in* perspective (see Figure 2.27) coherent surface shadow maps are placed on the surface of an object and therefore using an *inside out* perspective. The main challenges here are to get a good coherence and to place the CSSMs in a way, that high frequency visibility queries are possible. As the CSSMs are located on the surface they record depth values in all directions through using cube maps.

Precomputation

In the preprocessing step the Coherent Surface Shadow Maps have to be created for the complete surface of the scene. Ritschel et al. [RGKS08] therefore used a *texture atlas* that contained the position, normal, area, radiance and BRDF parameters. The atlas is divided into so-called *charts*. Each chart contains information about a connected area in the scene. See Figure 2.30 which shows a texture atlas for a Cornell Box scene.

The texture atlas can be used to get locations on the surface for the CSSMs. Ritschel et al. [RGKS08] propose two methods to place the CSSMs to get good coherency. The first is a *Zig-Zag* strategy, where the texels of each chart are traversed line by line - every texel is used to render a cube depth map. If a chart was finished, the chart with the nearest 3d position is used to continue.

The second strategy is the *Spiral* one. Here the border of a chart is traversed and each texel in a chart is only visited once. The traversal path ends up in a spiral. Figure 2.31 shows both traversal strategies with three charts.

Coherent Surface Shadow Maps allow for self occlusion tests, since they are working in an inside out perspective. However, when there are moving objects in the scene, CSSMs are not a suitable structure to do visibility tests. Therefore each object in the scene contains an additional Coherent Shadow Map. CSMs are suitable to test if a given object occludes a ray between two points. A visibility test in a scene with moving objects between the points p_i and p_j is calculated as follows: First each object tests with its CSSM if there is self occlusion. If one or

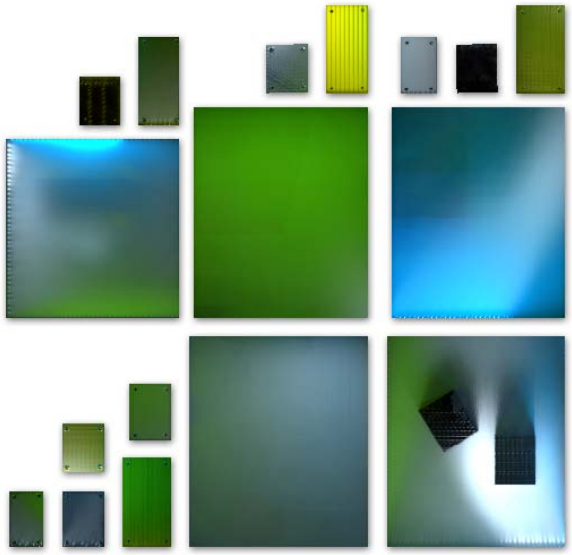


Fig. 2.30: Texture atlas with several charts from a Cornell Box scene. (Images courtesy of Ritschel et al. [RGKS08])

both tests return *occlusion*, then there is occlusion. If both tests turn out to have no occlusion the CSMs of all the other objects are used to test if they do intersect the ray between the points p_i and p_j. Note that this test only turns out correct as long the objects do not intersect with their convex hulls.

Algorithm outline

The following section will give an overview of the steps needed to compute global illumination based on Coherent Surface Shadow Maps.

1. **Hierarchical Radiosity** Ritschel et al. [RGKS08] used a modified version of the Hierarchical Radiosity (HR) [HSA91] with clustering (HRC) [SAG94]. Normally three steps are necessary for Hierarchical Radiosity using clustering:

    - **Refine** In the Refine step the link of the root node is subdivided recursively. A so called *oracle function* decides if a given node is sufficiently subdivided or not.
    - **Gather** The gather step gathers all incoming light from other links and sums up their contribution.
    - **PushPull** In the Push phase the radiosities are added from higher level nodes to lower level nodes according to their hierarchy. In the Pull

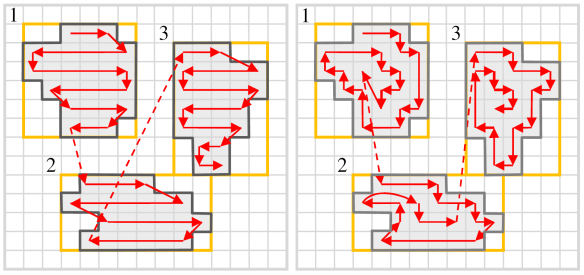


Fig. 2.31: The image shows the Zig-Zag traversal strategy on the left and on the right, the Spiral strategy. (Images courtesy of Ritschel et al. [RGKS08])

phase the area weighted averages are computed from bottom to top. This way the hierarchy gets consistent again.

Since the GPU implementation of the Refine function is non-trivial because their are two tree traversal simultaneously for the sender- and receiver node the refinement is only performed with the sender node. Hence when performing HRC the first step is to compute a cut through the scene. This cut defines, which nodes will be used as sender nodes. The receiver nodes are always the leaf nodes of the tree. The second step performs the gathering according to the cut and the third step performs the push and pull. Since Ritschel et al. [RGKS08] modified the first step, the third step reduces to a pull phase without the push step. Figure 2.32 illustrates the three steps showing one iteration.

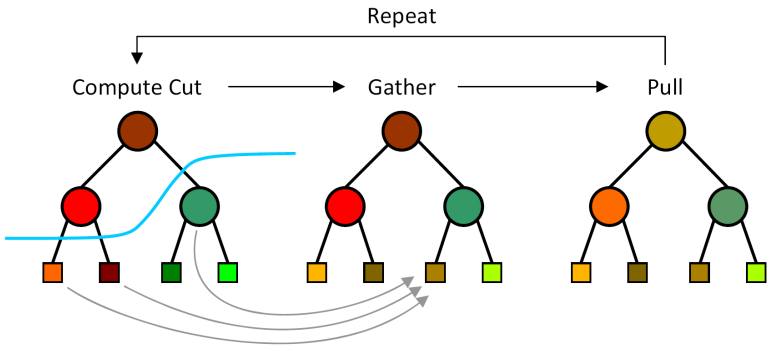


Fig. 2.32: Illustration of the three necessary steps to perform HRC. First a cut through the hierarchy is calculated in the Refine step. Then the gathering is performed. To get a consistent hierarchy a Pull step is done afterwards (Images courtesy of Ritschel et al. [RGKS08])

Global illumination calculation is performed in the texture atlas. To build up a consistent hierarchy they used bounding spheres.

2. **Traversal** To traverse the tree in a GPU friendly way, it was stored as a stackless version, where every node had two pointers. The first pointer points to the left child. The second pointer is called the *skip-pointer*. It points to the next node that has to be used, when the oracle function decides that there is no need for further refinement. Note, the tree is traversed in a depth first order. Since the refinement is only done for sender nodes the hierarchical cut and the gathering step can be performed at once. The tree is traversed downwards until the oracle function decides that no further refinement is necessary for a given sender- and receiver node. Then the contribution of the sender node is added to the receiver node and the process is repeated until all sender nodes contributed to the given receiver node.

3. **Final Gathering** The final gathering step is done after several diffuse bounces were calculated. The current illumination is stored in the texture atlas and then used to calculate the last bounce, which in opposite to the other bounces, also allows glossy surfaces.

   Walter et al. [WFA+05] introduced the so called *Lightcuts*. Their technique is a fast method to perform final gathering. It is based on a hierarchical representation of virtual point lights. The idea is to place cuts through the hierarchy. A cut defines, like it was done in the HRC, which virtual point lights are used for shading. But before shading is performed the error for each VPL inside the cut is calculated. The node with the biggest error is chosen and if it is over a perceiveable threshold for the human eye (2 percent due to Weber's law) it gets replaced by its children. This way a minimal set of VPLs is used to illuminate the scene. Lightcuts reduce the amount of VPLs to use for shading but is not well suited for a GPU implementation. Ritschel et al. [RGKS08] therefore use a slightly modified version of their HRC method. Instead of having a receiver node in the radiosity hierarchy they now have a receiver pixel on the screen and the oracle function decides whether to subdivide or not by calculating an upper bound of the Phong BRDF. This way they can achieve very good visual results while using an adaptively amount of virtual point lights.

Their Coherent Surface Shadow Map method was mainly used with hierarchical radiosity clustering but can be used with any other technique that needs visibility queries like Monte Carlo rendering. Hence they wanted to support very glossy surfaces, they also exploited temporal coherence by jittering the positions of the VPLs slightly to omit single highlights. However, with their approach the images need 20 seconds to converge.

2.6.5 Imperfect Shadow Maps

Imperfect Shadow Maps were introduced by Ritschel et al. [RGK+08] in 2008. They are a novel new approach to generate hundreds of imperfect shadow maps at once every frame. This enables them to use imperfect shadow maps for virtual point light sources. Hence there is no need to cache the shadow maps as proposed by Laine et al. [LSK+07] (see Section 2.6.2) since it is possible to rebuild them for each virtual point light on the fly and enabling us to have completely dynamic scenes.

The basic idea behind imperfect shadow maps is to use a different representation of the scene geometry to enable faster rendering of the shadow maps. This can be achieved by using a point cloud for the whole scene instead of polygons. With this approach the representation of the scene is not perfect, but it is sufficient to produce adequate imperfect shadow maps for indirect illumination computation.

Scene Preprocessing

As mentioned in the Introduction the whole scene is represented by a point cloud. The computation of this point cloud representation is done in a preprocessing step. Ritschel et al. [RGK+08] propose a method, where random triangles are selected with a probability according to their size. In the triangle, they pick a random sample point and store its location in barycentric coordinates. This enables them to have dynamic scenes without recalculating the point cloud. Furthermore it is possible to calculate the normal and reflectance for each sample point using barycentric coordinates.

ISM Creation

Rendering the point cloud The point cloud is used to create the imperfect shadow maps by splatting them into a depth map. Each point is therefore represented by a box splatting kernel and its size depends on the squared distance to the view point, in our case a VPL. To be able to render hundreds of imperfect shadow maps at once they are all written into one big depth map. For example, to render 1024 imperfect shadow maps with a dimension of 128x128 pixels for all the virtual point lights, it is useful to use one large depth map with a dimension of 4096x4096 pixels. During rendering, in the vertex shader each incoming point sample gets assigned to one shadow map in the depth map. Since virtual point lights distribute their light over the whole hemisphere, projective shadow maps are not sufficient and parabolic maps introduced by Brabec et al. [BAS02] are used. Figure 2.33 shows such a depth map that consists of 4 small imperfect shadow maps.

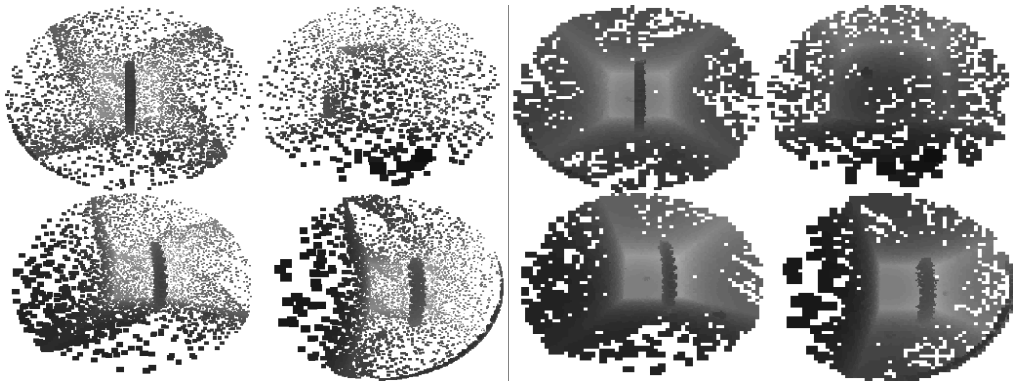


Fig. 2.33: This figure shows the use of the imperfect shadow maps in combination with virtual point lights. The left image shows the four ISMs without any pull/push method (see Section 2.6.5). There are a lot of holes in it. The quality of the ISM can be improved by performing pull/push as proposed by Ritschel et al.[RGK$^+$08] as seen on the left image. Most holes of the ISM are be filled on the right image. Note that the images were made brighter for better illustration.

Improving ISM quality To enable fast calculation of the imperfect shadow maps the amount of sample points is limited. But this introduces holes in the depth map. To overcome this issue, a pull/push approach from Marroquim et al. and Grossman et al. [GD98, MKC08] is used to fill them. This method performs on a data structure similar to an image pyramid. To build it, in the pull phase the image is down-sampled by a factor of two. Only the valid pixels, those who have valid depth values in the finer level, are used to calculate the average depth in the coarser level. In the second step, the push phase, the holes in the finer levels are filled by interpolating the pixels of the coarser level. To get even better results, they used two thresholds to reject outliers in the pull and push phase. In the pull phase only those depth values are averaged that are close to each other and in the push phase only those depth values are replaced, that are far from the lower level depth value. These thresholds get scaled by 2^l according to the mip map level l, where 0 is the fines level. Both thresholds were set to 5% of the scene extent. To keep up high frame rates only the first two levels were calculated [RGK$^+$08]. Figure 2.34 shows a comparison between classic and imperfect shadow maps and the quality improvement by performing the pull-push method on the ISM.

Imperfect shadow maps are a fast way to calculate hundreds of shadow maps on the fly. Although they are not perfect, they can be used for virtual point lights to calculate indirect illumination. The precomputation step creates a point cloud representation of the scene and can be calculated very fast. The ISMs are created by splatting the point samples with a box kernel into the depth map. An additional pull/push method fills the holes in the depth map to improve quality.

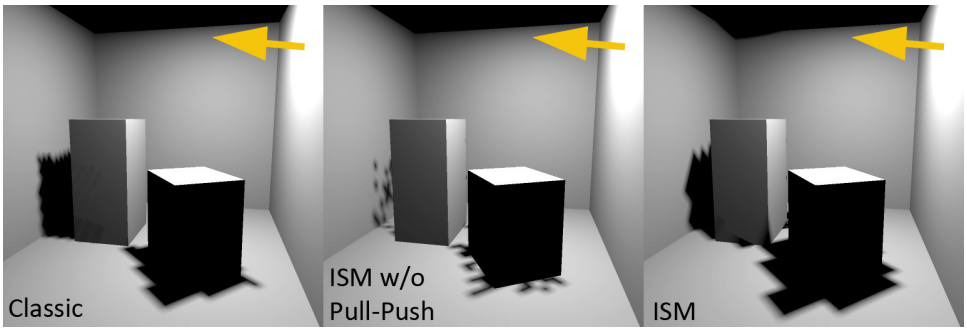


Fig. 2.34: The left image shows classic the result with a classic shadow map. The middle uses am imperfect shadow map, where no pull-push was applied. The right image shows the result after pull-push was performed on the imperfect shadow map. The result is similar to the classic shadow map but can be produces much faster (Image courtesy of Ritschel et al. [RGK+08]).

2.7 Temporal Coherence

Temporal coherence can be described as similarity between consecutive frames. Since the camera or objects only move slightly between two frames, a lot of information rendered in the previous frame keeps the same as in the current one. This coherence between two consecutive frames can be exploited to reduce computation costs per frame by reusing already calculated information.

Temporal coherence has already been exploited in off line ray casting algorithms. Havran et al. [HBS03] was able to increase frame rates up to 47%.

Furthermore Michael Schwärzler [Sch09] introduced a temporal coherence-based method to calculate physically correct soft shadows in real time and Ritschel et al. [RGKS08] used temporal coherence and quasi random VPL positions to avoid artifacts, when rendering images with glossy surfaces (see Section 2.6.4). In this section we will introduce two real-time approaches that both exploit temporal coherence to achieve better performance or results.

2.7.1 The Real-Time Reprojection Cache

The idea of a real time reprojection cache was introduced by Nehab et a.l [NSI06]. It is the basis for several techniques that use temporal coherence, because it uses reprojection to get information from the last frame.

While rendering, a vertex v does not only get transformed with the current world- M_w, view- M_v and projection M_p matrix, it also gets transformed with the matrices of the last frame (M_{wprev}, M_{vprev}, M_{pprev}).

$$v' = M_p * M_v * M_w * v \quad (2.37)$$

$$v'_{prev} = M_{pprev} * M_{vprev} * M_{wprev} * v \quad (2.38)$$

Here, v' and v'_{prev} are the transformed vertices, once for the current frame and once with the transformation of the previous frame. In the pixel shader the transformed coordinates of the last frame must be divided through $(v_{prev})_w$. These coordinates then can be used to get information from the last frame. However, since pixels may appear or disappear through motion of the camera or the objects itself, a test has to be performed if the looked up information is valid or not. This is done by storing the depth buffer of the previous frame, which then gets compared with the calculated depth of the current frame. If the difference exceeds a given threshold then the lookup is set to *cache miss*, otherwise to *cache hit*. Figure 2.35 shows an image and the corresponding cache hits and misses.

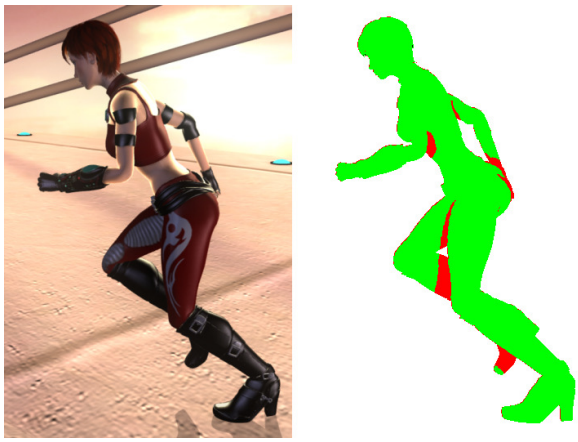


Fig. 2.35: This image shows the rendered scene on the left and the corresponding cache hits (green) and misses (red) on the left. (Image courtesy of Nehab et al. [NSI06]).

Nehab et al. [NSI06] proposed, that for cache hits a cheap shader could be executed, while for misses a more complex one could be used. They observed that hit ratios about 90% are common and furthermore gained speed ups of 30 to 100% when using depth of field or motion blur with stereoscopic rendering.

2.7.2 Pixel-Correct Shadow Maps with Temporal Reprojection and Shadow Test Confidence

Scherzer et al. [SJW07] introduced this shadow mapping technique that makes it possible to have pixel correct shadow maps with only very small overhead costs.

Shadows calculated with the normal shadow mapping method suffer from two types of artifacts. The first one is undersampling since the shadow map normally does not have a high enough resolution. In this case one texel of a shadow map covers several pixels in screen space and spatial aliasing occurs. The second artifacts are the temporal aliasing ones that occur when the light source is moving.

The basic idea is to use a so-called history buffer that accumulates shadow map test results of the previous frames. To have low memory consumption they use so called *exponential smoothing*. The equation to calculate a new exponentially smoothed shadow result looks as followed

$$s_{x,y}(n) = w * f_{x,y}(n) + (1 - w) * s_{x,y}(n - 1) \tag{2.39}$$

where n corresponds to the frame number and $f_{x,y}$ is the shadow map test result for the current fragment at position (x, y). It returns 0 if the fragment is in shadow and 1 otherwise. $s_{x,y}(n - 1)$ is the shadow result of the previous frame. w is a weighting value, that defines how much the new shadow test should influence the overall result. Note that a simple lookup in the previous buffer with the same screen space coordinates is not sufficient, since the camera may has moved. Instead a reprojection into the screen spaced coordinates from the last frame has to be performed. This is similar done like described in Section 2.7.1. The fragments are reprojected into the screen space position of the last frame and then the lookup is performed. Normally these coordinates are not exactly in the center of a texel and therefore filtering of the history buffer is necessary. Scherzer et al. observed that bilinear filtering leads to good results. In Equation 2.39 values will be read and written into the history buffer at once. Since current hardware does not support read and write in the same buffer at once, it must be double buffered.

With the reprojection it may happen that fragments lie outside of the history buffer. Furthermore, due to camera movement it happens that fragments get occluded or disoccluded on the screen. These fragments have to be identified and therefore the history buffer not only contains the smoothed shadow result but also the depth of each fragment. If the difference between the current depth and the depth stored in the history buffer exceeds a given threshold, then only the current shadow test result is written into the history buffer.

Confidence The former described temporal smoothing reduces temporal aliasing. However, it does not reduce spacial aliasing artifacts. Therefore Scherzer et al. combined the weighting value w from Equation 2.39 with the confidence of a lookup in the shadow map. To calculate the confidence for a shadow map lookup, they used the following equation

$$conf_{x,y} = 1 - \max |x - center_x|, |y - center_y|) * 2 \tag{2.40}$$

where (x, y) are the lookup coordinates in the shadow map and $(center_x, center_y)$ are the center coordinates of the nearest texel in the shadow map. If the lookup coordinates are exactly at $(center_x, center_y)$, then the confidence is 1 otherwise it decreases in relation to the maximum distance to the texel center. With this confidence calculation, the shadow converges to a pixel correct shadow. However, it only converges when different rasterizations are used and therefore they perform sub-pixel jittering using Halton sequences. It turned out that simple translational jittering was not sufficient, hence they have also added rotational jittering.

The final weight value w is the confidence applied with a power function.

$$w_{x,y} = conf_{x,y}^{m} \tag{2.41}$$

where m is a parameter that was chosen by Scherzer et al. between 3 to 15. If m is low, then the history buffer confidence is high for the current shadow test result and thus, temporal flickering is visible. If m is high, then the history buffer shows no temporal flickering artifacts but adapts slowly to the exact shadow. Scherzer et al. made m dependent on whether there is camera movement or not. This way, when the camera is moving, m is set to a low value. Then the history buffer adapts the new shadow test results very fast. Furthermore Scherzer et al. observed that temporal aliasing is not visible. When the camera stops moving, m will be increased to reduce temporal aliasing. Figure 2.36 shows the results and improvement of this approach

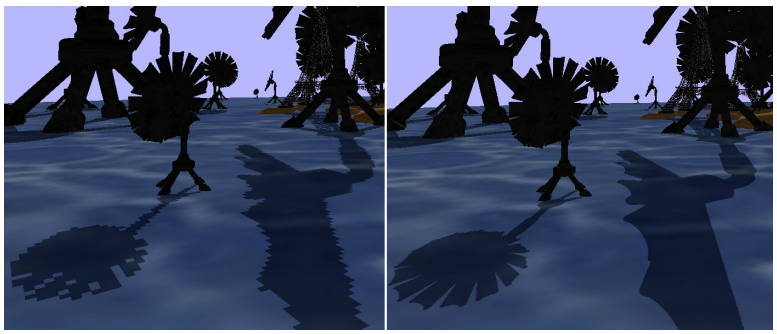


Fig. 2.36: The left image shows a scene rendered with the light space perspective shadow maps method [WSP04]. The right image shows the pixel-correct shadows with the method proposed by Scherzer et al. [SJW07]. (Images courtesy of Scherzer et al. [SJW07]).

The temporal coherence method introduced by Scherzer et al. makes it possible to have pixel correct shadows with only little overhead. The idea to combine the influence weight with the confidence of the shadow map lookup leads to pixel-correct shadow results.

Chapter 3

Real-Time Global Illumination Using Temporal Coherence

Darkness cannot drive out darkness only light can do that. Hate cannot drive out hate only love can do that.

Martin Luther King, Jr.

3.1 Introduction

In Section 2.6.5 we introduced the concept of imperfect shadow maps developed by Ritschel et al. [RGK+08]. With ISMs it is possible to generate hundreds of shadow maps per frame. Thus it is possible to calculate global illumination using instant radiosity (see Section 2.6). The method even allows for arbitrary dynamic scenes, camera and light movement. However, since objects move in the vicinity of the light source, the VPLs will abruptly change their position and orientation. This will lead to unwanted sudden illumination changes, called temporal aliasing.

In this chapter we are going to describe our contribution to the real-time global illumination challenge. We introduce two main extensions to the imperfect shadow maps approach that improve the visual quality and the performance.

The first extension deals with illumination computation from the virtual point lights. When performing global illumination calculation using the instant radiosity approach, visibility and illumination calculation for the VPLs are very time consuming. The problem with expensive visibility calculation was solved by Ritschel et al. [RGK+08] with the introduction of imperfect shadow maps. However, high illumination computation costs are still a challenge and our new method targets this problem, while still maintaining good quality global illumination. The

idea is to reduce the number of virtual point lights which illuminate the scene. However, if we just reduce the amount of VPLs, temporal aliasing will occur and this is certainly not desired. To reach both goals, low illumination computation costs and low temporal aliasing, our idea was to exploit temporal coherence between consecutive frames. This way, we can reuse information from previous frames and furthermore are able to smooth the illumination over time. We get better visual quality and better performance since only a low number of VPLs is used for shading every frame. The concept and methods used to exploit temporal coherence will be described in Section 3.5.

In the instant radiosity approach rays are shot from the main light source into the scene and at their hit points VPLs are created. To get multiple light bounces, new rays starting from the VPLs must be shot recursively. Ritschel et al. [RGK+08] were able to calculate multiple light bounces by extending the imperfect shadow maps to so-called imperfect reflective shadow maps. The problem here is that a huge texture map must be computed first and afterwards only a very small subset of the information is really used. Thus the fill-rate of this method is quite high and our idea was to calculate multiple light bounces without the need to render a complete imperfect reflective shadow map. Our solution to this problem is described in Section 3.6.

To give a better understanding on how all the steps and methods work together, we will give a short description of our rendering framework in Section 3.2. Furthermore we will focus on the generation of appropriate sample sets used for the generation of new VPL sets in Section 3.3. Our framework also supports glossy indirect illumination and we will outline our concept to calculate illumination from a glossy VPL in Section 3.4.

3.2 The Rendering Framework

Our rendering framework is similar to the one from Ritschel et al. [RGK+08]. It is basically a deferred rendering system, i.e. it decouples geometry rendering from shading computation. The benefit is that only visible pixels are shaded (in a 2D post process) and thus, shading is independent of scene complexity. The rendering system makes use of a so called G-Buffer, which stores surface/material, normal and depth information in world space from the point of view of the camera. When rendering the geometry, the G-Buffer is set to be the render target. In the 2D post process the information in it is used to calculate the final fragment color.

With a deferred rendering system the amount of pixels that have to be shaded can be drastically reduced. However, since illumination computation for all virtual point lights is still computationally expensive, so-called interleaved sampling [WKH01] is used (see Section 3.4.1). With interleaved sampling, shading

for a given virtual point light is only performed on a subset of pixels.

To calculate the virtual point light set for the first bounce the framework also maintains a light G-Buffer. This buffer is similar to the main G-Buffer but stores the information rendered from the point of view of the light source. After creating this buffer, the first bounce VPL set is created out of it (see Section 3.3).

After all VPLs were used to illuminate the scene, some merging and filtering operations must be performed due to interleaved sampling. Here our framework slightly differs, because in the merging operation, temporal coherence is exploited to calculate the final indirect illumination.

3.3 Generating Virtual Point Lights

To create a new set of VPLs for the first light bounce, an appropriate sample set has to be generated for sampling the light G-Buffer. Similar to Ritschel et al. [RGK+08], we also use importance sampling to generate a sample set that has a distribution according to an importance buffer stored in the light G-Buffer. They generated a new VPL set in a two pass manner. First they warped the complete initial sample set and afterwards they created the VPL set according to the samples. In our approach we combined importance sampling and VPL set creation to one pass performed on the graphics hardware. The base method used here is called *hierarchical warping* and was introduced by Clarberg et al. [CJAMJ05].

To perform importance sampling, we first create sample points that have uniform distribution. Standard random functions return sample sets that have very high discrepancy and would introduce unwanted artifacts. As an alternative, we have implemented a Halton sample generator [Hal64], which has low discrepancy.

The idea behind hierarchical warping is to warp an existing sample set according to the distribution of the importance buffer. This is done in a hierarchical manner and alternating in horizontal and vertical order. Figure 3.1 shows the process of warping an initial sample set. First the sample set is divided according to the upper and lower summed importance/probability. Then the sample set is vertically warped, so that both areas have the same size again. Afterwards two horizontal warps are performed, one on the upper and one on the lower half of the quad. After the warp, the sample set has a distribution that matches the importance of the importance map. The quad can be split into four sub quads and the warping steps can be repeated in each of it recursively.

Our single pass solution creates a new VPL set as follows: We first create a vertex buffer that contains a Halton sequence of sample points. The sample points are in the interval from -1 to 1. The render targets are one-dimensional textures that can store a set of VPLs (see Section 4.4.1 for more details). Each sample has a unique id, corresponding to one fragment in the render target data

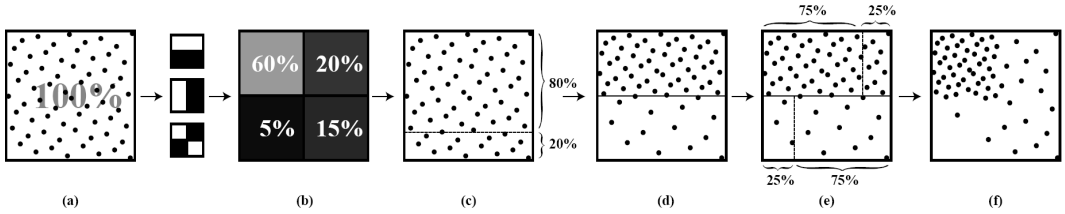


Fig. 3.1: These images show the process of hierarchical warping for one iteration. (a) shows an initial sample set. (b) is an importance distribution that can be derived from according mipmap levels of the importance map. In (c) the sample set is divided into two areas. The size of the areas correspond to the vertical importance distribution. The upper summed importance distribution is 80%, the lower 20%. Figure (d) shows the first warp of the sample set. Now, about 80% of the sample points are in the upper half and about 20% in the lower half of the quad. The first part was to do a vertical warp, the second step is to do two horizontal warps on the upper and lower half of the quad. According to the importance distribution, the area is divided into two areas (e). After the horizontal warps the sample set shows a distribution (f) that corresponds to the importance map. (Images courtesy of Clarberg et al. [CJAMJ05])

textures. Instead of redistributing the complete sample set at once, we perform the warping steps consecutively on a single sample point. In Shader Model version 3.0 recursions are not allowed, however, the proposed method can be implemented without any recursion. We start at a coarse mipmap level, with importance map dimension (2×2) and read out the four importance values. The vertical and horizontal borders in the first iteration step are set to be between $[-1, 1]$. We then perform vertical and horizontal warping on the sample point. Depending on the initial position of the sample point and the importance values, it now lies in one of the four quads and the border intervals are adapted accordingly to the quad. For example, if the sample point is in the bottom left quad, the vertical and horizontal borders would be set to $[-1, 0]$. In the second iteration step we change the mipmap level to the next finer one (4×4). But this time we only read out the four importance values from the bottom left quad and again do the vertical and horizontal warping steps with respect to the border intervals. These steps are repeated until we reach the finest mipmap level of the importance map.

After the warping steps of the current sample, its position is used to lookup the surface information in the light G-Buffer. From this information a new VPL will be created.

3.4 Virtual Point Light Shading

In this section we will explain our method to illuminate the scene with virtual point lights. Similar to Ritschel et al. [RGK+08] or Laine et al. [LSK+07], we use interleaved sampling (discussed in Section 3.4.1) to reduce shading costs. Furthermore we are able to calculate illumination for all VPLs at once using only one single render call (see Section 3.4.2). Finally we introduce our method to calculate glossy indirect illumination in Section 3.4.3.

3.4.1 Interleaved Sampling

In the instant radiosity method (see Section 2.6), there are two computationally expensive parts. The first one is to query the visibility for virtual point lights, and the second one is to shade each screen pixel with the VPLs. The first problem, fast visibility queries for the VPLs, was solved by Ritschel et al. [RGK+08] with the introduction of so-called imperfect shadow maps (see Section 2.6.5). Expensive shading costs can be avoided, or at least reduced, with a method called *interleaved sampling*, introduced by Keller and Heidrich [WKH01].

Interleaved sampling is a very good method to reduce shading costs. The idea behind it is to only shade a subset of all screen pixels with a given virtual point light. This is done by splitting the camera G-Buffer into $n \times n$ tiles. Each tile of this split G-Buffer shows the complete scene, but each tile is a subset of the camera G-Buffer pixels, with a different offset. A pixel in the split G-Buffer with indices (a, b) in tile (i, j) corresponds to the pixel $(an + i, bn + j)$. Figure 3.2 shows the original camera G-Buffer on the left and the resulting split G-Buffer on the right side.

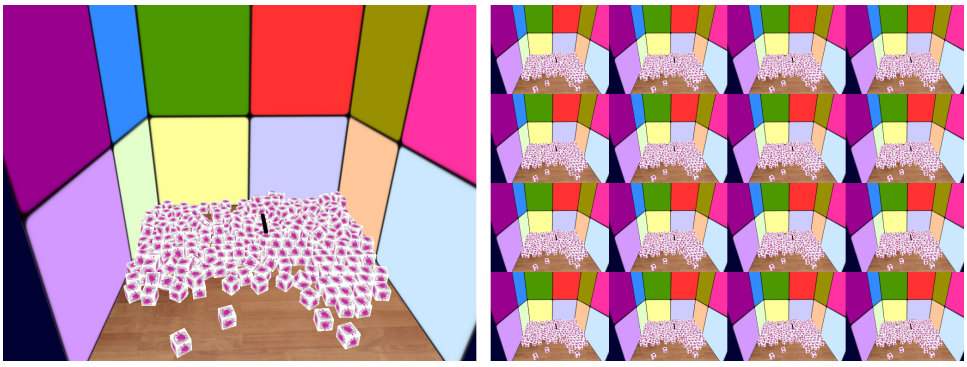


Fig. 3.2: Left image shows the color channel of the camera G-Buffer. On the right side, the split G-Buffer.

Each VPL gets assigned to one of the tiles in the split G-Buffer and shading is

then performed only on the pixels of one tile (see Section 3.4.2).

After all shading computations are finished, an accumulation buffer, which stores the indirect illumination of all VPLs, has to be merged. In this merging step we exploit temporal coherence. We look up the illumination and surface information from the last frame and calculate the final indirect illumination value accordingly (see Section 3.5).

After this step, the results must be filtered to smooth out the different illumination values from the tiles. However, since the geometry of the scene introduces discontinuities in the buffer, the filter has to be geometry aware. This prevents that values for filtering are taken into account that actually do not correspond to the same surface element. The filtering is done with a box filter of the size $(n \times n)$. While Segovia et al. [SIMP06] proposed to compute a discontinuity buffer, Laine et al. [LSK+07] calculated discontinuity on the fly, while filtering. We also follow this approach and set the thresholds for a maximum depth difference and maximum normal difference to $\alpha = 0.01$ (depth threshold) and $\beta = 0.8$ (threshold for the minimal dot product of the two normal vectors). For more details on geometry aware filtering we refer to the work of Laine et al. [LSK+07].

3.4.2 Tiled Mesh Geometry

In our deferred rendering system the illumination for all pixels is calculated in a 2D post process. However, to compute illumination from all VPLs at once, a single screen space quad is not adequate. We need an appropriate mesh geometry that is adapted to the tiles in the split G-Buffer and furthermore contains information about the VPL that should be used for shading. For every VPL, a single quad with the size of a tile is created. These single quads are placed at the assigned tile position in screen space. The vertices for a single quad store the screen space position mapped to texture lookup coordinates and also the id of the assigned VPL. This way, the vertex shader is able to read out the needed information from the virtual point light data structures (see Section 4.4). Usually it happens, that several VPLs are assigned to one tile. Then we just create several quads at the according tile position. Note that we have to disable the depth buffer to get correct results. All these quads are stored in one so-called *tiled mesh geometry*. This way we can perform the complete illumination computation for all VPLs with one single render call.

3.4.3 Glossy Surfaces

Ritschel et al. [RGK+08] pointed out, that they were able to handle diffuse and glossy indirect illumination. However they do not state wether they also support multiple glossy light bounces.

We wanted to support global illumination for diffuse and glossy surfaces even with multiple glossy light bounces. Therefore the G-Buffer always stores specular intensity and specular power for each surface point. At shading time, glossy light bounces are quite easy to calculate. There is a light source, the virtual point light, a camera and a surface point to shade. Figure 3.3 illustrates this setup. However, a VPL only stores the diffuse light that it emits. The emitted specular part into direction of the surface point has to be calculated related to the previous light source. Therefore, we have to calculate the outgoing light from a VPL with respect to the surface parameters at the point where the VPL is placed. The important parameter here is the reflection vector (shown in red), because it is dependent on the incident light direction. The incident light comes from the main light source if the VPL created simulates the first light bounce. In this case we can store the light source position in a uniform shader variable. However, when we have multiple light bounces, all other VPLs need an additional lookup to get the position of the VPL from the previous bounce that illuminates it.

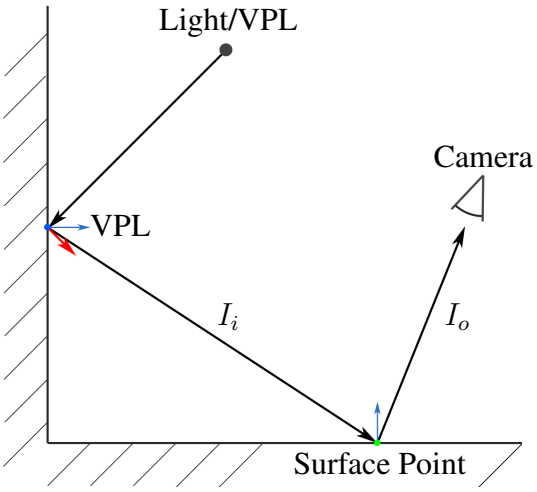


Fig. 3.3: This figure illustrates our method to calculate glossy indirect illumination.

The same effect also appears when a set of new virtual point lights is calculated. Here, the illumination of a new VPL also depends on the current set of VPLs and its incident light direction. In the first case, this is again the main light source, and in all other cases, the previous virtual point lights. This additional texture lookup may lead to problems when vertex animations are used that need several texture lookups in the vertex shader because there are only four texture stages available in Shader Model 3.0. However, we have implemented a simple mesh animation that uses one extra texture lookup and did not notice any problems.

After shading is performed with all virtual point lights, the accumulation buffer looks as shown in Figure 3.4

Fig. 3.4: This figure shows the accumulation buffer after shading with all VPLs. To get the final illumination the accumulation buffer must be merged, so that it does not contain any tiles anymore. Afterwards a geometry aware box filter is used to smooth indirect illumination over $n \times n$ pixels.

3.5 Temporal Coherence

Temporal coherence may be described as *similarity between consecutive frames*. This similarity allows information from previous frames to be reused in the new frame and thus, computation costs to be reduced (see Section 2.7).

The problem with indirect illumination computation is that many VPLs are needed to get visually pleasing results. This leads to very high illumination computation costs although we are already using interleaved sampling. However, if we just reduce the number of VPLs, temporal aliasing will occur and the visual quality will suffer. Our idea is to use temporal coherence to target these issues. First we are able to reduce shading costs because we reuse information from the

last frame and second we reduce temporal aliasing, because indirect illumination gets smoothed over time.

However, since we have a deferred rendering system, we cannot just perform a reprojection while rendering the geometry because we need the information in the later 2D post process. In Section 3.5.1, we therefore introduce the movement buffer that allows us to efficiently perform the reprojection. Section 3.5.2 will explain how we calculate a so-called *confidence value* that defines how confident information from the last frame is.

3.5.1 The Movement Buffer

In the original reprojection cache paper [NSI06], the reprojection was performed by multiplying a vertex in the vertex shader with the current matrices and the matrices from the last frame. In the pixel shader the screen space coordinates of the last frame were available and could be used to gather information from the previous frame. However, in our deferred rendering system we must cache these reprojected coordinates, because they are needed later and not during geometry rendering.

A solution to this problem is a so-called *movement* buffer, which stores the position difference from the previous to the current rendered frame on a per screen-pixel basis. We calculate the movement of each pixel by using reprojection as proposed by Nehab et al. [NSI06] and perform the transformation once with the current matrices and once with the ones from the last frame. In screen space we then subtract the positions from each other and write $(\triangle x, \triangle y)$ into the movement buffer. We want to have the difference and not the absolute position because later this information can be used for confidence calculation (see Section 3.5.2).

Since the movement of each pixel is compared in screen space, we can use the movement buffer like a distortion buffer to get the right lookup coordinates. To get the illumination I_{prev}, depth $depth_{prev}$ and normals n_{prev} from the previous frame, we just have to look up the movement buffer m_{buf} at the position of the current screen-pixel (x_s, y_s). The returned values from the movement buffer are the lookup offset values $(\triangle x, \triangle y)$. The indirect illumination from the previous frame can be easily found at $I_{pref}(x_s + \triangle x, y_s + \triangle y)$. The depth and normals are looked up in the same way. However, having the old indirect illumination, depth and normals is only half of the work. We have to calculate how confident the indirect illumination value from the previous frame really is. This will be explained in the next section.

3.5.2 Confidence Computation

We can get the previous indirect illumination per screen-pixel with the previously described movement buffer. However, since the objects are moving, and the illumination and the camera may change, we have to calculate how confident an indirect illumination value from the previous frame is. We therefore take three different parameters into account.

1. The relative position change from one frame to the other. For performance reasons we divide the position change in two parts. The difference in depth per screen-pixel and the screen-space difference in the movement buffer.

2. The difference between the pixel's normal. If the normal changes, illumination will also change and hence, the previous values are not as confident as the new ones. We calculate the difference using the dot product between the normals.

3. Global illumination is a global process. So it may happen that a pixel does not move (no change in position and normal) but the confidence should still be low because, due to other moving objects, the illumination changes a lot. Therefore the difference of the previously calculated illumination and the current illumination also reduces the confidence. We use an exponential growing of the difference, so that low differences have low impact and higher differences more.

The confidence for each pixel is calculated as followed:

$$\begin{aligned}
\triangle pos &= ||(x_s - x_{prev}, y_s - y_{prev}, d_s - d_{prev}) w_{pos}|| && (3.1)\\
\triangle normal &= (1 - (n \cdot n_{prev})) w_{normal} && (3.2)\\
\triangle illumination &= saturate(||I_{new} - I_{prev}||^3) w_{illumination} && (3.3)\\
confidence &= c_{Base} saturate[1 - && (3.4)\\
&\quad \max(\triangle pos, \triangle normal, \triangle illumination)] && (3.5)
\end{aligned}$$

where $[*]_{prev}$ always directs to values from the previous frame, (x_s, y_s) is the screen position, d_s the screen depth, n the normal of the screen pixel and c_{Base} is the base confidence. I_{new} is the indirect illumination calculated in this frame. The normals are double-buffered in the G-Buffer similar to the front and back buffer. The depth from the previous frame is stored in the fourth channel of the old indirect illumination buffer. The final indirect illumination for the current frame is:

$$I = I_{prev} confidence + (1 - confidence) I_{new} \quad (3.6)$$

In Figure 3.5 we show the confidence for an arbitrary scene. We have to note here that we do not store the confidence in a dedicated buffer but calculate it on the fly, when the split G-Buffer gets merged again (see Section 3.4.1).

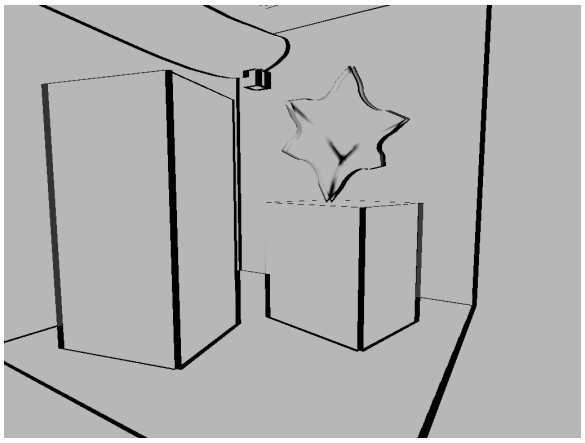


Fig. 3.5: Image illustrating the confidence for a given arbitrary scene. The brighter the pixels, the more confident a previous pixel is. Note how the confidence is low around sharp corners.

3.6 Multiple Light Bounces

In this section we are going to describe how multiple light bounces can be performed. We first show the method proposed by Ritschel et al. [RGK+08] and then introduce our new approach.

In the original imperfect shadow map method, Ritschel et al. [RGK+08] performed multiple light bounces by extending the ISMs to *imperfect reflective shadow maps* (IRSM) as it was done for standard shadow maps by Dachsbacher et al. [DS05]. In Section 2.6.3 we presented a method that made also use of reflective shadow maps. IRSMs store the position, normal and an illumination value and they are created similar to the ISMs. So the point cloud representation of the scene is also used to get high frame rates and the illumination depends on the corresponding VPL. Figure 3.6 shows the illumination of an IRSM. Ritschel et al. [RGK+08] then performed importance sampling (see Section 3.3) on the IRSM to get new VPLs. This method works well to calculate new sets of virtual point lights. However, as importance sampling is used to find interesting VPL positions, a lot of information is rendered into the IRSM but never used. Therefore it would be more interesting to have a method that needs less fill rate. The next section will introduce our new multiple bounce algorithm that is able to calculate new VPL sets faster than the previous method.

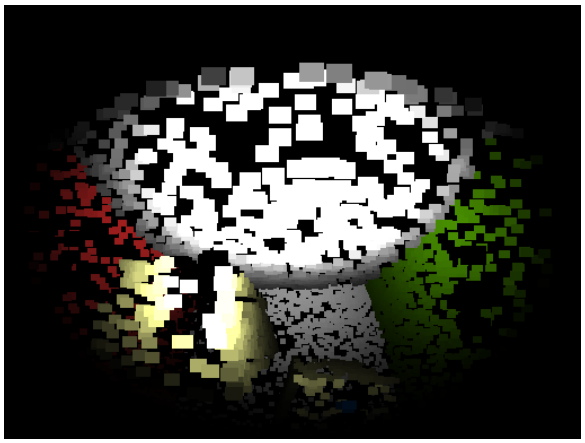


Fig. 3.6: Image shows the illumination map of the imperfect reflective shadow map.

3.6.1 Algorithm outline

Similar to Ritschel et al. [RGK+08] we also use the point cloud representation as a possible set of new VPL locations. Our idea is to assign each sample point to an existing VPL. Then for all assigned sample points we want to find the one that a) receives the most illumination from the assigned VPL and b) actually is visible to the assigned VPL.

To find the new VPL set we only need one single render pass, compared to the two pass method proposed by Ritschel et al. [RGK+08]. Therefore our render targets are one-dimensional floating point textures that can store a new VPL set. Each VPL set has a position, a normal and a color map and each fragment of these maps corresponds to a new VPL with a given id (see Section 4.4 for more details).

Figure 3.7 gives an overview of the algorithm. In the first step, each sample point is assigned to one VPL. The next step is to calculate the illumination from the assigned VPL. Like proposed in Section 3.4.3 we calculate a glossy light bounce. The specular intensity of a new VPL is calculated from the specular intensity of the surface at the sample point position multiplied with the specular intensity of the assigned VPL. This is necessary to get correct specular light bounces.

After illumination computation we have to find out if the sample point is visible to the existing VPL or not. This can be done with a simple shadow test with the ISM from the assigned VPL. The result of the shadow test is multiplied with the illumination. Thus the illumination is zero if the point sample is not visible to the assigned VPL.

In the last step we use the depth buffer to find the most important sample point. This is simply done by writing out a depth value for each sample point that is re-

lated to the computed illumination from the assigned VPL. The compare function of the depth buffer must be switched to $\geq$ and the final VPL set will contain only those new VPLs that have the highest contribution to the scene. However, this method to select new VPLs is not the best because weaker VPLs never contribute to the illumination. It would be better to select them according to a probability that depends on their illumination.

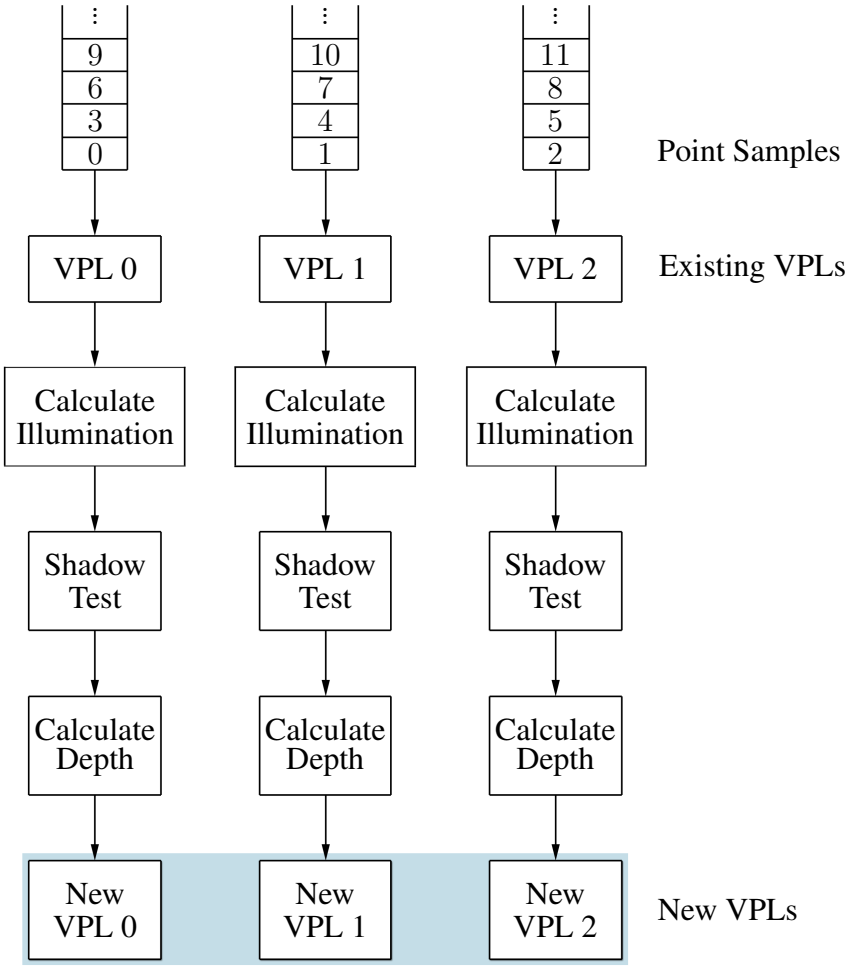


Fig. 3.7: This figure illustrates our method to calculate a new set of VPLs.

Note that with this approach we always get one new VPL for an existing one. In the approach from Ritschel et al. this is not the case.. Furthermore we are able to take specularity of the material into account, allowing for specular light bounces (see Section 3.4.3).

3.7 Summary

In this chapter we have introduced our new real-time global illumination method using temporal coherence. It reuses the information from the last frame to calculate the new illumination based on a so-called confidence value. Furthermore we have introduced a new method to calculate multiple glossy light bounces that reduces fill rate and therefore computes multiple light bounces faster than the method developed by Ritschel et al. [RGK+08].

Chapter 4

Implementation

No matter how fast light travels it finds the darkness has always got there first, and is waiting for it.

Terry Pratchett

4.1 Introduction

In this chapter we go into more details on the implementation. As we implemented several rendering methods, different light sources and sample generators, the software architecture had to be as flexible as possible. We first want to outline the needed steps for global illumination calculation using a renderer that supports multiple bounces. Figure 4.1 shows a screen shot of our implementation.

1. **Create G-Buffer** The G-Buffer will be explained in further detail in Section 4.2. It is basically a buffer that contains additional information per screen-pixel.

2. **Create G-Buffer of light** For the light source we also create a G-Buffer. Note that the light G-Buffer is slightly different to the screen space G-Buffer (see Section 4.3 for more details).

3. **Create samples** The so-called sampler computes all new VPLs at once for a given sample set. Therefore, we have to setup appropriate samples first. We use a Halton sequence generator for good sample distribution.

4. **Create VPLs** The sampler alters the samples accordingly and calculates new VPLs using the previously generated light G-Buffer.

5. **Create imperfect shadow maps** For the created VPLs we have to create imperfect shadow maps, so that the indirect illumination can be calculated.

6. **Split G-Buffer** To reduce shading cost, not every pixel gets shaded by each VPL. The screen space G-Buffer gets split into $n \times n$ tiles (see Section 3.4.1).

7. **Accumulate indirect illumination** As we now have the split G-Buffer, the VPL set and its corresponding imperfect shadow maps, we can calculate first bounce indirect illumination in an accumulation buffer.

8. **Calculate new VPL set** If the renderer should support multiple light bounces, a new set of VPLs has to be created. Section 3.6 introduced our new method and outlined the suggested method by Ritschel et al. [RGK$^+$08].

9. **Create imperfect shadow map** For the new VPL set new imperfect shadow maps have to be calculated

10. **Accumulate indirect illumination** The illumination from the new VPLs is additively written into the accumulation buffer. Steps 8 to 10 are repeated for further light bounces.

11. **Merge indirect illumination** The results in the accumulation buffer are still divided into $n \times n$ tiles and have to be merged again. Afterwards we have one big indirect illumination buffer. Note that in this step the illumination from the previous frame is taken into account. While merging, the confidence is calculated and the last frame's illumination as well as the current illumination stored in the accumulation buffer are combined (see Section 3.5).

12. **Filter indirect illumination** As only a subset of VPLs was used for a fragment, the indirect illumination buffer has to be filtered, to get smooth indirect illumination.

13. **Calculate direct illumination** In this step we perform standard illumination according to the type of light source in a 2D post process.

14. **Perform Tonemapping** In the last step we sum up direct and indirect illumination. To take the dynamic range of the display into account we perform tone mapping with the method proposed by Reinhard et al. [RSSF02].

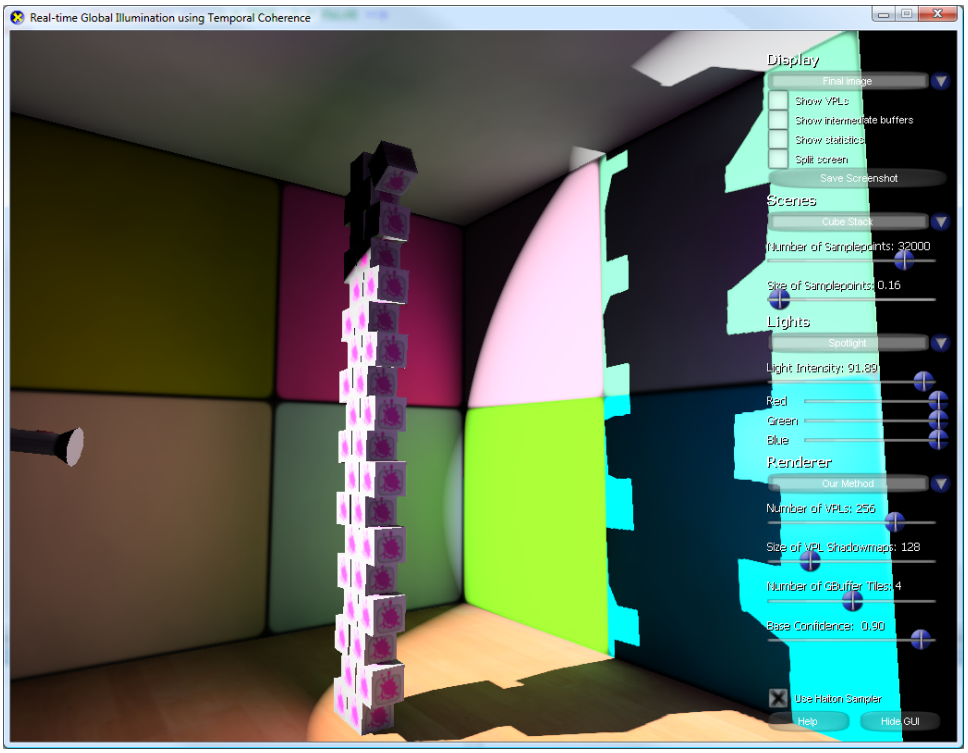


Fig. 4.1: Screenshot of our application.

4.2 Camera G-Buffer

The G-Buffer has to store enough information to allow for fast shading calculation, but at the same time should be as small as possible, because then the memory usage and memory bandwidth will be small. Since we have to transform the data into several different coordinate frames, the best basis is the world space coordinate system. In our case we set up the G-Buffer with four render targets where each of these has 32 bits:

1. **Render target 0** The render target with id zero stores the red R, green G and blue B color information in 8 bits per channel. The remaining 8 bits store the diffuse intensity DI of the surface. Render target setup: (R, G, B, DI)

2. **Render target 1** This render target is a little bit more complex. It stores the normal n of each fragment, the specular intensity SI as well as the specular power SP. To get everything to fit into 32 bits we have to "compress" the normal representation. As normals should always have a length of 1, we can calculate one coordinate from the other two. However, this is only true if we store the normals in screen space, as we know that the sign of the third

axis always has to be positive or negative (depends on whether we use left handed or right handed coordinate systems). As we store the G-Buffer data in world space, this assumption is not true anymore and the sign of the third axis has to be stored somehow. For this reason we multiply the specular intensity SI with ± 0.5 depending on the sign of n_z and shift the result by 0.5. Thus, if $n_z \geq 0$ the value stored in the render target will be in the interval from $0.5 \leq SI \leq 1.0$, otherwise $0.5 > SI \geq 0.0$. The calculation looks as follows:

$$SI_{modified} = SI(step(0, n_z) - 0.5) + 0.5 \tag{4.1}$$

As we have only unsigned numbers, n_x and n_y also have to be mapped between 0 and 1. The render target finally looks as: $(0.5n_x + 0.5, 0.5n_y + 0.5, SI_{modified}, SP)$

3. **Render target 2** In the third render target the linear depth is stored in all 32 bits. This way, linear depth values are still accurate enough. Linear depth gets calculated as follows:

$$depth_{linear} = \frac{(M_{proj} * M_{view} * M_{world} * p)_z}{z_{Far} - z_{Near}} \tag{4.2}$$

where p is a vertex, M_{proj} is the projection matrix and M_{view} the view matrix of the camera. M_{world} corresponds to the world matrix of the current object. z_{Far} and z_{Near} is the distance to the far- and respectively the near plane of the camera. The depth values are in the interval from zero to one and stored as one single float component in the render target: $(depth_{linear})$.

4. **Render target 3** The fourth render target stores the movement of a pixel with respect to the previous frame. This buffer is the before mentioned movement buffer m_{buf} from Section 3.5.1 and it is the only buffer that works in screen space. The render target looks as follows: $(x_s - x_{prev}, y_s - y_{prev})$, where (x_s, y_s) are the current screen space coordinates and (x_{prev}, y_{prev}) are the screen spaced coordinates from the previous frame.

4.3 Light G-Buffer

As a sampler will create new VPLs from the light G-Buffer, we have to add more data to the shadow map. The light G-Buffer is similar to the reflective shadow maps introduced by Dachsbacher and Stamminger [DS05] but does not contain emitted flux per shadow map texel. The light G-Buffer has to store enough data so that new virtual point lights can be setup up from it. As the camera's G-Buffer, it contains four render targets with a size of 32 bits:

1. **Render target 0** This render target stores the surface color and its diffuse intensity (R, G, B, DI). However, as VPLs should be created out of this data, the color information is heavily blurred. Thus, when we render the light G-Buffer we set the mipmap level to a predefined value.

2. **Render target 1** As in the camera G-Buffer the normals are compressed. See Equation 4.1 for the calculation of $SI_{modified}$. The render target is set up as $(0.5n_x + 0.5, 0.5n_y + 0.5, SI_{modified}, SP)$.

3. **Render target 2** This render target stores the linear depth from the point of view of the light source $(depth_{linear})$ (see Equation 4.2).

4. **Render target 3** The fourth render target stores a so called *importance* value per texel. The importance value defines the probability that a VPL should be placed at this texel. Thus VPLs are set more likely at texels with high importance. The importance of a texel is calculated as follows

$$importance = lightIntensity(surfaceColor \cdot lightColor)(0.5 + SP) \tag{4.3}$$

   where $lightIntensity$ is a value that corresponds to the amount of light that arrives at this fragment. For a spot light source, it depends on the size of the cone and the distance. $surfaceColor$ is the mipmapped color of the surface and $lightColor$ is the color of the emitted light. As a VPL that is placed on a surface with high specular power will produce high frequency illumination, it is a good idea to place a lot of VPLs on glossy surfaces. Therefore Dachsbacher and Stamminger in [DS06] suggested to multiply the importance value with the specular power SP. However, a completly diffuse surface would have zero importance and therefore we biased it with 0.5.

4.4 Light G-Buffer Sampling

A sampler object uses the light G-Buffer to find new positions for virtual point lights. As a sampler has to work different for a point light or a spot light, it is tightly coupled to the currently active light source. To be more precise, each instant radiosity renderer instance receives the sampler object from the light source instance. Each sampler generates a complete data set for new VPLs, includeing position, normal, average surface color and surface properties. Note that the sampler can only generate new VPLs at points that are visible to the main light source. For multiple light bounces see Section 3.6.

4.4.1 Virtual Point Light Data Structure

Virtual point lights are placed on surface points in the scene. They have to imitate the local material properties, including diffuse and specular properties. Thus the data structures necessary to represent a VPL set are stored in three one dimensional floating point texture maps where each of the maps has a size of 128 bits.

1. **position** Simply stores the position of the VPL, the fourth value is not used. Map setup: $(x, y, z, -)$.

2. **normalSISP** The setup of this buffer is the same as for the G-Buffers in Section 4.2, except that the normal values n_x and n_y are not mapped to be between 0 and 1 anymore. Map setup: $(n_x, n_y, SI_{modified}, SP)$

3. **colorDI** In this map we store the emittance color (R_e, G_e, B_e) of the VPL. This value depends on the heavily blurred surface color and the incoming light of the light source. The diffuse intensity is simply copied from the material properties of the surface. Map setup: (R_e, G_e, B_e, DI).

4.4.2 Spotlight Sampler

A spotlight normally has a sampler that is able to handle rays that are in the frustum of the spot. So normally rays are projected with the spot light's projection matrix to get the lookup coordinates in the light G-Buffer. But since we want to perform importance sampling, this is not necessary anymore. In Section 3.3 we described how hierarchical warping developed by [CJAMJ05] is working. In this approach, all the samples are already in the projected screen space, because the importance map is already in projected screen space. So the importance sampler for a spot light first alters a sample in screen space according to the importance map and then looks up the data in the light G-Buffer.

4.4.3 Pointlight Sampler

In contrast to the spotlight sampler, the pointlight sampler has to process rays into arbitrary directions. We have not implemented any importance sampling functionality into this type of sampler. In this case, the light G-Buffer consists of cube maps that are used to generate new VPLs.

4.5 Virtual Point Light Shading

All the shading computation itself is done entirely on the graphics hardware. Shading computation is shared within the vertex- and the pixel shader. We performed as much computation as possible in the vertex shader to increase performance. Furthermore the computation costs in the pixel shader were reduced with efficient data layout. The following enumeration will give an outline of the necessary steps to shade a tile with an assigned virtual point light source.

1. **Additional data** One of the first steps in the vertex shader is to calculate the screen space position of the quad corners. This way we know which screen space position each pixel had before the G-Buffer was split. Additionally the column and row ids are also stored for further usage.

2. **VPL color** Then the illumination color of the virtual point light is looked up, according to the VPL id stored in the tiled mesh geometry. The diffuse intensity of the VPL is also stored for later use.

3. **VPL matrix** Next, the coordinate system of the virtual point light will be created. Therefore the VPL data texture maps $position$ and $normalSISP$ are used. Later we also need the specular intensity and specular power, so we store them already at VPL matrix creation time.

4. **Calculate pixel position** When shading the G-Buffer with a given VPL we have two important spaces: the world space and the space of the VPL. The world space is used to calculate information that deals with the main light source and the camera positions to handle glossy surfaces (see Section 3.4.3) and the shading itself is performed in VPL space. At this point, the reason why we store the depth as a linear value becomes obvious. We can use the inverted view-projection matrix to calculate the near and far plane positions at a defined screen spaced pixel. The actual world position of this pixel is then a linear blend between the far plane and the near plane position:

$$p_{far} = \begin{pmatrix} x_s \\ y_s \\ 1 \\ 1 \end{pmatrix} \cdot M_{VP}^{-1} \tag{4.4}$$

$$p_{near} = \begin{pmatrix} x_s \\ y_s \\ 0 \\ 1 \end{pmatrix} \cdot M_{VP}^{-1} \tag{4.5}$$

$$p_{world} = (1 - depth)p_{near} + (depth)p_{far} \tag{4.6}$$

where $depth$ is the depth value stored in the G-Buffer. Furthermore we can multiply the inverted view-projection matrix with the virtual point light matrix and then also use linear interpolation to calculate the position p_{VPL} with respect to the VPL coordinate system. As the depth changes from pixel to pixel in the G-Buffer, the linear blend is performed in the pixel shader.

5. **Reflection vector** To support glossy light bounces, we calculate a reflection vector that defines the direction in which the VPL sends most of the light. Note that depending on the number of light bounces, the incident light direction, used for the calculation, either depends on the main light source or on the previous light bounce VPL.

6. **Depth lookup** The first step in the pixel shader is to lookup the current depth value. Then the before mentioned positions in world and VPL space are calculated.

7. **Visibility query** To find out if the current pixel is visible to the virtual point light, a shadow test using the ISM must be performed. We first do a parabolic transformation of the VPL space position p_{VPL} according to the upper hemisphere of the VPL and afterwards map the lookup coordinates to the corresponding tile in the ISM. Then we can compare the looked up distance with the z-component of p_{VPL}. Depending on the result the current pixel will be visible to the VPL or not.

8. **Additional surface information** If the current pixel is visible, the color, normal, diffuse intensity, specular intensity and specular power are looked up from the split G-Buffer.

9. **Geometry factor** Depending on the relative position of the current pixel to the VPL used for shading, a geometry factor η will be calculated as follows:

$$\eta = \frac{\max(0, \cos\Theta_s)\max(0, \cos\Theta_r)}{\max(1, ||d||^2)} \tag{4.7}$$

where Θ_s is the angle between the normal of the VPL and the direction vector to the current pixels position. Θ_r is the angle between the normal of the surface of the current pixel and the incident direction vector. d is the distance between the VPL and the surface point. Note, at this point, we bias the global illumination computation, because we clamp the minimum distance to 1 (see Section 2.6.1).

10. **Incoming light** The incoming light from the VPL depends on the diffuse intensity DI_{VPL}, the specular power SP_{VPL} and the specular intensity SI_{VPL}.

Note that the specular intensity of the VPL depends on the surface point where the VPL is located, multiplied with the amount of specular light that reaches the VPL from the main light source. This is important when multiple light bounces are calculated. Then the specular intensity gets recalculated every time a new VPL set is calculated (see Section 3.6). The equation for the incoming light at point p looks as followed

$$I_i = \eta[I_{VPL}DI_{VPL} + c_{Light}SI_{VPL}\max(0, ref \cdot dir)^{SP_{VPL}}] \tag{4.8}$$

where I_{VPL} is the illumination color of the VPL. ref is the reflection vector calculated in the vertex shader and dir is the direction vector from the VPL to the surface point p. c_{Light} is the color of the main light source.

11. **Outgoing light** The amount of outgoing light at point p in the direction of the camera is the last step that has to be performed. It looks similar to the incoming light equation.

$$I_o = I_i[c_pDI_p + SI_p\max(0, n_p \cdot halfVec)^{SP_p}] \tag{4.9}$$

where DI_p, SI_p, SP_p are the surface parameters. c_p is the color, n_p is the normal at point p and $halfVec$ is the normalized sum of the incoming and outgoing light direction.

4.6 Final Composition

When the shading with the virtual point lights is finished, the accumulation buffer gets merged and filtered as previously described. The last step that has to be performed afterwards is to combine direct and indirect light and calculate the final result using a tonemapper developed by Reinhard et al. [RSSF02]. Figure 4.2 shows the final result after composition and tone mapping. Note that the scene was captured at a different time step than Figure 3.4.

4.7 Summary

In this chapter we went into details of our implementation. We have outlined the main steps to get real-time global illumination using temporal coherence. Furthermore we introduced the necessary steps to compute illumination from a given VPL and explained the used buffers in our system. Finally we outlined, how the composition of direct and indirect illumination is performed.

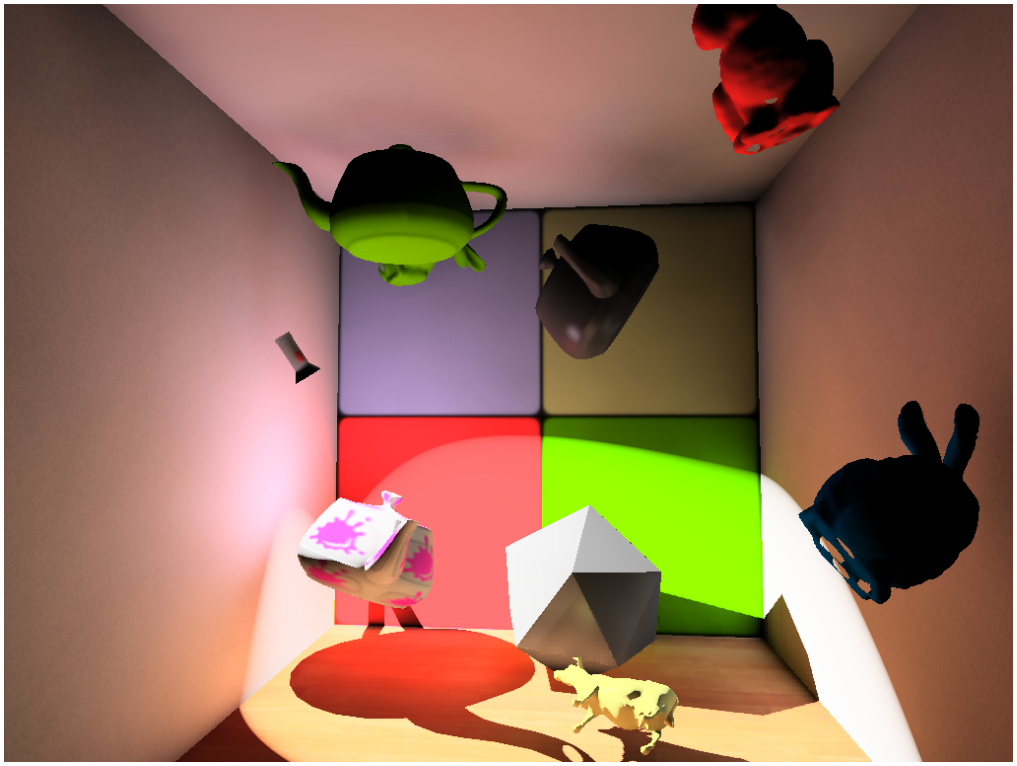


Fig. 4.2: This figure shows the final results after composition of indirect with direct illumination and the tonemapper. It was rendered using our method with 256 virtual point lights and one light bounce at 73 fps.

Chapter 5

Results

It is impossible to travel faster than the speed of light, and certainly not desirable, as one's hat keeps blowing off.

Woody Allen

5.1 Introduction

The new approach was implemented and tested using our own software framework developed by Wolfgang Knecht and myself. It uses DirectX 9c as rendering API and PhysX for physics simulation. The test device was an Intel Quad Core i7 920 running at 2.66GHz with 6GB of RAM and two NVIDIA GeForce 295GTX in SLI mode with 1768MB of video memory. The operating system was a Windows Vista 64-Bit. To compare our results with other methods we implemented instant radiosity using standard shadow maps and also the imperfect shadow maps method developed by Ritschel et al. [RGK+08].

Since we need several texture buffers, the memory consumption with our method is pretty high. On the other hand we have to mention that for multiple light bounces our new algorithm needs far less memory than the method proposed by Ritschel et al. However, similar to the ISM approach we have quite large data buffers in our system, namely the G-Buffer, the split G-Buffer and the light G-Buffer. Furthermore we have the large imperfect shadow map that has at least two mip map levels. To perform pull/push on the ISM we must double buffer the ISM. Additionally we have buffers for merging and filtering. The VPL sets also need memory on the graphics hardware, but since we have a maximum of 1024 VPLs per set the memory consumption is low. Depending on the number of sample points in the scene, there are additional vertex buffers that store all the necessary information (position, normal, uv's, sample id).

The final visual result and the frame rates heavily depend on the selected parameters. Therefore it is normally best to select appropriate values manually. In this thesis we have not focused on heuristics to find optimal parameter values, however we have implemented the possibility to support weights for sampling the objects. Since the sample points are distributed over all triangles with respect to their size, we simply multiply the size of each triangle with the assigned weight. Note that we assign weights on a per-object basis. For example, the Sibenik Cathedral has a higher weight on the pillars and therefore more sample points are placed on them. The possibility to assign weights gives an artist the freedom to control the behavior of indirect illumination and focusing sampling on visually important parts.

The following results were all rendered with a screen resolution of 1024x768 pixels and unless otherwise mentioned with 32k sample points. We will first investigate the behavior of our method compared to the approach proposed by Ritschel et al. [RGK+08] with respect to temporal coherence. Then the results with multiple light bounces are presented.

5.2 Temporal Coherence

In this section different parameter setups will be used to show how the visual quality and performance of the algorithm changes. The used scene is a simple Cornell Box with four objects in it. As there is an object with a higher polygon count than the cubes have, the total number of triangles is 3,372 and the number of vertices is 10,116. Figure 5.1 shows a sequence of images of the Cornell Box rendered with 4, 16, 64, 256 and 1024 virtual point lights. The base confidence for temporal coherence is set to 0.9 and each single imperfect shadow map has a resolution of 128×128.

The performance behavior with a different number of virtual point lights is shown in Table 5.1 and Table 5.2 shows the behavior with different resolutions for the imperfect shadow maps. Due to the temporal coherence our method is always slightly slower than the one proposed by Ritschel et al. [RGK+08]. However, the visual quality of our method is much better.

# VPLs	*fps*	*time (ms)*	*our method - fps*	*our method - time (ms)*
4	233	4,3	169	5,9
16	159	6,3	151	6,6
64	134	7,5	125	8,0
256	81	12,3	73	13,7
1024	30	33,3	27	37,0

Tab. 5.1: This table shows the influence of the number of VPLs that are placed in the scene. The frames per second and times in the first two columns belong to our implementation of the imperfect shadow maps approach by Ritschel et al. [RGK$^+$08]. The last two columns show the results of our new temporal coherence rendering system. The frame rates are lower because of the overhead due to temporal coherence, but the visual results are better with the same amount of VPLs.

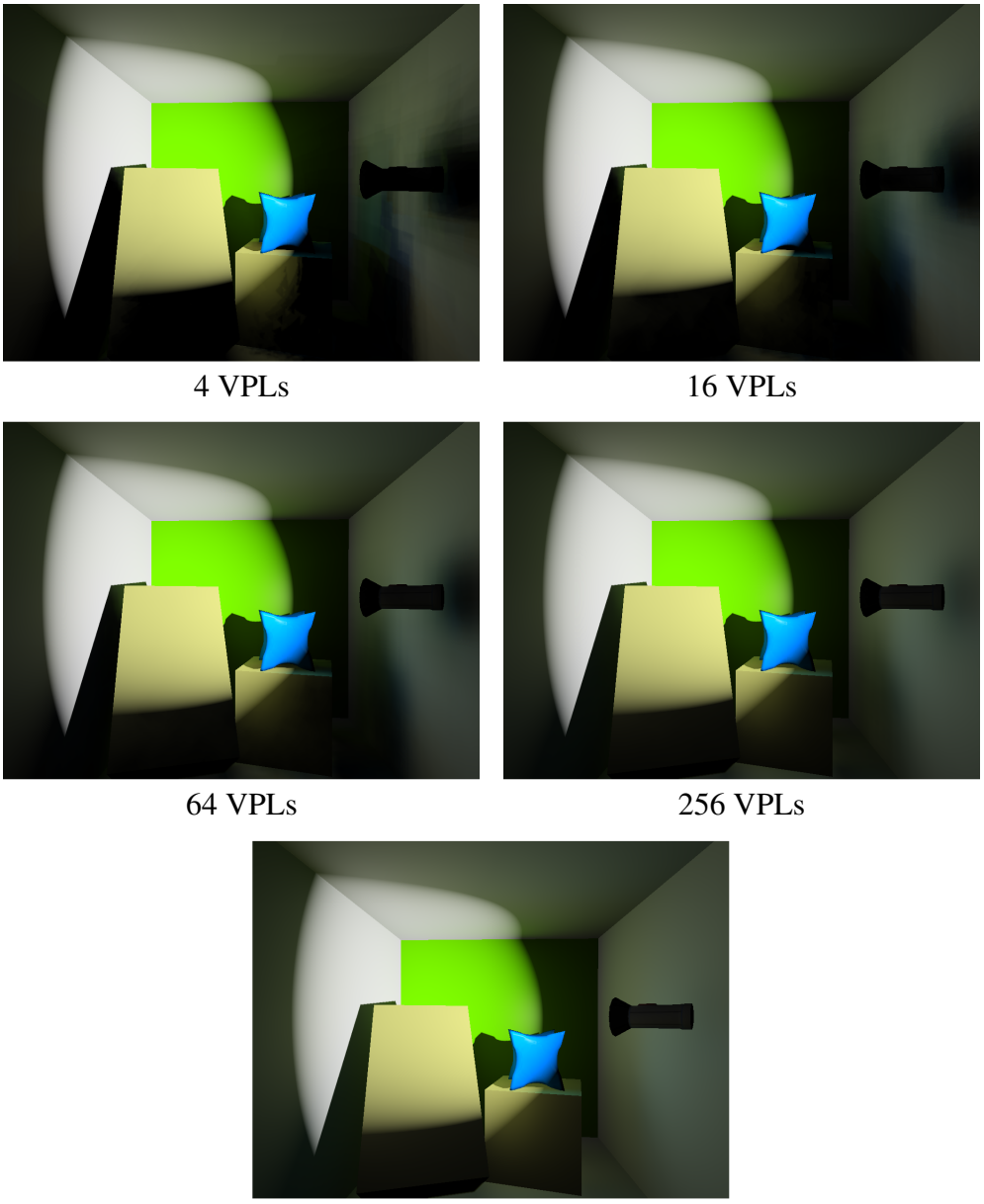


Fig. 5.1: This figure shows the Cornell Box rendered with a different numbers of virtual point lights. The visual artifacts are visible on the right wall. They decrease the more VPLs are used. Note that with 1024 VPLs there are nearly no indirect shadows anymore. This comes from a too low number of sample points. Furthermore, due to resampling of the VPL positions the images suffer from temporal artifacts when using less than 256 VPLs.

ISM Res.	*fps*	*time (ms)*	*our method - fps*	*our method - time (ms)*
64	90	11,1	79	12,7
128	81	12,3	73	13,7
256	57	17,5	54	18,5
512	2,3	429,2	2,2	450,5

Tab. 5.2: This table shows the influence of different imperfect shadow map resolutions. The frames per second and times in the first two columns belong to our implementation of the imperfect shadow maps approach by Ritschel et al. [RGK$^+$08]. The last two columns show the results of our new temporal coherence rendering system.

The average overhead due to temporal coherence is approximately $1.51ms$. This value was calculated by averaging the difference of the rendering times with 4, 16, 64, 256 and 1024 VPLs. The overhead mainly comes from rendering the movement buffer and the confidence calculation.

Our measurements furthermore showed that the time needed for rendering linearly depends on the number of virtual point lights used. However, this linearity is not true for different resolutions of the imperfect shadow maps. There is a huge performance impact when we have a setup with 256 VPLs and an ISM resolution of 512×512 pixels. Actually this is not really surprising as we then have a large texture map containing all ISMs with a resolution of 8192×8192 pixels.

Figure 5.2 shows a comparison of the two methods (our method is shown on the right column), once with 64 (upper row) and once with 256 VPLs (lower row). All the other parameters are kept equal.

Figure 5.3 shows a comparison of the imperfect shadow maps approach and our new method to a reference image, rendered with 1024 VPLs using standard shadow maps.

Note how the images in the left column suffer from hard edges/artifacts whereas the images in the right column are much smoother. Furthermore temporal aliasing artifacts occur, when the scene is dynamic. In this scene the stack of cubes collapses and the VPLs will change their position from one frame to the other. In our approach we change every VPL position each frame, but because of the temporal smoothing dynamic scenes do not suffer from aliasing artifacts as much. However, when using only 64 VPLs some temporal aliasing is also visible with our approach. In this test scene, our method with 256 virtual point lights gives the visually most pleasing results. It has no sharp edges and renders temporally smooth frames without any aliasing artifacts.

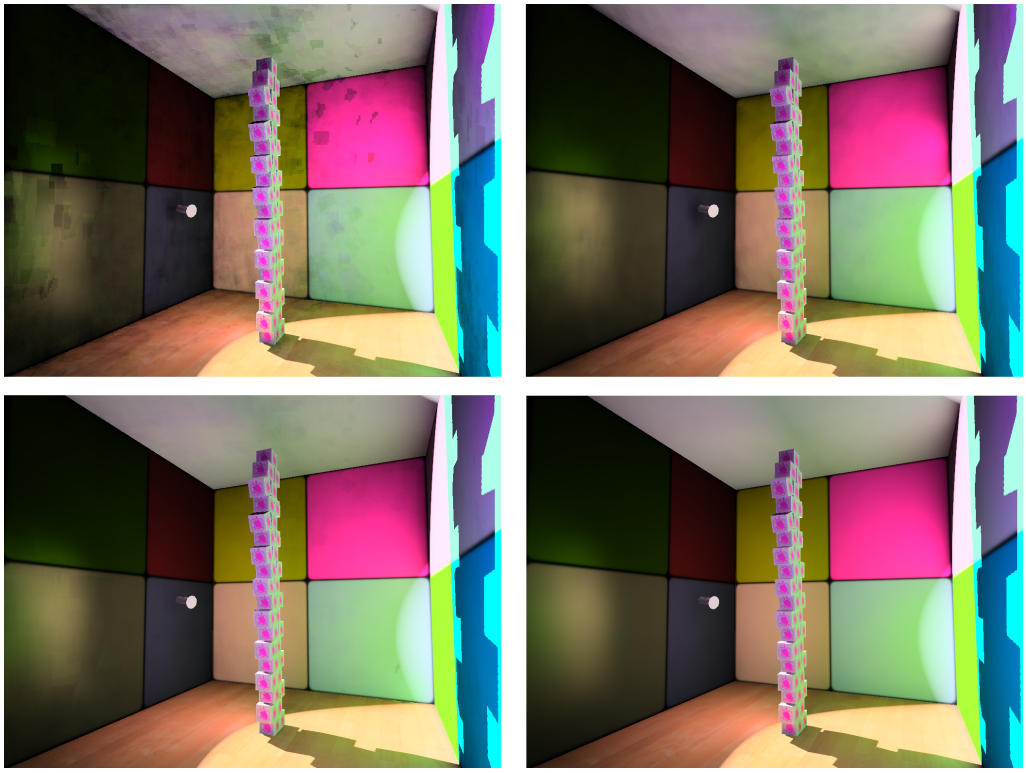


Fig. 5.2: This figure shows a comparison to the method proposed by Ritschel et al. shown on the left column. Our method is shown in the right column. The images in the top row are rendered using only 64 VPLs and therefore suffer from visible artifacts. The lower row is rendered using 256 VPLs. Here the images are visually more pleasing. However, when the cube stack collapses our method keeps up smooth rendering without flickering artifacts, while the other method does not.

Method by Ritschel et al. rendered in 11,7ms

Our new method rendered in 13,9ms

Reference image rendered with standard shadow maps in 6,25sec

Fig. 5.3: This figure shows a comparison between the imperfect shadow maps method, our new method and a reference rendering, using standard shadow maps. For the reference rendering we have used 1024 virtual light sources, whereas we used 256 VPLs for the other two methods. In the method proposed by Ritschel et al. [RGK$^+$08] the aliasing artifacts are clearly visible. In our method they are not visible due to the temporal smoothing over time.

5.3 Multiple Light Bounces

This section gives a performance analysis of our multiple light bounce method compared to the imperfect reflective shadow maps approach. Table 5.3 shows the measured results with different numbers of light bounces.

# bounces	*calc. time (ms)*	*our method-calc. time (ms)*	speed up
2	3,54	0,39	9,2
3	10,69	0,79	13,5
4	14,08	1,23	11,5
5	18,56	1,45	12,8
6	22,02	2,17	10,1

Tab. 5.3: This table shows the behavior of our multiple bounce calculation method compared to the imperfect reflective shadow maps approach. The measured timings include only the calculation time for the light bounces. ISM creation and shading was disabled for the measurements. In average our method actually performs 11 times faster than the other one. Note that we start with two light bounces, as the first bounce is calculated with the light G-Buffer sampler. Furthermore note that the timing results are rounded and that the speed up calculation is based on the accurate timings.

The more light bounces made, the faster our method is compared to the IRSM approach. Figure 5.4 visualizes, how more than one light bounce may enhance the image.

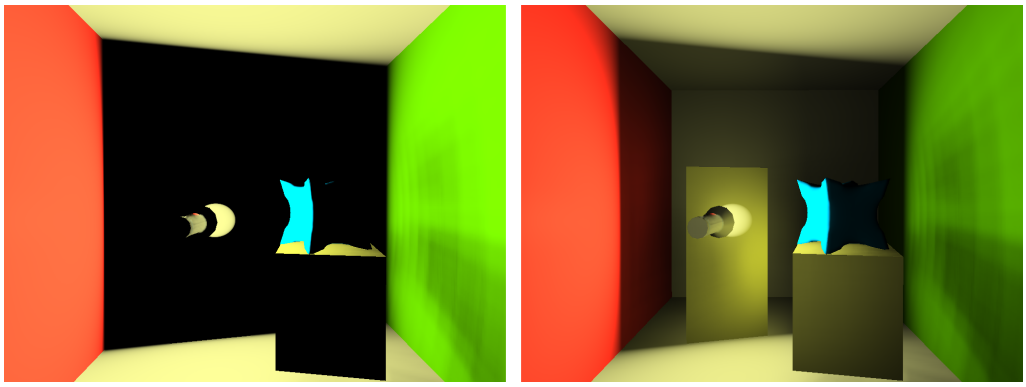


Fig. 5.4: The image on the left is rendered using our method with only one light bounce. As all the VPLs are placed on the small illuminated spot, the rest of the Cornell Box is not visible. One additional light bounce is already sufficient to illuminate the whole scene.

To get correct timings we disabled shading and ISM creation for multiple light bounces. We then recorded the rendering times for 1 to 6 light bounces and subtracted the time needed for rendering without any multiple bounces. We got the

final speed up factors by dividing the IRSM timings through the timings of our method. It turned out that our method is more than 11 times faster then the imperfect reflective shadow map approach. While the time needed with the IRSM method is approximately 4.52 milliseconds for one light bounce, our method is able to calculate a new set of VPLs in 0.40 milliseconds.

Note that our new temporal coherence method combined with the new multiple bounce method outperforms the previous method in visual quality as well as in rendering speed. Our method renders the Cornell Box with 256 VPLs and two light bounces at 46 frames per second, whereas the method proposed by Ritschel et al. has 43 frames per second.

5.4 Limitations

Our approach is able to handle fully dynamic scenes including moving light, camera, objects, deformable meshes and dynamic materials. The only thing that we have to pay attention to, is that the number of texture lookups in the vertex shader is limited in DirectX 9c to four lookups. Therefore, deformable geometries may run into this limit for the ISM shaders.

Furthermore temporal coherence needs the vertex position of the previous frame, which means, that every geometry animation needs to be performed twice, to get the right lookup offsets in the movement buffer. Here the previously mentioned texture lookup could also be a limitation.

With the implementation of temporal coherence we were successfully able to reduce temporal flickering and therefore enhancing the visual quality of the images. However, temporal coherence also introduces temporal artifacts that could be described with the word *afterglowing*. This normally happens when objects are moving very fast or the overall light in the scene suddenly decreases due to occlusion of the light source. We have developed a small stress test scene where the objects and the light source are moving very fast. Figure 5.5 shows a screenshot from this test. The artifacts are clearly visible, however they are less disturbing than temporal flickering.

Our presented method to calculate multiple light bounces is more than 11 times faster than the method using imperfect reflective shadow maps. Because Ritschel et al. use importance sampling to find new VPL sets, it is also possible that two or more VPLs have the same source VPL, while our method only produces one new VPL for one source VPL. This may lead to a bad distribution, because weaker VPLs have no chance to contribute to the illumination. However, with simple modifications this would also be possible.

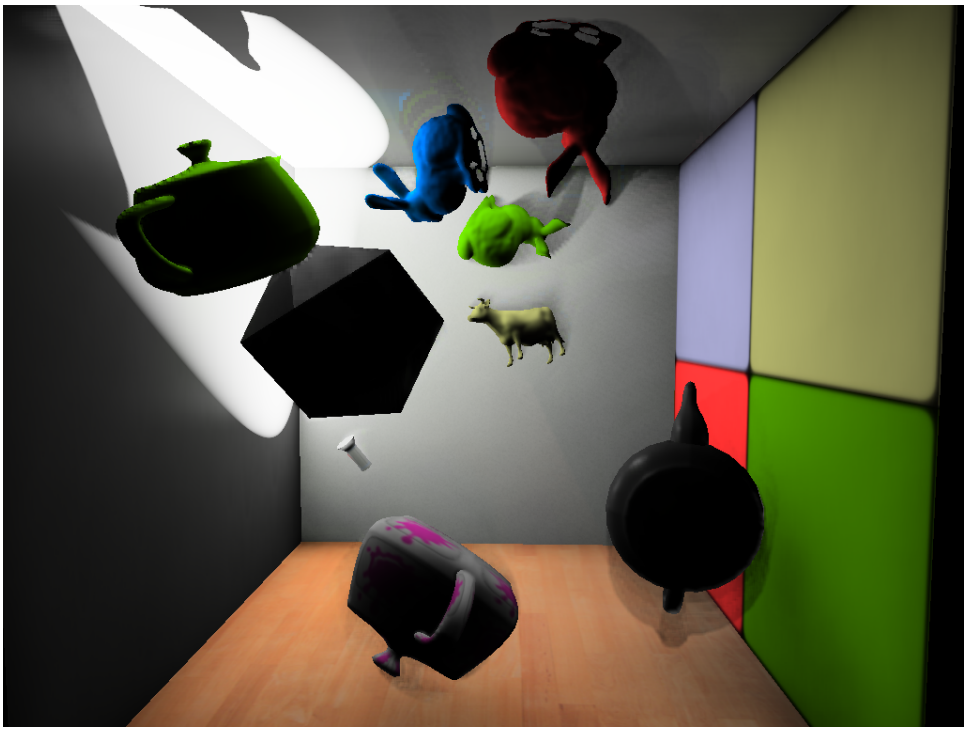


Fig. 5.5: This figure shows a stress test scene where the main light source and the objects are moving very fast. Here the temporal artifacts are clearly visible. However, they are less disturbing than temporal flickering, because they change more smoothly.

5.5 Summary

In this chapter we have compared our results with the method developed by Ritschel et al. [RGK+08] using several different parameter setups. The tests showed that our method generates visually pleasing results with less required light sources, and provides good quality global illumination in real time.

Chapter 6

Conclusion

Always remember, sir, that light and shadow never stand still.

Benjamin West

This thesis gave a detailed overview on the field of global illumination in real-time environments. Chapter 1 was an introduction to the questions: "*What is global illumination?*" and "*Why is global illumination so important?*". Global illumination describes how light propagates in the environment. Light, emitted by a light source, gets reflected at surface points several times until it finally hits the eye. This process introduces a lot of features that allow the human visual system to extract an impressive amount of information about our environment. Therefore the more realistic the computer generated images are, the more information can be extracted from them.

In Chapter 2 we gave an overview on the state of the art in real-time global illumination and temporal coherence. We furthermore introduced the two methods *Instant Radiosity* (2.6) and *Imperfect Shadow Maps* (2.6.5) on which this thesis is based on. In the instant radiosity approach, rays are shot from the light sources into the scene. The idea is to place so-called virtual point lights at hit points of those rays. For further light bounces, rays, starting from an existing virtual point light, can be shot recursively to place new virtual point lights. When shading the surface, all the virtual point lights will contribute to the final illumination. However, instant radiosity introduces two time-consuming tasks. First we have to check, if a surface point is visible to the virtual point light and second we have to calculate the illumination for all virtual point lights. This is the point where imperfect shadow maps become important. Imperfect shadow maps are an approximation to standard shadow maps. However they can be created much faster compared to standard shadow maps. It is actually possible to create hundreds of imperfect shadow maps every frame. Thus arbitrary dynamic scenes can be supported. The idea behind this method is to use a point cloud representation of the

scene. These points are then splatted into the shadow map and some additional methods are used to improve the quality of the imperfect shadow map.

Our new method (see Figure 6.1) introduced in Chapter 3 targets at the other time-consuming task: Virtual point light shading. The main idea is to exploit temporal coherence between consecutive frames and reuse as much information from the previous frame as possible. This way, it is possible to reduce the time needed for shading as less virtual point lights are placed every frame. However, since we wanted to support arbitrary dynamic scenes, it is important, that the confidence of the information from the last frame is calculated and weighted appropriately. In our approach we use three different indicators, that reduce the confidence of the information from the last frame. The first one is the position change, the second one the normal change of the surface and the third one the illumination difference between the newly calculated illumination and the illumination of the old frame. The first two indicators are pretty obvious, because if the position or the normal of a surface point change, illumination will also change. The third indicator is used to reduce temporal smoothing artifacts. Objects moving in the vicinity of the light source may cause a sudden change in the overall illumination of the scene. The first two indicators may not change but the illumination will probably change a lot and therefore the confidence of the previous frame should be low, so that the illumination adapts quickly to the new lighting situation.

We furthermore developed a new method to calculate multiple light bounces that also supports glossy surfaces. The idea is to use the point samples as possible virtual point light positions and to assign each sample point to an already existing virtual point light. For all point samples that are assigned to a virtual point light we then want to find the sample that receives the most illumination from the assigned virtual point light and is actually also visible to it. Our method creates a new set of virtual point lights directly in one pass. In contrast the method proposed by Ritschel et al. [RGK+08], which needs two passes. One to create the imperfect reflective shadow map and a second one to perform importance sampling on it. However, since we do not need to create a complete imperfect shadow map, we are able to reduce fill-rate drastically and thus get better performance.

In Chapter 4 we outlined details on our implementation and finally in Chapter 5 we compared our method with the method proposed by Ritschel et al. [RGK+08]. Our method is slightly slower due to the overhead of temporal coherence calculation, when the same number of virtual point lights is used. However, it is worth the extra costs, since it improves the overall image quality. Although our method suffers from temporal smoothing artifacts, they are visually less disturbing than the temporal aliasing artifacts that occur with the other method.

Our novel method to calculate multiple light bounces reduces fill-rate drastically and therefore works up to 11 times faster than the approach using imperfect reflective shadow maps. When we combine our temporal coherence method with

the new multiple light bounce method, we are actually able to outperform the previous method in visual quality and rendering performance.

Finally, our tests showed that the introduced methods are able to provide visually pleasing global illumination, while maintaining high frame rates.

6.1 Future Work

The presented new method works very well, however, there is still a lot of work that could be done to improve the overall quality of global illumination computation. One thing is to develop more appropriate sampling strategies for the scenes. Up to now it is possible to give different weights on objects that influence, how much sample points are assigned to an object. Maybe there are ways to perform this step automatically. Furthermore sample points are located on random positions on the triangle. This is also a point where quasi random approaches could lead to better results, as the ISM would have a better coverage.

Dachsbacher et al. [DS06] splatted a sphere geometry onto the G-Buffer to reduce the shading costs, as only those pixels were calculated, that lied inside an isosurface of influence by a VPL. We have not implemented this method yet, but it could increase the rendering speed a lot, especially when very glossy objects are in the scene.

The introduced method for multiple light bounces is faster, but at the same time only the fittest VPLs will be in the set. This leads to a non optimal distribution and therefore some randomness should be introduced to this method. This could be done by adding some random offsets to the depth values that are used to find the most important VPL.

The confidence calculation is based on three parameters. However, maybe there are better parameters or different calculation modes, that give better results. One idea would be to also temporally smooth a confidence buffer in a way that areas that had a huge change in past frames, will still have low confidence for a couple of frames.

Fig. 6.1: This figure shows an image, rendered with our method at 73 frames per second.

List of Figures

1.1 Image on the left is rendered with local illumination. Image on the right is rendered using a global illumination approach. Note the color bleeding and caustics as well as the more natural look of the image. (Image courtesy of Guerrero [Gue07]) 7
1.2 Image rendered using global illumination. Some possible light paths are shown with its corresponding light transport notation. (Image courtesy of Guerrero [Gue07]) 8
1.3 Graphical illustration of the rendering equation. 10
1.4 Graphical illustration of the different material types. The red ray illustrates ω_i, while the green ray shows ω_o. From left to right: diffuse, glossy specular, perfect specular and retro-reflective surfaces. 12

2.1 The images illustrate the light map technique. A texture gets multiplied by a light map to get the final illuminated surface. 15
2.2 Head rendered using precomputed radiance transfer with self-shadowing and self-interreflection. (Image courtesy of Sloan et al. [SKS02]) . 16
2.3 Result of the visibility function V_p for a given point p. (Image courtesy of Guerrero [Gue07]) 17
2.4 This figure shows the first four bands of the SH basis functions. (Images courtesy of Guerrero [Gue07]) 18
2.5 This figure shows the three different basis function types. (Images courtesy of Guerrero [Gue07]) 19
2.6 Reconstruction of three spherical functions with increasing order of approximation. (Images courtesy of Green [Gre03]) 20
2.7 Figure showing the process of outgoing radiance calculation for objects with diffuse BRDFs (Images courtesy of Sloan et al. [SKS02]) 21
2.8 The left image uses only environment lighting without any ambient occlusion. Adding AO (middle) enhances realism but now looks partly too dark. In the image on the right side indirect illumination is also taken into account, which increases realism even more. (Images courtesy of Bunnell [Bun05]) 24

2.9 For every vertex a surface element is calculated. (Images courtesy of Bunnell [Bun05]) 24
2.10 Illustration of radiance transfer between two surface elements. (Image courtesy of Bunnell [Bun05]) 25
2.11 Left: Occlusion calculation is correct. Middle: Occlusion calculation gets too dark. Right: After the second pass, darkening is reduced. (Images courtesy of Bunnell [Bun05]) 26
2.12 In the top row, the scene is shown with a varying number of indirect light bounces. From left to right: zero, one and two indirect bounces. The bottom row shows the corresponding indirect illumination only. (Images courtesy of Bunnell [Bun05]) 27
2.13 Artifacts due to a coarse mesh. (Image courtesy of Kautz et al. [KLA04]) 27
2.14 Images rendered with the SSDO method. The scene and the lighting can be completely dynamic. (Images courtesy of Ritschel et al. [RGS09]) 28
2.15 The left graphic illustrates directional occlusion and the right indirect light calculation. (Images courtesy of Ritschel et al. [RGS09]) 29
2.16 Oriental room scene rendered with Antiradiance and implicit visibility. (Image courtesy of Dachsbacher et al. [DSDD07]) 30
2.17 The illustrations show the different formulations of the rendering equation. (a) Shows the traditional rendering equation using operator **G** and **K**. (b) Shows the new formulation using **U**, **K** and **J** operators. (Images courtesy of Dachsbacher et al. [DSDD07]) . . 31
2.18 (a) First, the light paths from the light sources are created. Each light bounce is stored as a VPL. (b) Then, the illumination for a given surface point is calculated by taking all the Virtual Point Lights and the directional light into account. (Images courtesy of Segovia [Seg07]) 34
2.19 Image rendered using the incremental instant radiosity method introduced by Laine et al. (Image courtesy of Laine et al. [LSK+07]) 35
2.20 (a) Rays are shot from the primary light source. At the hit points virtual point lights are placed. (b) The scene is rendered by using the VPLs to illuminate all the visible surface points. (c) For each VPL a visibility test is performed to delete invalid VPLs (Images courtesy of Laine et al. [LSK+07]) 36

2.21 These images show the different domains for the light sources. a) Shows the unit disc with the projected VPL directions. In this 2D space the redistribution and creation/deletion of VPLs is performed and afterwards backprojected to the hemisphere as seen in figure b). Figure c) shows the Delauney Triangulation on the unit sphere for an omnidirectional point light source. (Images courtesy of Laine et al. [LSK+07]) 37
2.22 Image on the left shows the unfiltered combined accumulation buffer. On the right the final accumulation buffer after the geometry aware box filter was applied. (Images courtesy of Laine et al. [LSK+07]) 40
2.23 Image on the left shows the splatted geometry for VPLs placed on the rings surface. Image on the right shows the resulting image with fine caustic effects. (Images courtesy of Dachsbacher and Stamminger [DS06]) 41
2.24 The illustration shows emission for diffuse (left) and glossy VPLs (right) (Images courtesy of Dachsbacher and Stamminger [DS06]) 42
2.25 Illustration that shows the bounding geometry for a given VPL (diffuse and glossy).(Images courtesy of Dachsbacher and Stamminger [DS06]) 43
2.26 Images rendered with coherent surface shadow maps. (Images courtesy of Ritschel et al. [RGKS08]) 43
2.27 Left: Standford dragon rendered from multiple views. Right: List of coherent depth maps. (Images courtesy of Ritschel et al. [Rit07]) 44
2.28 Illustration shows the depth values of a shadow map with depth values between the front and the back faces of an object. (Image courtesy of Weiskopf and Ertl [WE03]) 45
2.29 Depth value z_{avg} which lies between z_1 and z_2 is sufficient to get correct visibility results. (Images courtesy of Ritschel et al. [Rit07]) 45
2.30 Texture atlas with several charts from a Cornell Box scene. (Images courtesy of Ritschel et al. [RGKS08]) 47
2.31 The image shows the Zig-Zag traversal strategy on the left and on the right, the Spiral strategy. (Images courtesy of Ritschel et al. [RGKS08]) 48
2.32 Illustration of the three necessary steps to perform HRC. First a cut through the hierarchy is calculated in the Refine step. Then the gathering is performed. To get a consistent hierarchy a Pull step is done afterwards (Images courtesy of Ritschel et al. [RGKS08]) . 48

2.33 This figure shows the use of the imperfect shadow maps in combination with virtual point lights. The left image shows the four ISMs without any pull/push method (see Section 2.6.5). There are a lot of holes in it. The quality of the ISM can be improved by performing pull/push as proposed by Ritschel et al.[RGK+08] as seen on the left image. Most holes of the ISM are be filled on the right image. Note that the images were made brighter for better illustration. 51
2.34 The left image shows classic the result with a classic shadow map. The middle uses am imperfect shadow map, where no pull-push was applied. The right image shows the result after pull-push was performed on the imperfect shadow map. The result is similar to the classic shadow map but can be produces much faster (Image courtesy of Ritschel et al. [RGK+08]). 52
2.35 This image shows the rendered scene on the left and the corresponding cache hits (green) and misses (red) on the left. (Image courtesy of Nehab et al. [NSI06]). 53
2.36 The left image shows a scene rendered with the light space perspective shadow maps method [WSP04]. The right image shows the pixel-correct shadows with the method proposed by Scherzer et al. [SJW07]. (Images courtesy of Scherzer et al. [SJW07]). . . . 55

3.1 These images show the process of hierarchical warping for one iteration. (a) shows an initial sample set. (b) is an importance distribution that can be derived from according mipmap levels of the importance map. In (c) the sample set is divided into two areas. The size of the areas correspond to the vertical importance distribution. The upper summed importance distribution is 80%, the lower 20%. Figure (d) shows the first warp of the sample set. Now, about 80% of the sample points are in the upper half and about 20% in the lower half of the quad. The first part was to do a vertical warp, the second step is to do two horizontal warps on the upper and lower half of the quad. According to the importance distribution, the area is divided into two areas (e). After the horizontal warps the sample set shows a distribution (f) that corresponds to the importance map. (Images courtesy of Clarberg et al. [CJAMJ05]) 59
3.2 Left image shows the color channel of the camera G-Buffer. On the right side, the split G-Buffer. 60
3.3 This figure illustrates our method to calculate glossy indirect illumination. 62

3.4 This figure shows the accumulation buffer after shading with all VPLs. To get the final illumination the accumulation buffer must be merged, so that it does not contain any tiles anymore. Afterwards a geometry aware box filter is used to smooth indirect illumination over $n \times n$ pixels. 63
3.5 Image illustrating the confidence for a given arbitrary scene. The brighter the pixels, the more confident a previous pixel is. Note how the confidence is low around sharp corners. 66
3.6 Image shows the illumination map of the imperfect reflective shadow map. 67
3.7 This figure illustrates our method to calculate a new set of VPLs. . 68

4.1 Screenshot of our application. 72
4.2 This figure shows the final results after composition of indirect with direct illumination and the tonemapper. It was rendered using our method with 256 virtual point lights and one light bounce at 73 fps. 79

5.1 This figure shows the Cornell Box rendered with a different numbers of virtual point lights. The visual artifacts are visible on the right wall. They decrease the more VPLs are used. Note that with 1024 VPLs there are nearly no indirect shadows anymore. This comes from a too low number of sample points. Furthermore, due to resampling of the VPL positions the images suffer from temporal artifacts when using less than 256 VPLs. 83
5.2 This figure shows a comparison to the method proposed by Ritschel et al. shown on the left column. Our method is shown in the right column. The images in the top row are rendered using only 64 VPLs and therefore suffer from visible artifacts. The lower row is rendered using 256 VPLs. Here the images are visually more pleasing. However, when the cube stack collapses our method keeps up smooth rendering without flickering artifacts, while the other method does not. 85
5.3 This figure shows a comparison between the imperfect shadow maps method, our new method and a reference rendering, using standard shadow maps. For the reference rendering we have used 1024 virtual light sources, whereas we used 256 VPLs for the other two methods. In the method proposed by Ritschel et al. [RGK+08] the aliasing artifacts are clearly visible. In our method they are not visible due to the temporal smoothing over time. 86

5.4 The image on the left is rendered using our method with only one light bounce. As all the VPLs are placed on the small illuminated spot, the rest of the Cornell Box is not visible. One additional light bounce is already sufficient to illuminate the whole scene. . . 87
5.5 This figure shows a stress test scene where the main light source and the objects are moving very fast. Here the temporal artifacts are clearly visible. However, they are less disturbing than temporal flickering, because they change more smoothly. 89

6.1 This figure shows an image, rendered with our method at 73 frames per second. 93

List of Tables

5.1 This table shows the influence of the number of VPLs that are placed in the scene. The frames per second and times in the first two columns belong to our implementation of the imperfect shadow maps approach by Ritschel et al. [RGK+08]. The last two columns show the results of our new temporal coherence rendering system. The frame rates are lower because of the overhead due to temporal coherence, but the visual results are better with the same amount of VPLs. 82

5.2 This table shows the influence of different imperfect shadow map resolutions. The frames per second and times in the first two columns belong to our implementation of the imperfect shadow maps approach by Ritschel et al. [RGK+08]. The last two columns show the results of our new temporal coherence rendering system. 84

5.3 This table shows the behavior of our multiple bounce calculation method compared to the imperfect reflective shadow maps approach. The measured timings include only the calculation time for the light bounces. ISM creation and shading was disabled for the measurements. In average our method actually performs 11 times faster than the other one. Note that we start with two light bounces, as the first bounce is calculated with the light G-Buffer sampler. Furthermore note that the timing results are rounded and that the speed up calculation is based on the accurate timings. . . . 87

Bibliography

[AK99] Franz Aurenhammer and Rolf Klein. *Voronoi Diagrams - Aurenhammer, Klein*, chapter 5, pages 201–290. Elsevier Publishing House, December 1999.

[BAS02] Stefan Brabec, Thomas Annen, and Hans-Peter Seidel. Shadow mapping for hemispherical and omnidirectional light sources. In *In Proceedings of Computer Graphics International*, pages 397–408, 2002.

[Boa08] CGAL Editorial Board. *CGAL User and Reference Manual*, 3.4 edition, 2008.

[Bun05] Michael Bunnell. *GPU Gems 2*, pages 223–233. Addison-Wesley Longman, har/cdr (1. april 2005) edition, 2005.

[CJAMJ05] Petrik Clarberg, Wojciech Jarosz, Tomas Akenine-Möller, and Henrik Wann Jensen. Wavelet importance sampling: efficiently evaluating products of complex functions. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 1166–1175, New York, NY, USA, 2005. ACM.

[CT81] Robert L. Cook and Kenneth E. Torrance. A reflectance model for computer graphics. In *SIGGRAPH '81: Proceedings of the 8th annual conference on Computer graphics and interactive techniques*, pages 307–316, New York, NY, USA, 1981. ACM.

[CWH93] Michael F. Cohen, John Wallace, and Pat Hanrahan. *Radiosity and realistic image synthesis*. Academic Press Professional, Inc., San Diego, CA, USA, 1993.

[DS05] Carsten Dachsbacher and Marc Stamminger. Reflective shadow maps. In *I3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 203–231, New York, NY, USA, 2005. ACM.

[DS06] Carsten Dachsbacher and Marc Stamminger. Splatting indirect illumination. In *I3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 93–100, New York, NY, USA, 2006. ACM.

[DSDD07] Carsten Dachsbacher, Marc Stamminger, George Drettakis, and Frédo Durand. Implicit visibility and antiradiance for interactive global illumination. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, page 61, New York, NY, USA, 2007. ACM.

[GD98] J. P. Grossman and William J. Dally. Point sample rendering. In *In Rendering Techniques 98*, pages 181–192. Springer, 1998.

[Gre03] Robin Green. Spherical harmonic lighting: The gritty details. *Archives of the Game Developers Conference*, March 2003.

[Gue07] Paul Guerrero. Approximative real-time soft shadows and diffuse reflections in dynamic scenes. Master's thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria, 2007.

[Hal64] John H. Halton. Algorithm 247: Radical-inverse quasi-random point sequence. *Communications of the ACM*, 7(12):701–702, 1964.

[HBS03] Vlastimil Havran, Jiří Bittner, and Hans-Peter Seidel. Exploiting temporal coherence in ray casted walkthroughs. In *SCCG '03: Proceedings of the 19th spring conference on Computer graphics*, pages 149–155, New York, NY, USA, 2003. ACM.

[Hec90] Paul S. Heckbert. Adaptive radiosity textures for bidirectional ray tracing. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 145–154, New York, NY, USA, 1990. ACM.

[HSA91] Pat Hanrahan, David Salzman, and Larry Aupperle. A rapid hierarchical radiosity algorithm. In *SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 197–206, New York, NY, USA, 1991. ACM.

[Kaj86] James T. Kajiya. The rendering equation. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 143–150, New York, NY, USA, 1986. ACM.

[Kel96] Alexander Keller. Quasi-monte carlo radiosity. In *Proceedings of the Eurographics workshop on Rendering techniques '96*, pages 101–110, London, UK, 1996. Springer-Verlag.

[Kel97] Alexander Keller. Instant radiosity. In Turner Whitted, editor, *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, Annual Conference Series, pages 49–56. ACM SIGGRAPH, ACM Press/Addison-Wesley Publishing Co., 1997. ISBN 0-89791-896-7.

[KL05] Janne Kontkanen and Samuli Laine. Ambient occlusion fields. In *SI3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 41–48, New York, NY, USA, 2005. ACM Press.

[KLA04] Jan Kautz, Jaakko Lehtinen, and Timo Aila. Hemispherical rasterization for self-shadowing of dynamic objects. In *Proceedings of Eurographics Symposium on Rendering 2004*, pages 179–184. Eurographics Association, 2004.

[KSS02] Jan Kautz, Peter-Pike Sloan, and John Snyder. Fast, arbitrary brdf shading for low-frequency lighting using spherical harmonics. In *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering*, pages 291–296, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[LSK+07] Samuli Laine, Hannu Saransaar, Janne Kontkanen, Jaakko Lehtinen, and Timo Aila. Incremental instant radiosity for real-time indirect illumination. In *Proceedings of Eurographics Symposium on Rendering 2007*, pages 277–286. Eurographics Association, 2007.

[MKC08] Ricardo Marroquim, Martin Kraus, and Paulo Roma Cavalcanti. Special section: Point-based graphics: Efficient image reconstruction for point-based and line-based rendering. *Computers and Graphics*, 32(2):189–203, 2008.

[MMAH06] Mattias Malmer, Fredrik Malmer, Ulf Assarson, and Nicolas Holzschuch. Fast precomputed ambient occlusion for proximity shadows. *Journal of Graphics Tools*, 12(2):59–71, 2006.

[Nic70] Fred E. Nicodemus. Reflectance nomenclature and directional reflectance and emissivity. *Applied Optics*, 9(6):1474–1475, 1970.

[Nie92] Harald Niederreiter. *Random number generation and quasi-Monte Carlo methods*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1992.

[NRH03] Ren Ng, Ravi Ramamoorthi, and Pat Hanrahan. All-frequency shadows using non-linear wavelet lighting approximation. *ACM Transactions on Graphics*, 22(3):376–381, 2003.

[NRH04] Ren Ng, Ravi Ramamoorthi, and Pat Hanrahan. Triple product wavelet integrals for all-frequency relighting. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 477–487, New York, NY, USA, 2004. ACM.

[NSI06] Diego Nehab, Pedro V. Sander, and John R. Isidoro. The real-time reprojection cache. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Sketches*, page 185, New York, NY, USA, 2006. ACM.

[OBM06] Brian Osman, Mike Bukowski, and Chris McEvoy. Practical implementation of dual paraboloid shadow maps. In *Sandbox '06: Proceedings of the 2006 ACM SIGGRAPH symposium on Videogames*, pages 103–106, New York, NY, USA, 2006. ACM.

[ON94] Michael Oren and Shree K. Nayar. Generalization of lambert's reflectance model. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 239–246, New York, NY, USA, 1994. ACM.

[PH04] Matt Pharr and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

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

[RGK+08] Tobias Ritschel, Thorsten Grosch, Min H. Kim, Hans-Peter Seidel, Carsten Dachsbacher, and Jan Kautz. Imperfect shadow maps for efficient computation of indirect illumination. *ACM Transactions on Graphics*, 27(5):1–8, 2008.

[RGKS08] Tobias Ritschel, Thorsten Grosch, Jan Kautz, and Hans-Peter Seidel. Interactive global illumination based on coherent surface shadow maps. In *GI '08: Proceedings of graphics interface 2008*, pages 185–192, Toronto, Ont., Canada, Canada, 2008. Canadian Information Processing Society.

[RGS09] Tobias Ritschel, Thorsten Grosch, and Hans-Peter Seidel. Approximating dynamic global illumination in image space. In *I3D '09: Proceedings of the 2009 symposium on Interactive 3D graphics and games*, pages 75–82, New York, NY, USA, 2009. ACM.

[Rit07] Ritschel, Tobias and Grosch, Thorsten and Kautz, Jan and Mueller, Stefan. Interactive Illumination with Coherent Shadow Maps. In *Eurographics Symposium on Rendering, June 25 - 27, Grenoble, France*, 2007.

[RSSF02] Erik Reinhard, Michael Stark, Peter Shirley, and James Ferwerda. Photographic tone reproduction for digital images. *ACM Transactions on Graphics*, 21(3):267–276, 2002.

[SA07] Perumaal Shanmugam and Okan Arikan. Hardware accelerated ambient occlusion techniques on gpus. In *I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games*, pages 73–80, New York, NY, USA, 2007. ACM Press.

[SAG94] Brian Smits, James Arvo, and Donald Greenberg. A clustering algorithm for radiosity in complex environments. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 435–442, New York, NY, USA, 1994. ACM.

[Sch09] Michael Schwärzler. Accurate soft shadows in real-time applications. Master's thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria, 2 2009.

[Seg07] Benjamin Segovia. *Interactive Light Transport With Virtual Point Lights*. Thèse de doctorat en informatique, October 2007.

[SHHS03] Peter-Pike Sloan, Jesse Hall, John Hart, and John Snyder. Clustered principal components for precomputed radiance transfer. *ACM Trans. Graph.*, 22(3):382–391, 2003.

[SIMP06] Benjamin Segovia, Jean-Claude Iehl, Richard Mitanchey, and Bernard Péroche. Non-interleaved deferred shading of interleaved sample patterns. In *GH '06: Proceedings of the 21st ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*, pages 53–60, New York, NY, USA, 2006. ACM.

[SIP06] Benjamin Segovia, Jean-Claude Iehl, and Bernard Péroche. Bidirectional Instant Radiosity. In *Proceedings of the 17th Eurographics Workshop on Rendering*, June 2006.

[SIP07] Benjamin Segovia, Jean-Claude Iehl, and Bernard Péroche. Metropolis Instant Radiosity. *Computer Graphics Forum*, 26(3):425–434, September 2007.

[SJW07] Daniel Scherzer, Stefan Jeschke, and Michael Wimmer. Pixel-correct shadow maps with temporal reprojection and shadow test confidence, June 2007.

[SKS02] Peter-Pike Sloan, Jan Kautz, and John Snyder. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *ACM Transactions on Graphics*, pages 527–536, 2002.

[SKvW+92] Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, and Paul Haeberli. Fast shadows and lighting effects using texture mapping. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 249–252, New York, NY, USA, 1992. ACM.

[SSZK04] Mirko Sattler, Ralf Sarlette, Gabriel Zachmann, and Reinhard Klein. Hardware-accelerated ambient occlusion computation. In *9th Int'l Fall Workshop VISION, MODELING, AND VISUALIZATION (VMV)*, pages 119–135, Stanford (California), USA, November16–18 2004.

[WE03] D. Weiskopf and T. Ertl. Shadow Mapping Based on Dual Depth Layers. In *Proceedings of Eurographics '03 Short Papers*, pages 53–60, 2003.

[WFA+05] Bruce Walter, Sebastian Fernandez, Adam Arbree, Kavita Bala, Michael Donikian, and Donald P. Greenberg. Lightcuts: a scalable approach to illumination. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 1098–1107, New York, NY, USA, 2005. ACM.

[WKH01] Alexander Keller Wolfgang, Er Keller, and Wolfgang Heidrich. Interleaved sampling. In *Rendering Techniques 2001 (Proceedings 12th Eurographics Workshop on Rendering*, pages 269–276. Springer, 2001.

[WSB01] Ingo Wald, Philipp Slusallek, and Carsten Benthin. Interactive distributed ray tracing of highly complex models. In *In Rendering Techniques 2001: 12th Eurographics Workshop on Rendering*, pages 277–288. Springer, 2001.

[WSP04] Michael Wimmer, Daniel Scherzer, and Werner Purgathofer. Light space perspective shadow maps. In *Proceedings of Eurographics Symposium on Rendering 2004*, 2004.

[ZIK98] Sergej Zhukov, Andrej Inoes, and Grigorij Kronin. An ambient light illumination model. In George Drettakis and Nelson Max, editors, *Rendering Techniques '98*, Eurographics, pages 45–56. Springer-Verlag Wien New York, 1998.