

Final-Placement BRDF Editing

Aner Ben-Artzi

Submitted in partial fulfillment of the
Requirements for the degree
of Doctor of Philosophy
in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2007

© 2007

Aner Ben-Artzi
All Rights Reserved

Abstract

Final-Placement BRDF Editing

Aner Ben-Artzi

The scene descriptions of computer-generated worlds include the geometry and location of objects, the lighting configuration, and the material properties of object surfaces. Material properties can often be described with the *Bi-directional Reflectance Distribution Function* (BRDF). The BRDF is a function that encodes the way light interacts with a surface, giving rise to different material perceptions, such as wood, metal, or plastic. Using traditional rendering techniques it is not possible to modify the BRDF of a complex scene while rendering it at interactive rates. Therefore, BRDFs are often chosen in simplified scenarios, and only later rendered in their final placement within a scene with complex geometry and lighting. This can lead to unexpected results since the appearance of objects depends on the interaction of lighting, geometry, and materials. In this thesis, I discuss how to use precomputation-based rendering techniques to allow for interactive rendering of changing BRDFs in scenes with complex lighting and geometry, including cast shadows and global illumination. This new capability allows artists and designers to choose the material properties of objects based on accurate feedback so they can have confidence that their decisions are well-informed. Final-placement decision making has been shown for lighting design, and this work is inspired by such precomputed radiance transfer (PRT) for interactive relighting. Linear rendering via a dot-product is introduced for BRDF editing in direct, complex lighting with cast shadows. This requires a new linear BRDF representation that is useful for both editing and rendering. For BRDF editing with global illumination, the image is no longer linear in the BRDFs of the scene. Therefore a

new bilinear reflection operator is introduced to enable precomputing a polynomial representation that captures the higher-order interactions of editable BRDFs. Optimizations for both the precomputation and rendering phases are presented. The final render-time operation is a per-pixel dot-product whose complexity is independent of geometric and lighting complexity, as these aspects of the scene are fixed. Runtime improvements including curve-switching, object freezing, and incremental rendering are used to improve rendering performance without loss of quality.

CONTENTS

<i>Part I BRDFs: Then and Now</i>	2
<i>1. Introduction</i>	3
1.1 A Walkthrough with the Phong BRDF	5
1.2 Current State of the Art	6
1.3 Organization and Contributions	8
<i>2. Previous Work</i>	10
2.1 BRDF Representations	10
2.2 BRDF Editing	14
2.3 Precomputed Radiance Transfer (PRT)	16
2.4 Global Illumination	18
<i>3. A New Editable and Renderable Representation</i>	19
3.1 Cook-Torrance: an analytic BRDF example	22
3.2 Ashikhmin-Shirley: an anisotropic analytic BRDF	23
3.3 Making use of the quotient BRDF	25
3.3.1 Ashikhmin-Shirley with Fresnel edits in 2 factors	25
3.3.2 Anisotropy with 1 Factor	27
3.4 Other Analytic and Hybrid BRDFs	27
3.5 Measured/Factored BRDFs	28

<i>Part II Interactive BRDF Editing Beyond Simple Lighting</i>	32
4. <i>Interactive Rendering in Complex Lighting</i>	33
4.1 Method Overview	33
4.2 Environment Map Precomputation	37
4.2.1 Computing Overlaps	37
4.2.2 Initial Approach	40
4.2.3 Our Algorithm	41
4.2.4 Two Factor BRDFs	44
4.2.5 Use of Daubechies 4 Wavelets	45
4.3 BRDF Editing Results	47
4.3.1 Real-time Edits	47
4.3.2 Precomputation Numbers	51
5. <i>Bilinear Rendering Operators for BRDF Editing</i>	53
5.1 Theory of BRDF Editing in Global Illumination	53
5.1.1 Basic Framework using the bilinear K operator	55
5.1.2 Polynomial Representation for Multi-Bounce	57
5.2 Managing Complexity	61
5.2.1 Low-Frequency BRDF approximations	62
5.2.2 Diffuse approximation for further bounces	64
5.2.3 Slower Decay of BRDF Series	68
5.2.4 Equivalent Albedo	69
6. <i>Interactive Rendering with Global Illumination</i>	72
6.1 Implementation	73
6.1.1 Monte Carlo Precomputation	73

6.1.2	Rendering in Real Time	77
6.1.3	Extensions	80
6.2	Global Illumination Results	82
6.2.1	Editing and Visual effects	82
6.2.2	Precomputation Times and Memory Usage	85
6.2.3	Rendering Speed	87
 <i>Part III New Concepts Applicable to BRDF Editing and More</i>		88
7.	<i>Temporal Coherence for Real-Time Precomputed Rendering</i>	90
7.1	Use in BRDF editing	92
7.2	Use in Relighting	94
8.	<i>Visibility Coherence for Precomputation</i>	97
8.1	Use in BRDF Editing	98
8.2	Use in Offline Still and Animation Rendering	98
8.3	Introduction	99
8.4	Background	101
8.5	Preliminaries	103
8.6	Algorithm Components	104
8.6.1	Estimating Visibility	105
8.6.2	Correcting Visibility Estimates	108
8.7	Analyzing Possible Component Combinations	109
8.7.1	The Scenes	110
8.7.2	Correctness Analysis (Scanline vs. Grid)	111
8.7.3	Performance Analysis	112
8.7.4	Discussion of Analysis	115

8.8	Optimizations and Implementation	116
8.8.1	Optimized Algorithm	117
8.8.2	Implementation	120
8.8.3	Time and Memory Considerations	123
8.9	Other Considerations	125
9.	<i>User Interface for 1D Function Editing</i>	127
9.1	Empirical Definition of Helper Functions	130
9.2	Smoothness and Layered operators	131
9.3	Use in BRDF Editing and Other Graphics Applications	131
<i>Part IV Conclusion</i>		133
10.	<i>Conclusion</i>	134
10.1	Final-Placement	134
10.2	Theoretical Framework with Few Assumptions	135
10.3	Complement to Relighting	136
10.4	Continuous Editing and Discrete Choices	137
11.	<i>Future Directions</i>	138
11.1	Higher-Complexity Functions	138
11.2	Practical Extensions	139
11.3	Advantages of Precomputation-based Rendering	141
<i>References</i>		142

LIST OF TABLES

4.1	Daubechies 4 Coefficients	45
4.2	Timings and Memory for Complex Lighting	52
5.1	Table of Notation for Chapters 5 and 6	55
6.1	Timings and Storage for Global Illumination	86
7.1	Didactic Example of Incremental vs. Non-Linear	93
8.1	Percentage of Shadow-Rays Traced	112
8.2	Visibility Timings	113
8.3	Shadow-Ray Timings	123

LIST OF FIGURES

1.1	Teatray with Cloth	4
1.2	Edit in Complex Lighting	6
1.3	Edit in Point Lighting	6
1.4	Edit without Shadows	7
3.1	Edit of Phong Curve	21
3.2	Specularity Edit on Pearls	22
3.3	Fresnel Edit on Stems	23
3.4	Anisotropic Edit on Teapot	25
3.5	McCool Edit on Cloth	29
3.6	Factored BRDF Edits	30
3.7	Measured BRDF Edits	31
4.1	Light-Bands Overlaps	38
4.2	Triangle-Range Overlap	38
4.3	Point vs. Area Samples	42
4.4	Teatray Results	48
4.5	Earrings Results	49
5.1	Rendering Operators	54
5.2	BRDF Polynomial	54
5.3	Comparison of Bounce Resolution	60

5.4	Exponential Growth wrt BRDF Bases	61
5.5	BRDF Edits in Cornell Box	65
5.6	Errors in Simpler Approximations	66
6.1	Interior Design Session	72
6.2	Light Paths	75
6.3	Vase with Flowers	82
6.4	Room Closeup	84
6.5	Two-Bounce Glossy Edits	85
7.1	Incremental BRDF Editing	94
7.2	Incremental Relighting	95
8.1	Fast Visibility Ground Truth Comparisons	98
8.2	Domino Scene Comparing Visibility Calculations	99
8.3	Voronoi Regions	104
8.4	Grid-Based Evaluation	106
8.5	Warping Blockers	106
8.6	Light-space Flooding	108
8.7	Shapes and Bunny Ground Truth Comparisons	110
8.8	Error Analysis of Visibility Coherence Components	111
8.9	Plant Comparisons of Visibility Algorithms	117
9.1	Y-Translation	128
9.2	X-Translation	129
9.3	Y-Amplification	130
9.4	X-Amplificaiton	130

Acknowledgments

To my advisor, Ravi, thank you for carrying me through these five years and helping me come out on the other side with a body of work that I can be proud of. Your example was a high bar to reach for, and I got farther for stretching to reach it.

To Eitan Grinspun, thank you for your support and encouragement and friendship. You added a constant light¹ to my time at Columbia.

To Ryan Overbeck, Kevin Egan – my coauthors and lab mates – thank you for the chats, the lunches, and your support. Without you, this work would not be what it is today. Everyday I’d come into the lab, the first thing I did was check to see if you were around.

To Shree Nayar and Peter Belhumeur, thank you for always having the time to listen, and presenting me with excellent role models. To Frédo Durand and Szymon Rusinkiewicz, thank you for sharing your wisdom and experience on our projects together. And thank you to all four for sharing your time to be on my committee and participate in my final step on this journey.

To Jason Lawrence, thank you for visits to Princeton that I always looked forward to, and a ‘snazzy’ time editing text hours before the deadline.

To my parents, Dalia and Itai, thank you for raising me to believe that what I envision for my future will one day be my present. This thesis and doctoral degree seemed so far away at one time, and now they’re a reality. You’ve given me everything I need to be successful.

To my wife Nicole, thank you for your constant support and being my cheerleader every day.

¹ literally and figuratively

This thesis is dedicated to my wife, Nicole, and to our future life together.

Part I

BRDFS: THEN AND NOW

1. INTRODUCTION

Computer graphics develops tools for skilled artists, designers, and engineers to create images based on the building blocks of geometry, material properties, and lighting. Interactive (versus offline) design of these scene components allows users not only to converge on a goal more quickly, but also to explore the design space and discover new effects. Material properties are often expressed as the bi-directional reflectance distribution function (BRDF), defined in (Nicodemus *et al.*, 1977). While lighting design and geometric modeling have analogs in the real world, it is not usually possible to continuously edit material properties. However, in a computer simulated world, editing BRDFs has no fundamental distinction from editing other aspects of a scene (such as lighting or geometry), and is equally important in contributing to the final scene appearance. Figure 1.1¹ shows the same scene, illuminated by the same environment map, with different BRDFs. Notice how the choice of materials affects the mood of the arrangement. Also, the materials have to be selected in their final-placement since aspects such as the elongation of specularities and shadow shape result from an interaction of all three aspects of scene appearance : lighting, geometry, and materials.

Lighting design has been an active field of research for photorealistic rendering since its introduction in (Nimeroff *et al.*, 1994) and (Dobashi *et al.*, 1995). More recent work by (Sloan *et al.*, 2002) has rekindled an interest in real-time relighting

¹ All images of 3D scenes in this thesis are rendered with systems implemented for this work, except where explicitly stated for ground-truth comparisons.

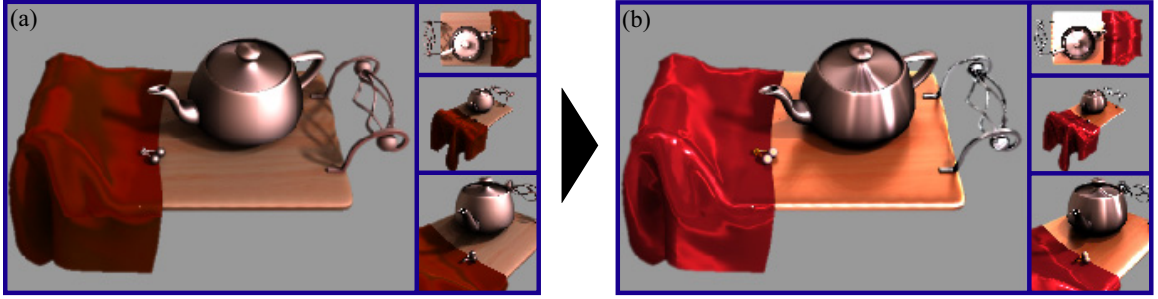


Fig. 1.1: Teatray with Cloth: A scene shown before (a) and after (b) the BRDFs of all objects were edited. The lighting, geometry, and view have not changed. Notice the differences in shadows and highlights.

of scenes that incorporate environment map lighting and global illumination effects such as cast shadows and interreflections. Since (Sloan *et al.*, 2002)’s seminal work, progressive improvement has been made to increase the generality and realism of relighting systems by incorporating all-frequency effects, and more complex BRDFs and geometry (Liu *et al.*, 2004; Ng *et al.*, 2003; Ng *et al.*, 2004; Sloan *et al.*, 2003; Sloan *et al.*, 2005; Wang *et al.*, 2004).

Despite the large body of work on relighting, BRDF editing has remained in the realm of rough-cut design. BRDF parameters are chosen independently of the final lighting, geometry, or camera position. This makes final-cut renderings the first time an artist can see how his or her choices in material properties affect the final image. In recent years, measured data has become more common in computer generated scenes. Range-scans capture geometry, light probes capture lighting environments, and acquisition systems such as (Dana *et al.*, 1999) and (Matusik *et al.*, 2003) capture real-world BRDFs. Editable BRDF representations have lagged in their incorporation of measured data. The most common use of acquired BRDF data is to fit it to an analytic model, or use it as-is. Our work fills a major gap in the way scenes are designed by enabling final-placement BRDF editing, as well as demonstrating how

measured BRDF data can be edited and used without resorting to parametric fits. With the use of our real-time rendering system, images such as the one shown in Figure 1.1 can be updated as the user edits the BRDFs of scene objects. Updates include all-frequency complex lighting and accurate cast shadows. The cloth in Figure 1.1 uses a measured BRDF that is never fit to any analytic model, yet can still be edited intuitively. Previous methods were only able to render such scenes in simple lighting or without cast shadows, greatly limiting their usefulness for determining the BRDFs' influence on the appearance.

1.1 A Walkthrough with the Phong BRDF

We seek to edit BRDFs in their final placement on objects already arranged in a scene, lit by complex illumination. With this ability, we provide control over an important component of appearance: material properties. More powerful than simply selecting a pre-existing material, the designer has the freedom to adjust the BRDF to simulate novel materials. This can include fine-tuning a measured dataset, exploring variations of analytic models, venturing outside the analytic form of those models, or designing a new BRDF from scratch. Let's begin with an example that must be similar to one of the first explorations of BRDF space in computer graphics, and which should be familiar to anyone who has worked with computer-generated scenes: choosing a Phong exponent. Figure 1.2 shows a scene illuminated by an environment map, with increasing Phong exponent; ranging exponentially from 4 to 1024. Each of these images would take several minutes to render with traditional ray-tracing.

Many interesting effects occur in this scene: The objects appear to become darker resulting from the localization of very bright points, due to the narrower Phong lobe. Shadow lines appear and disappear as the Phong lobe narrows, giving different relative

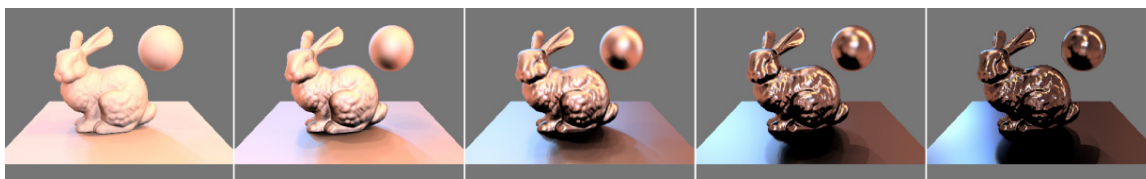


Fig. 1.2: Edit in Complex Lighting: All surfaces use the Phong model with exponent increasing from left to right (4, 16, 64, 256, 1024). The lighting environment is an HDR light probe of St. Peter's Basilica.

importance to different parts of the illumination. Dark reflections appear and become more defined.

1.2 Current State of the Art

How does one choose the exact Phong exponent to preserve or avoid certain complex interactions between the BRDF and the scene's geometry and lighting? A binary search over the space is one option, but hardly a viable solution if each rendering takes several minutes. The traditional approach is to simplify some aspect of the scene to allow for interactive rendering. One simplification is to use a point light. Often the BRDF is chosen by visualizing a sphere illuminated by a point light. Figure 1.3 shows this scenario, and clearly reveals the drawbacks of trying to choose BRDF parameters based on simple lighting.



Fig. 1.3: Edit in Point Lighting: A point light is used to illuminate this scene to enable interactive rendering. The visual effects are quite different from environment map illumination, and are not useful for determining final-placement appearance.

The most noticeable difference between Figure 1.2 and Figure 1.3 is that since

there is no light in the reflected direction of the plane, it quickly becomes black, revealing nothing about its color or shadows. An alternative that preserves the richness of full-sphere illumination is to render using preconvolved environment maps, as seen in Figure 1.4.

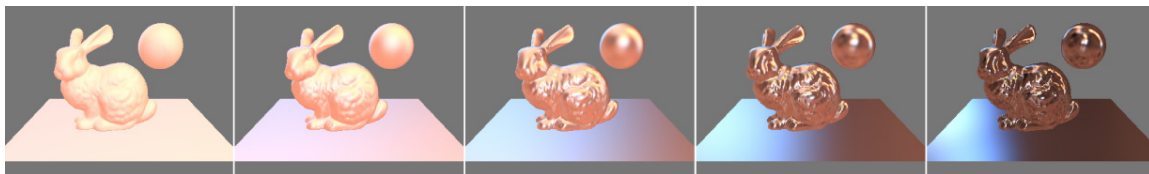


Fig. 1.4: Edit without Shadows: Rendering with preconvolved environment maps captures the effect of full-sphere illumination while ignoring cast shadows.

As expected, no information about the shadows can be extracted from the images, and even a talented imagination would have a hard time filling them in. This also leads to a difference in shading quality as noticeable by the incorrect brightness of the bunny's knee. Finally, interaction between the scene objects is completely lost: Is the bunny hovering above the plane? Is the sphere very large and far away? Preconvolved environment maps are restricted to radially symmetric BRDFs, and thus cannot handle half-angle-based or measured BRDFs ((Kautz *et al.*, 2000)). Renderings such as those in Figure 1.3 and Figure 1.4 are still useful to an experienced designer. The designer's familiarity with the Phong model, coupled with the limited information provided by simple-lighting or shadowless renderings can guide the BRDF selection process. However, there is clearly a better solution - to render the final images correctly, as in Figure 1.1 and Figure 1.2. The main contribution of this thesis is a real-time rendering algorithm and system that allows for final-placement BRDF editing.

1.3 Organization and Contributions

This thesis is organized into four parts as follows:

Part I provides a motivation, background information, and introduction to a new BRDF representation which will be used in Part II. Chapter 2 provides a history of related work. The most recommended background readings are given in each section. Chapter 3 introduces a new BRDF representation, which is used for interactive editing. It linearizes the often non-linear nature of BRDF edits. This representation is useful for editing as well as rendering operations, minimizing the need to convert between representations at run-time. It is designed to respect the long history of analytic BRDF models, as well as include support for data-driven representations.

Part II is the main contribution of this thesis, describing how to edit BRDFs in complex lighting, and then in global illumination, fulfilling the promise of final-placement BRDF editing. Chapter 4 discusses how to use the representation introduced in Chapter 3 for real-time BRDF editing with complex lighting and shadows. Chapter 5 introduces a bilinear operator framework to describe the reflection equation. This framework enables us to describe the higher-order effects of BRDF changes on a scene when global illumination is considered. Chapter 6 uses the bilinear operator framework introduced in Chapter 5 in an interactive BRDF editing system that takes into account interreflections for true global illumination rendering.

Part III goes into more detail on novel concepts that have applicability for both BRDF editing (as used in Part II), as well as other rendering applications. Chapter 7 discusses an improvement to the dot-product operation which is at the heart of precomputation-based rendering. Using temporal coherence, higher quality rendering can be achieved at the same cost. This speedup is applicable to both BRDF editing and relighting systems. Chapter 8 discusses how to calculate direct-lighting visibility

from an environment map more efficiently. This process is used for speeding up the precomputation stage of real-time editing methods, as well as offline rendering. Chapter 9 introduces a new set of operations to manipulate the shape of 1D functions. Editing 1D functions is at the heart of many graphics-related operations, including the BRDF editing discussed in Part II, and adjusting other 1D behavior such as animation paths and exposure curves.

Part IV provides a brief conclusion and discussion on future directions for the work. Chapter 10 highlights some important conclusions of the work presented in this thesis. Chapter 11 is a discussion of immediate, mid-term, and long-term extensions for this work.

2. PREVIOUS WORK

The work in this thesis draws together several areas within rendering that have had separate histories. Section 2.1 discusses BRDF representations, which have often been developed with little consideration for interactive BRDF editing (Section 2.2). Section 2.3 talks about precomputed radiance transfer (PRT), which has been used to enable real-time relighting. PRT techniques are the forerunners to the precomputation-based rendering used in Chapter 4 for BRDF editing. Section 2.4 highlights some of the relevant works in global illumination that serve as the backdrop for the methods introduced in Chapter 6.

2.1 BRDF Representations

The BRDF was formally defined as a ratio between incoming irradiance and outgoing radiance by (Nicodemus *et al.*, 1977). It can also be thought of as a probability distribution function which describes the way an incoming light ray is likely to be scattered by the surface into outgoing light rays. Throughout this thesis, I will use the notation $\rho(\omega_{in}, \omega_{out})$ for the BRDF, where ω_{in} is the incoming light direction, and ω_{out} is the outgoing view direction. A direction can be represented as a unit cartesian vector: $\omega \equiv (x, y, z)$ or in spherical coordinates¹: $\omega \equiv (\theta, \phi)$.

¹ This document uses the graphics convention that θ is the elevation angle, and should not be confused with the mathematical community’s notation that chooses ϕ for that role.

Analytic: Functional representations of surface reflectance properties have been around even before the first CG renderings (Minnaert, 1941; Hapke, 1963; Torrance & Sparrow, 1967). These functional models describe a ratio that relates how much light falling on a surface from a particular incoming direction will exit towards a particular outgoing view direction. Adjustable parameters allow a single model to describe an entire family of materials. For example, the Minnaert and Hapke models describe the lunar surface, adjusting for different amounts of moon dust. For the most part, the graphics community has continued this tradition of creating models which are functions of the four dimensions: θ_{in} , ϕ_{in} , θ_{out} , and ϕ_{out} , and a small set of parameters that controls the behavior of these functions. Most models have discovered that a reparameterization of the canonical directions yields simpler expressions. For example, the Torrance-Sparrow BRDF defines a new angle: $\theta_{half} = \cos^{-1}(N \cdot (\frac{\omega_{in} + \omega_{out}}{|\omega_{in} + \omega_{out}|}))$, and Minnaert defines $\omega_{reflected} = N \cdot (2\omega_{in}) - \omega_{in}$. Most BRDF models continue to use these two popular reparameterizations. In 1998, Rusinkiewicz formalized the half- and difference-angle parameterization as a change of variables from the canonical four dimensions.

Among the first BRDF models used in computer graphics (and also the simplest and still most ubiquitous) is the Phong model (Phong, 1975). A slightly more elaborate model is Blinn-Phong (Blinn, 1977), which replaces the $\theta_{reflected}$ parameterization with θ_{half} and adds some normalizations and a Fresnel term. Blinn-Phong is very similar to Torrance-Sparrow (Torrance & Sparrow, 1967), and its adaptation for computer graphics – Cook-Torrance (Cook & Torrance, 1982). The main difference is the function that controls the shape of the specular fall-off. As we show in Chapter 3, these three models are unified into a single model by our linearized representation. The models mentioned so far can only capture isotropic reflections. Anisotropic extensions of these models are in popular use for representing materials such as brushed

metal. The Ward model (Ward, 1992) and Ashikhmin-Shirley BRDF (Ashikhmin & Shirley, 2000) add a parameter that allows (indirect) control of the eccentricity of the highlight. Their main difference is the choice of highlight direction, with Ward using $\omega_{reflected}$ and Ashikhmin-Shirley using ω_{half} .²

A single material is often represented as the sum of several BRDFs. Most commonly, a diffuse layer and a specular layer are summed, each one being represented with some function of (ω_i, ω_o) . Each analytic BRDF model has a small set of parameters that can be adjusted to control the material properties represented by the model. Throughout this work, I will refer to these parameters as *user-controlled variables* (UCV). BRDF design and editing with these models amounts to adjusting these 0D parameters, usually with a slider or color picker. Examples of such UCVs are K_s – a scalar for the specular component of the material, n or σ – a scalar that controls the width of the specular highlights, and α_d – a color value that multiplies the diffuse component. In this thesis, I assume that user interfaces already exist to control the behavior of analytic BRDFs, and that the values of these UCVs will be passed to the rendering engine from these interfaces.

Measured BRDFs: Analytic BRDF models are restricted to representing behavior that fits some prescribed form. However, real-world materials deviate from these ideal forms (sometimes slightly, sometimes greatly). To better represent real materials, their reflectance properties can be measured, and those measurements are either used directly (Matusik *et al.*, 2003), or fit to one or more analytic forms (Ward, 1992; Lafortune *et al.*, 1997). BRDF measurements can be achieved with a controlled setup and a gonireflectometer or digital camera. The incoming and outgoing directions are controlled via light and camera placement, and measurements are tabulated into a 4D

² Interestingly enough, the Ashikhmin-Shirley paper is titled "An Anisotropic *Phong* BRDF model", even though Phong is associated with $\theta_{reflected}$.

dataset. (Dana *et al.*, 1999; Matusik *et al.*, 2003) captured a database of materials with such a setup.

Instead of fitting the data to a model, the 4D/3D data may be factored into 2D or 1D tabulated sets for direct use in rendering applications via table lookups (Kautz & McCool, 1999; McCool *et al.*, 2001). These factored representations often use the Rusinkiewicz (Rusinkiewicz, 1998b) reparameterization to improve data coherence along the chosen dimensions. High quality measurements such as those made by (Matusik *et al.*, 2003) have lead to an interest in higher quality factorization techniques (Lawrence *et al.*, 2004). Such measured datasets can be used without fitting to analytic functions in order to preserve the real-world nuances in the data. (Ashikhmin *et al.*, 2000) described a hybrid BRDF that uses a 2D map for variation along the half-angle dimensions, but an analytic expression for variation along other dimensions (difference-, view-, and light-directions). For the highest-quality offline renderings, the 4D measured data is used directly³. (Lawrence *et al.*, 2006) have shown that lower dimensional factors can be extracted with low reconstruction error.

Despite recent advancements in BRDF measurements, analytic models such as Blinn-Phong (Blinn, 1977) and Cook-Torrance (Cook & Torrance, 1982) are still very popular in many applications due to their simplicity and adequate accuracy for many applications.

³ However, interpolation between the available measurements is not trivial.

2.2 *BRDF Editing*

Parametric models can be easily edited (under point source lighting) by simply moving sliders for their parameters, and such software is available for research (Rusinkiewicz, 1998a) and production (Maya, 3DMax). Abstractions can be used on top of a model’s parameters to make material editing more intuitive. The simplest example is to create a shininess parameter on top of the Phong model, which may be linked to the exponent non-linearly and additionally control a normalization parameter simultaneously. These types of edits rely on the GPU pipeline to provide interactive feedback. Shadowing techniques may be used to enhance the rendered results. However, GPU speed depends on the number of lights, complexity of the shadow technique, number of vertices (and pixels), and BRDF complexity. As these burdens increase, the GPU renderings slow down, and quickly fall out of the interactive regime. Specifically, for complex BRDFs that make use of modern shader capability, interactive rates cannot be maintained for more than a few lights when per-pixel calculations are performed in Phong-style shading. In our current work, we respect the existing history for editing analytic BRDFs, and our techniques are compatible with current popular user interfaces.

Preconvolved environment maps (Ramamoorthi & Hanrahan, 2002) are able to render arbitrary BRDFs under complex lighting, but only in the low-frequency domain. The preconvolution step is quick, and may be combined with live editing of BRDFs. (Colbert *et al.*, 2006) uses preconvolved environment maps and fits to the Ward model (Ward, 1992) for editing BRDFs that can be represented by that model. Any method based on preconvolved environment maps is limited to renderings without cast shadows. As seen in Chapter 1 (Figure 1.4), this is a large drawback when attempting to make decisions about the choice of BRDF parameters.

Previous work has not focused much on real-time editing of data-driven reflectance. (Ashikhmin *et al.*, 2000) use GPU-base rendering with 2D maps, which can be edited using standard image-editing software. However, this approach is not always intuitive for deriving new physically-based BRDFs, and these methods suffer from the same limitations as analytic GPU-based rendering. In this work, we introduce a more intuitive editing technique of measured data, based on 1D factors. Our representation can be used with GPU shaders for simple lighting scenarios (Chapter 3). We also describe a new real-time rendering system that can render using this representation in real-time under complex illumination, including cast shadows, and even include full global illumination.

For specific applications like car paint design, specialized systems (Ershov *et al.*, 2001) have been developed to allow edits of a custom designed BRDF model. They quote near-interactive frame-rates (1-6fps), but like GPU approaches suffer from degraded speed as the BRDF becomes more complex, and do not demonstrate general complex illumination. To our knowledge commercial systems allow users to specify weights for a linear combination of pre-defined materials [PEARL], while viewing the results in a static scene under complex illumination. However, a few pre-defined materials do not suffice for many applications: (Matusik *et al.*, 2003) cites a requirement for 45 linear basis materials to allow good reconstruction of materials from their database.

2.3 Precomputed Radiance Transfer (PRT)

The linearity of light transport has been leveraged in a large body of work on relighting. Beginning with (Nimeroff *et al.*, 1994), lighting could be edited by representing the user’s input as a linear combination of lighting functions that can be combined to generate a relit image. More recently, precomputed radiance transfer (PRT) for static scenes (Sloan *et al.*, 2002) has rekindled an interest in basis projections of editable lighting. This, and subsequent work later enabled changing view (Sloan *et al.*, 2003) and some dynamic geometry (Sloan *et al.*, 2005; Zhou *et al.*, 2005), but are all limited to low-frequency effects due to the choice of spherical harmonics as the basis for lighting. Editing within a restricted family of BRDFs is possible with (Sloan *et al.*, 2002), but only for low-frequency, radially symmetric Phong-like models.

Since a material designer wants to preserve the full richness of effects in the BRDF and lighting while editing, we focus instead on all-frequency wavelet-based approaches (Ng *et al.*, 2003). We build on PRT ideas for static scenes (Sloan *et al.*, 2002; Ng *et al.*, 2003). While those methods focus on lighting design with fixed BRDFs, we focus on BRDF editing, with fixed lighting. We are inspired by a body of recent work that underscores the importance of interreflections in relighting (Hasan *et al.*, 2006; Wang *et al.*, 2004; Kontkanen *et al.*, 2006). All these approaches exploit the linearity of relighting with respect to light intensity, even when global illumination is taken into account. In contrast, BRDF editing is fundamentally nonlinear when global illumination is considered. While recent advances allow for changing view and lighting (Sloan *et al.*, 2003; Wang *et al.*, 2004; Ng *et al.*, 2004), they require a precomputed factorization or tabulation of the BRDF, that can neither be edited continuously, nor recomputed in real-time.

It is also important to note some differences that make BRDF editing a funda-

mentally harder problem than relighting. The BRDF lobe at a given pixel is affected by all of the: lighting, view, and geometry (surface normal). This is why we must fix these quantities, while relighting methods can sometimes factor shadowing effects from material properties (Ng *et al.*, 2004). Some flexibility for viewpoint change is possible in both relighting (Liu *et al.*, 2004; Wang *et al.*, 2004) and our method if one uses an incident-outgoing factorization, but that is accurate and editable only for a limited class of diffuse-like materials or components. Moreover, for material design, the final image is not even linearly related to the BRDF if global illumination is considered, which is why we focus on all-frequency direct lighting.

Most previous PRT methods precompute a linear light transport vector at each image location, taking advantage of the linearity of light. We extend this concept to a general tensor of coefficients for a high-dimensional polynomial. The idea of going from linear to quadratic or cubic precomputed models has also begun to be explored in physical simulation (Barbic & James, 2005) but in the context of differential equations and model dimensionality reduction. In the context of real-time rendering, (Sun & Mukherjee, 2006) precompute with a larger product of functions, leading to an N -part multiplication at runtime. Each function is still precomputed independently, and the runtime calculations are still linear in any of the individual functions.

2.4 Global Illumination

Our precomputation method for global illumination effects in Chapter 6 is inspired by offline global illumination algorithms, such as Monte Carlo path tracing (Kajiya, 1986). We have also been able to adapt finite element radiosity (Cohen & Wallace, 1993), although we found meshing and complexity issues difficult to deal with for our complex scenes, and do not discuss it further. By basing the precomputation on path tracing, the quality of our renderings are limited (or unbounded) in the same way as rendering a single off-line image of the scene.

Global illumination techniques usually require the BRDF to be fixed, and use it for importance sampling or hierarchy construction—we develop extensions that are independent of the BRDF, and allow real-time editing. In effect, we precompute a *symbolic representation* of the image, which can be evaluated at runtime with polynomials in the user-specified BRDF values, to obtain the final intensity.

Séquin and Smyrl (Séquin & Smyrl, 1989) also precomputes a symbolic representation of the final image for recursive ray tracing—but not full global illumination. Phong shading can be evaluated at runtime, allowing later changes to surface parameters, while reflected and refracted contributions are handled with pointers to sub-expressions. In contrast, we seek to simulate complex lighting and full global illumination, with many possible illumination paths. Therefore, we cannot afford to store or sum all subexpressions. Instead, we show that the final symbolic expression is a polynomial and only precompute its coefficients. We also allow editing of general parametric and measured BRDFs.

In Chapter 5, we extend Arvo’s (Arvo *et al.*, 1994) linear operators for global illumination. Using that operator notation, it is possible to express the radiance due to multiple bounces without explicitly writing cumbersome nested integrals.

3. A NEW EDITABLE AND RENDERABLE REPRESENTATION

Editing BRDFs is a more structured process than editing images, lighting, or geometry. BRDFs have understandable behaviors that are coupled with known physical phenomena, and this is clearly visible in the structure of functional models such as Cook-Torrance (Cook & Torrance, 1982) and Ward (Ward, 1992). In short, not all 4D functions are BRDFs. To express this, we use the following representation for BRDFs:

$$\rho(\omega_i, \omega_o) = f_q \rho_q(\omega_i, \omega_o) f_o(\gamma_o(\omega_o)) f_A(\gamma_A(\omega_i, \omega_o)) f_B(\gamma_B(\omega_i, \omega_o)), \quad (3.1)$$

where the γ s are 1D reparameterizations of the canonical directions¹, and the f s are 1D functions that describes how the BRDF varies along their respective γ . Common choices for γ are θ_{half} , $\theta_{reflected}$, and θ_{diff} . f_o is dependent only on the view direction, ω_o . One might expect a similar term utilizing only ω_o , such as $(f_{in}(\gamma_{in}(\omega_i)))$. However, because there is only one view for a given pixel/image and many lighting directions, ω_o and ω_i are not treated symmetrically. The focus of this work is to recognize and address the complexity of realistic lighting environments, and any dependence of the BRDF on ω_i is captured in f_A or f_B . ρ_q is the *quotient BRDF* which contains more general, but uneditable variation in the BRDF. A common example of ρ_q is the geometric-attenuation-factor (GAF). f_q is the 0D *quotient factor* which may be edited, but is independent of the canonical directions (implicitly or explicitly). A

¹ The canonical directions are θ_{in} , ϕ_{in} , θ_{out} , and ϕ_{out} as defined in (Nicodemus *et al.*, 1977).

common example of f_q is a scaling factor that depends on the model’s parameters, but not ω_{in} nor ω_{out} .

2D parameterizations are reasonable to consider, such as the 2D microfacet distribution maps in (Ashikhmin *et al.*, 2000). However, inspection of these maps reveals that they exhibit variation only along 1D for realistic isotropic BRDFs. Anisotropic BRDFs will be shown to fit into the form of equation 3.1 with a proper choice of γ ’s. In addition, we will show in Section 3.5 that editing 1D functions is much more intuitive. 2D editing via image-editing software is inappropriate for BRDFs that are typically much more constrained than a general 2D function. It is important to note that many BRDF models describe a sum of terms, such as diffuse-plus-specular. In this work, we consider each of these terms to be its own BRDF, and handle summation in the trivial manner of adding the final radiances that result from each term. A longer list of editable factors (f_C, f_D, \dots) might be considered, but BRDFs rarely exhibit interesting behavior which a user may wish to edit, along more than two parameterizations (not including γ_o). Furthermore, we have not found a case where more factors are needed.

The benefits of this representation are three-fold. First, it is general enough to encompass all commonly used analytic BRDF models, as well as factored measured BRDF data. Second, it allows for intuitive editing by exposing the variations along different directions as understandable 1D functions. In Chapter 9, we will demonstrate our system for editing these 1D functions. Finally, and perhaps most importantly, it can be used in our real-time rendering systems, described in Part II.

As a simple example of how our representation can be applied to an analytic model, consider the scaled Phong model we used to generate Figures 1.3-1.4: $\sqrt{n} \cos^n(\theta_r)$. The parameterization $\gamma_A(\omega_i, \omega_o) = \theta_r$ is the angle between the reflected light and the view direction. The curve $f_A(\gamma_A) = \cos^n(\gamma_A)$, is controlled by the ‘parameter’

n . To avoid confusion with the curve parameterization, we will refer to an analytic parameter, such as the Phong exponent, as a *user-controlled variable* or UCV. The unchanging quotient BRDF, ρ_q , is 1 in this case, or the traditional cosine falloff, $\max(0, \theta_i n)$, if we wish to consider that as part of the BRDF. Finally, the quotient factor, f_q , is the scaling term that we added to keep the images from becoming too dark as the exponent increased: \sqrt{n} . The other two factors (f_B and f_o) are not used, and set to unity. Figure 3.1 shows the 1D factor (with f_q). Notice the y-scale of each graph. In some cases it will be more convenient to show the 1D factors with the scale factor. The y-scale label should make it clear when this is the case.

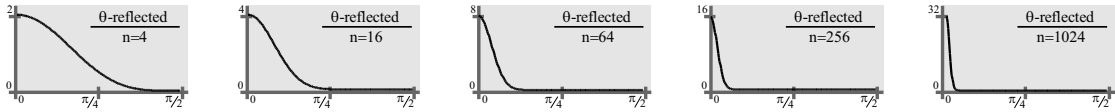


Fig. 3.1: Edit of Phong Curve: 1D factor of the Phong model used in Figures 2-4 with the exponent set to 4, 16, 64, 256, and 1024.

For purely functional models like the Phong example in Figure 3.1, the curves shown are only used by our rendering system, and not revealed to the user. The standard slider controls are available to the user, and they in turn determine the 1D function by direct evaluation of the BRDF model. The next four sections (3.1, 3.2, 3.3, and 3.4) describe in more detail how common analytic models fit into our representation. Section 3.5 shows how revealing these curves to direct user manipulation allows measured data to be edited.

3.1 Cook-Torrance: an analytic BRDF example

Like many analytic BRDFs, the Cook-Torrance model [1982] is already in the desired form presented in equation 3.1. The specular term of CT is:

$$\rho_{CT} = s \frac{F_{n,e}(\theta_d) G(\omega_i, \omega_o) D_\sigma(\theta_h)}{\pi(\omega_i \cdot N)(\omega_o \cdot N)}, \quad (3.2)$$

where N is the surface normal. The user-controlled variables are n (index of refraction), e (extinction coefficient), σ (mean slope distribution), and s (specular term scalar). The first two UCVs control the shape of the Fresnel term, $F(\theta_d)$, while σ controls the shape of the slope distribution function, $G(\omega_i, \omega_o)$, often taken to be the Beckmann distribution. The geometric attenuation factor, $G(\omega_i, \omega_o)$, and the denominator are fixed for all values of the UCVs, and form the quotient brdf, ρ_q . This BRDF requires two 1D factors: $f_A = D(\theta_h)$ and $f_B = F(\theta_d)$. The corresponding parameterizations are: the theta components of the half-difference parameterization: $\gamma_A = \theta_h(\omega_i, \omega_o)$ and $\gamma_B = \theta_d(\omega_i, \omega_o)$. The quotient factor, f_q , is $\frac{s}{\pi}$.

Figure 3.2 shows how a user's choice of σ changes the BRDF of the pearls. At each frame, a slider (not shown) is used to specify the value for σ , which is then used to generate the displayed curve, $f_A(\theta_h)$ (via explicit evaluation of the Beckmann function).

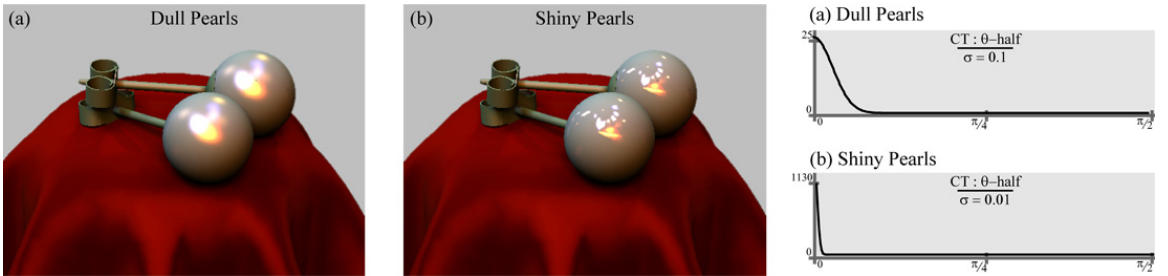


Fig. 3.2: Specularity Edit on Pearls: Changing the sigma in Cook-Torrance. Images in Grace light probe, with shadows generated using our real-time rendering system.

When the user changes the index-of-refraction (n) or the extinction coefficient (e), the new value is used to generate $f_B(\theta_d)$ (by evaluating the Fresnel equation). Figure 3.3 shows an example of changing the index-of-refraction in the Fresnel term of the posts. In this case, a different value was chosen for each of the three color channels. Note that this is not the same as multiplying by a single scalar color. As expected from the Fresnel effect, the color desaturates near grazing angles ($\theta_d = \frac{\pi}{2}$), as indicated by the merging of the three functions.

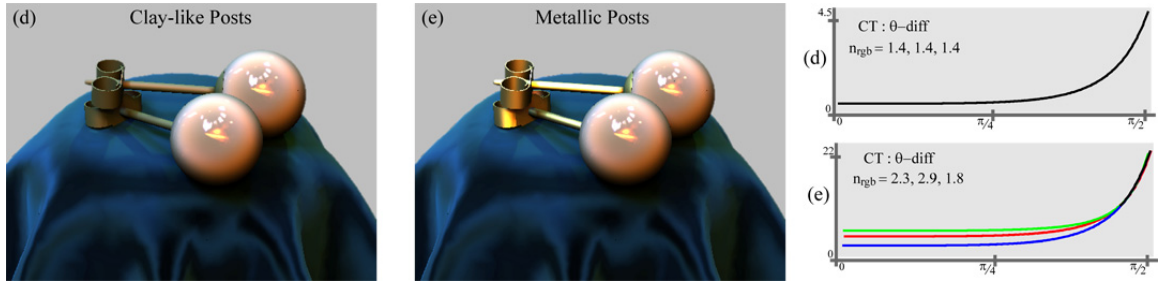


Fig. 3.3: **Fresnel Edit on Stems:** Changing the index of refraction in Cook-Torrance, using our system as described in Section 4.

3.2 Ashikhmin-Shirley: an anisotropic analytic BRDF

Our representation can also be applied to anisotropic analytic models, such as the anisotropic Phong BRDF (Ashikhmin & Shirley, 2000). We will use this to demonstrate a step-by-step separation of an analytic BRDF into the form required by equation 3.1.

We begin with the specular component of the AS BRDF, as it appears in their paper, with minor notational adjustments:

$$\rho_{AS} = \frac{\sqrt{(n_u + 1)(n_v + 1)}}{8\pi} \frac{(\cos \theta_h)^{n_u \cos^2 \phi_h + n_v \sin^2 \phi_h}}{\theta_d \max(\cos \theta_i, \cos \theta_o)} F(\theta_d) \quad (3.3)$$

The first step is to identify the UCVs of the model: n_u and n_v . These are similar to the exponent in the Phong model, but having two controls allows for anisotropic highlights. Next, we find the smallest subexpression that contains all instances of all UCVs:

$$\sqrt{n_u + 1} \sqrt{n_v + 1} (\cos \theta_h)^{n_u \cos^2 \phi_h + n_v \sin^2 \phi_h} \quad (3.4)$$

The rest of the BRDF becomes ρ_q . Next, we find f_q by factoring out any subexpression that does not rely on the canonical directions: $f_q = \sqrt{n_u + 1} \sqrt{n_v + 1}$. Then, we try to factor the remaining expression into factors, each of which is defined for some 1D parameterizations, γ . Note that once we find these factors, the same UCV cannot appear in more than one factor. A simple identity for exponents allows us to do exactly that (we name them α and β):

$$f_\alpha = (\cos \theta_h)^{n_u \cos^2 \phi_h} \quad (3.5)$$

$$f_\beta = (\cos \theta_h)^{n_v \sin^2 \phi_h} \quad (3.6)$$

The final step is to identify the 1D parameterization of all angle-dependent values. We must find an expression that does not involve the user-controlled variables, so we eventually obtain

$$\gamma_\alpha = \cos^{-1} \left((\cos \theta_h)^{\cos^2 \phi_h} \right); f_\alpha(\gamma_\alpha) = \cos^{n_u} \gamma_\alpha \quad (3.7)$$

$$\gamma_\beta = \cos^{-1} \left((\cos \theta_h)^{\sin^2 \phi_h} \right); f_\beta(\gamma_\beta) = \cos^{n_v} \gamma_\beta \quad (3.8)$$

The inverse-cosine was added because it is useful to think of the γ s as angles that range from 0 to $\frac{\pi}{2}$, but is not strictly necessary. The teapot in Figure 1.1 and Figure 3.4 uses this Ashikhmin-Shirley BRDF.

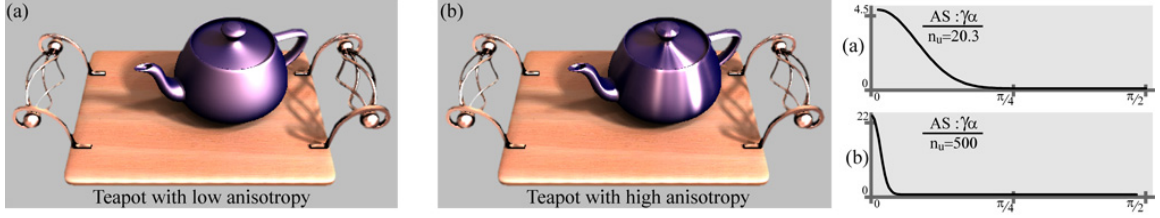


Fig. 3.4: **Anisotropic Edit on Teapot:** Anisotropic change using Ashikhmin-Shirley BRDF within our rendering system.

3.3 Making use of the quotient BRDF

The quotient BRDF (ρ_q) is important for preserving the complex 4D nature of BRDFs, while the 1D factors capture the *editable* component of the BRDF. It is incorrect to ignore the quotient BRDF simply because it is not editable. In addition to capturing static BRDF effects, it can be used in some cases to reduce the number of editable factors needed. Chapter 4 relies on a maximum of two editable factors, and Chapter 6 recommends a representation that uses only one editable factor.

3.3.1 Ashikhmin-Shirley with Fresnel edits in 2 factors

Most readers may wish to continue to the next subsection. Even though their original paper does not consider the index of refraction as one of the UCVs of the Ashikhmin-Shirley anisotropic model, it is implicit in the presence of the Fresnel term. It would seem that we therefore have a BRDF which requires three 1D factors:

$$\rho_{AS} = \rho_q f_q^{n_u n_v} f_\alpha^{n_u}(\gamma_\alpha) f_\beta^{n_v}(\gamma_\beta) F^n(\theta_d), \quad (3.9)$$

where n is the index of refraction, and the UCVs for each term have been noted as superscripts. The choice of θ_d for a third γ seems appropriate, but is unnecessary.

Expand $F^n(\theta_d)$ according to the Schlick approximation (1994):

$$\rho_{AS} = \rho_q f_q^{n_u n_v} f_\alpha^{n_u} f_\beta^{n_v} (R_n + (1 - R_n)(1 - \cos \theta_d)^5), \quad (3.10)$$

where $R_n = (n - 1)^2 / (n + 1)^2$. Distributing ρ_q and f_q gives us:

$$\rho_{AS} = f_\alpha^{n_u} f_\beta^{n_v} (\rho_q f_q^{n_u n_v} R_n + \rho_q f_q^{n_u n_v} (1 - R_n)(1 - \cos \theta_d)^5), \quad (3.11)$$

We now absorb the terms from the Schlick approximation to define $f_{q1}^{n_u n_v n} \equiv f_q^{n_u n_v} R_n$, and $f_{q2}^{n_u n_v n} \equiv f_q^{n_u n_v} (1 - R_n)$, and $\rho_{q2} \equiv \rho_q (1 - \cos \theta_d)^5$. This leaves us with a two-term definition for ρ_{AS} which shares the same editable factors among both terms:

$$\rho_{AS} = \rho_q f_{q1} f_\alpha f_\beta + \rho_{q2} f_{q2} f_\alpha f_\beta \quad (3.12)$$

This sharing of $f_\alpha f_\beta$ is useful since the user need not try to keep the two terms 'in sync' by editing a factor of one, and then editing a factor of the other to match. The practice of representing a BRDF as a sum of terms is acceptable under the following conditions: First, the number of UCVs, or unique editable factors should not increase to make the model unwieldy for editing. Second, when considering each UCV or editable factor independently, any relationship between terms must be explicit so that changes in one term automatically trigger corresponding changes in the other. This condition of explicit dependence is what we have in 9. Finally, the number of terms should be small (1-4) since simultaneous editing of many terms may eventually slow down the rendering engine. This is true of our complex lighting system, as well as GPU shaders where multi-pass rendering is common, but should be limited to a few passes. The same technique shown above for capturing the Fresnel term without introducing a new θ_d factor, can be used for other BRDF models.

3.3.2 Anisotropy with 1 Factor

Anisotropy is the result of an elongated highlight. As shown in section 3.2, two factors can be used to adjust the width of the highlight along the tangent and binormal directions. If we know the overall specularity of the material, we can separate the Ashikhmin-Shirley BRDF into a quotient that captures the width of the highlight, and a 1D factor that controls the ratio of the elongation in the two perpendicular directions:

$$\rho_{AS} = \rho_q (\cos \theta_h)^{n_u \cos \phi_h} (\cos \theta_h)^{(r n_u) \sin \phi_h} \quad ; \quad r \equiv n_v / n_u \quad (3.13)$$

$$\rho_{AS} = \rho_{qu}(\omega_i, \omega_o) (\gamma_r(\omega_h, \omega_o))^r \quad (3.14)$$

$$\gamma_r = (\cos \theta_h)^{n_u \sin \phi_h} \quad ; \quad \rho_{qu} = \rho_q (\cos \theta_h)^{n_u \cos \phi_h} \quad (3.15)$$

A similar single-factor form can be obtained if the amount and direction of anisotropy (r) is known, and only the width of the highlight needs to be adjusted.

3.4 Other Analytic and Hybrid BRDFs

The anisotropic Ward model (Ward, 1992) can be handled in essentially the same way as Ashikhmin-Shirley, above. For most analytic BRDFs, the factors, parameterizations, and UCVs can be found by inspection. Other examples with half angle and difference angle, $f_A(\theta_A)$ and $f_B(\theta_d)$, are Blinn-Phong (Blinn, 1977) and Schlick (Schlick, 1994).

In the Oren-Nayar model (Oren & Nayar, 1994), the dependence of terms on the canonical directions, ω_i and ω_o is static, though quite complicated. The UCVs serve only to define f_d as a scalar multiplier for these static terms. Using our representation, the 1D factors can be generated by evaluating the corresponding function, or by

explicitly tabulating the values through direct manipulation of the curves we have shown. In this way, any BRDF can be made into a hybrid between functional and data-driven. For example, the Oren-Nayar model can be separated into two factors using θ_{in} and θ_{out} as the parameterizations. The lack of UCVs defined over these factors does not preclude the original functions from being manipulated to generate new effects.

Ashikhmin et al. (2000) presented a hybrid BRDF model similar to Cook-Torrance that had a functional expression for all but the half-angle dependence. That factor was tabulated explicitly as a 2D map. Most plausible BRDFs required only a 1D map of θ_h , or two 1D factors: $f_A(\theta_h)$ and $f_B(\phi_h)$. Our model can use these tabulated factors without requiring an analytic expression. Editing would occur directly on the curves.

3.5 Measured/Factored BRDFs

We can use the same idea as seen in hybrid BRDFs to edit purely measured reflectance data. Isotropic factored BRDFs can be manipulated, by editing curves corresponding to each of the factors. We consider editing the recent homomorphic factorization (HF) of (McCool *et al.*, 2001). Specifically, the factorization is:

$$\rho_{HF} \simeq P'(\omega_i)Q'(\omega_h)P'(\omega_o) = \rho_q P(\theta_i)Q(\theta_h)P(\theta_o), \quad (3.16)$$

where for isotropic BRDFs, we are really editing 1D curves P and Q. (If the 2D terms are not symmetric because of noise or mild anisotropy, we can absorb that in the quotient, ρ_q). Figure 1.1 and Figure 3.5 use (McCool *et al.*, 2001)'s data for red velvet (factored from the CURET database (Dana *et al.*, 1999)). Figure 3.5 only edits

the P term, and therefore maintains the same feel of cloth (though a different type). Figure 1.1 shows the result of also editing the Q term, thus changing the quality to a more plastic appearance.

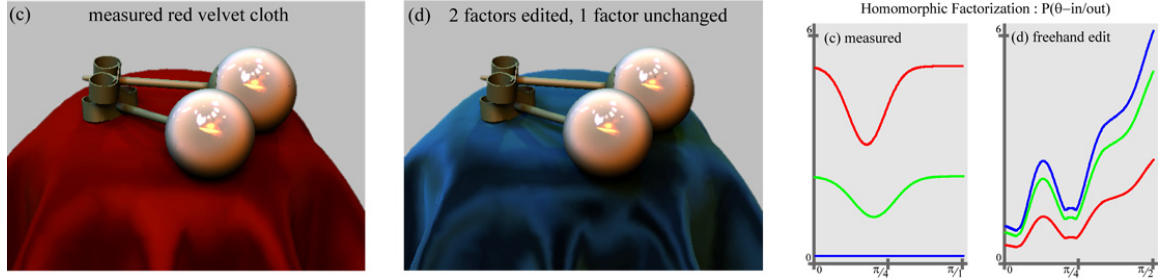


Fig. 3.5: **McCool Edit on Cloth**: Editing measured red velvet with freehand changes to the P factor of Homomorphic Factorized data.

We now consider fitting arbitrary, high-resolution measured BRDFs to our representation. Many dimensionality reduction techniques have been proposed for factoring measured BRDF data (Leung & Malik, 2001; Chen *et al.*, 2002; Vasilescu & Terzopoulos, 2004). However, not all methods of factorization are amenable to our editable representation. (Lawrence *et al.*, 2004) introduced a method using non-negative matrix factorization (NMF), to extract 1D factors with low reconstruction error. We collaborated with them on continued work in that area, and have now shown that all of the BRDFs from the database of (Matusik *et al.*, 2003) can be well approximated with the four 1D factors: θ_{half} , ϕ_{half} , θ_{diff} , and ϕ_{diff} . By using the Ruseinkewicz reparameterization, and a modified constrained least-squares (SACLS) method, the factors extracted have the desired property of non-negativity. SACLS is also able to separate the diffuse and specular terms of BRDFs that contain both. Unlike with general dimensionality reduction techniques, only a single set of factors is needed for each term, as opposed to a low-order sum of products. All of these properties mean that the extracted factors are meaningful as trends in the BRDF. The ϕ factors are

mostly constant, and the θ factors often resemble those of analytic models, such as Cook-Torrance. The added benefits of using real-world data is noticeable in the small variations from the clean lines of the analytic models - capturing the details of the acquired material's reflectance properties. Other irregular variations that don't align with any of the parameterizations can be absorbed in the quotient BRDF. We have found that the ϕ factors are rarely useful for editing, and absorb those into the quotient BRDF as well, leaving only the two θ factors. Having fewer factors also makes editing more intuitive, much like an abundance of UCVs in analytic models can be overwhelming. The factored BRDFs are then in the following form:

$$\rho_{factored} = \rho_d + \rho_s; \rho_d = \rho_{qd}G_d(\theta_i)H_d(\theta_o); \rho_s = \rho_{qs}G_s(\theta_d)H_s(\theta_h) \quad (3.17)$$

Figure 3.6 and Figure 3.7 show some of the 1D functions that form the factors of BRDFs from the Matusik database. Figure 3.6(a) shows a sample with considerable noise. If excessive noise exists in the measured data, it can be smoothed to produce a curve more similar to analytic forms. The curves can be edited in the familiar ways, as shown in Figure 3.6(b,c).

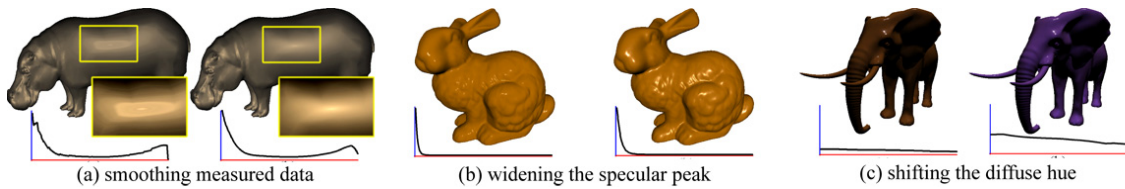


Fig. 3.6: Factored BRDF Edits: Edits with measured data similar to those possible with analytic models.

The exposure of the functions allows more freedom than available with a small number of parameters. Figure 3.7 shows some examples of localized edits, as well as freestyle artistic edits.

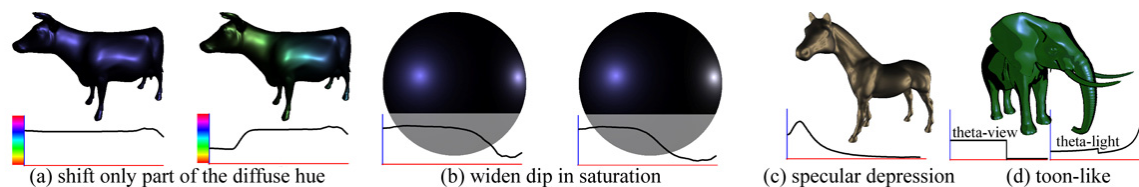


Fig. 3.7: **Measured BRDF Edits:** Edits not previously possible with analytic models

All of the images in Figure 3.6 and Figure 3.7 were generated with current GPU shader technology. They use simple lighting and do not include shadows. Our work (in conjunction with Jason Lawrence's) is the first to introduce the ability to edit measured reflectance even under these simple conditions. One of the reasons for showing this new family of edits with simple lighting is that the effects are quite different, and harder to isolate, in a complete scene with complex illumination. In order to see the results of such BRDF edits in the final scene, our real-time rendering engine must be used, described in Section 5. That system is equally capable of rendering analytic edits and measured edits, such as those seen here.

Section 9 details how the 1D functions shown throughout this work are edited. While computer graphics often fits splines to 1D functions, the operations and paradigms of these two models are incompatible, as explained in Section 9.

Part II

INTERACTIVE BRDF EDITING BEYOND SIMPLE LIGHTING

4. INTERACTIVE RENDERING IN COMPLEX LIGHTING

4.1 *Method Overview*

We use a precomputed approach to calculate all of the static scene-data and factor out the user-editable parts for render-time multiplication. Precomputed approaches to rendering rely on the linearity of the output image with respect to the editable inputs. In relighting, the exit radiance at a point is linearly dependant on the radiance of the lighting environment. We express this as follows:

$$R(x) = \sum_i T_i(x) L_i = \mathbf{T}(x) \cdot \mathbf{L} \quad (4.1)$$

where the radiance of each light basis, L_i , is weighted according to its contribution to the point being rendered, $T_i(x)$, and added to the contributions of all the other lights. This formulation assumes fixed geometry, material properties, and view. The natural 2D basis of the lighting – pixels in the environment map – is usually converted to some other orthonormal basis such 2D wavelets or spherical harmonics. As long as the transport vector, $\mathbf{T}(x)$, is also converted to the same orthonormal basis, the dot-product in equation 4.1 will remain valid. The transport vector encodes all of the static scene data which affects how much any given light contributes to the exit radiance at x . Notice that the lighting vector, \mathbf{L} , is common to all locations in the scene. If it had to be recomputed at each frame, for each pixel/vertex, the rendering operation would not proceed in real-time.

For BRDF editing, we wish to express our rendering operation as a similar dot-product, which we arrive at in equation 4.7, but replace the lighting vector with a user-editable BRDF vector. We assume fixed geometry, lighting, and view. In this chapter, we only render using direct lighting, since the image is not linearly dependent on the BRDF when considering interreflections. To express the outgoing radiance at a point, in the view direction, $R(x, \omega_o)$, we begin with the reflection equation:

$$R(x, \omega_o) = \int_{\Omega_{4\pi}} L(x, \omega_i) V(x, \omega_i, \rho(\omega_i, \omega_o)) S(\omega_i, \omega_o) d\omega_i, \quad (4.2)$$

where L is the lighting¹, V is the binary visibility function, and ρ is the BRDF, all expressed in the local coordinate-frame. In this frame, the BRDF is the same at each point, allowing edits of a single BRDF to be used over the whole surface. For now, S is the cosine-falloff term, $\max(\cos \theta_i, 0)$, but later it will include arbitrary (static) functions of the light and view directions, as expressed in the quotient BRDF, ρ_q , from equation 3.1.

Just as in Chapter 3, we consider a single term of a BRDF. A sum of terms (such as diffuse-like and specular lobes) can be handled separately and summed². Spatial variation is currently handled with a texture for any term that simply multiplies the corresponding image pixels. Editing spatial weights of the terms through texture painting is a natural extension, but we do not address it here. Different objects can have different BRDFs, and are treated independently. Finally, every operation is performed 3 times, once for each color channel. Our final scenes show a variety of materials, with complex multi-term BRDFs, color variations, and texturing.

We begin by substituting our representation from equation 3.1 into 4.2, with f_B

¹ We use environment maps as a convenient source of complex lighting for our sample scenes, but the mathematical development and our rendering system can handle arbitrary local lighting as well.

² Since this is simply a summation, they can also be edited simultaneously – such as modifying their relative weights to conserve energy

absorbed into ρ_q to simplify the exposition. (We later factor it out again, and deal formally with the two-factor version.)

$$R(x, \omega_o) = f_q f_o(\gamma_o(\omega_o)) \int_{\Omega_{4\pi}} L(x, \omega_i) V(x, \omega_i) S(\omega_i, \omega_o) f_A(\gamma_A(\omega_i, \omega_o)) d\omega_i \quad (4.3)$$

where we have absorbed ρ_q (now containing f_B) and $\max(0, \cos \theta_i)$ into S . As in standard precomputation techniques, we define the equivalent of a transport function, L' , as the product of all the fixed data:

$$L'(x, \omega_i, \omega_o) = L(x, \omega_i) V(x, \omega_i) S(\omega_i, \omega_o). \quad (4.4)$$

We use the name L' to suggest that it is a modified version of the lighting (specific to each location's visibility, normal, and static function). We now take the important step of expressing the BRDF factor (f_A) as a linear combination of J basis functions, b_j : The basis functions³ we choose are 1D Daubechies4 wavelets. Recall that γ_A is a 1D parameterization, enabling us to use a 1D basis to discretize f_A . Forming a discrete representation of the full 4D BRDF would substantially increase the system's memory requirements, making its implementation unfeasible. We will see later some of the benefits of using Daub4 over other 1D bases, such as 1D Haar or box functions.

Substituting equations 5.4 and 4.4 in equation 4.2, and pulling values not dependent on ω_i outside the integral,

$$R(x, \omega_i) = f_q f_o(\gamma_o(\omega_o)) \sum_j c_j \int_{\Omega_{4\pi}} L'(x, \omega_i, \omega_o) b_j(\gamma_A(\omega_i, \omega_o)) d\omega_i. \quad (4.5)$$

Using the fixed viewpoint assumption ($\omega_o = \omega_o(x)$), we define the light transport

³ We use the term basis functions loosely. We only need a set of functions whose linear combination approximates the BRDF. Specifically we do not make any extra assumptions about orthogonality.

coefficients $T_j(x)$, to be

$$T_j(x) = \int_{\Omega_{4\pi}} L'(x, \omega_i, \omega_o(x)) b_j(\gamma_A(\omega_i, \omega_o(x))) d\omega_i. \quad (4.6)$$

With a fixed view, f_o is just a texture that multiplies the final image ($f_o(\gamma_o(\omega_o(x))) = f_o(x)$), and f_q is a single scale factor over the whole image. We do not consider these lighting-independent factors further, as applying them to the final images is trivial. Substituting equation 4.6 into equation 4.5 allows us to express the outgoing radiance at a point x , in the direction $\omega_o(x)$, as the following dot-product:

$$R(x, \omega_o(x)) = \sum_j c_j T_j(x) = \mathbf{c} \cdot \mathbf{T}(x). \quad (4.7)$$

Equation 4.7 has the same form as precomputed relighting methods, except that we use BRDF coefficients c_j , instead of lighting coefficients. The user will manipulate the BRDF, either by editing curves (Chapter 4), or by varying analytic parameters. The 1D functions thus generated are then converted to the wavelet basis to obtain the coefficient vector \mathbf{c} . The dot-product in equation 4.7 is then performed at each location x , corresponding to each pixel in the image.

4.2 Environment Map Precomputation

We now consider the precomputation step, described by equation 4.6. It will be convenient to treat each pixel separately in what follows, so we can drop the dependence on spatial location x . To emphasize this, we write $L'(x, \omega_i, \omega_o(x)) = L'(\omega_i)$, where both x and $\omega_o(x)$ are fixed for a given pixel. We rewrite equation 4.6 with the above notational adjustments:

$$T_j = \int_{\Omega_{4\pi}} L'(\omega_i) b_j(\gamma(\omega_i)) d\omega_i. \quad (4.8)$$

Equation 4.8 takes the form of a projection. The lighting function, L' , is being projected onto the j th basis function. The 1D wavelet basis functions b_j , that are used for rendering, are better replaced with box functions for precomputation, since this makes the projection sparser and more efficient. It is a simple matter of using the fast wavelet transform to convert box coefficients to Daubechies4 coefficients at the end of precomputation, before rendering. Figure 4.1 shows examples of the bands formed by these box functions, when visualized on the sphere of incoming directions.

4.2.1 Computing Overlaps

This section details how we compute the overlaps $\Delta_{m,j}$ between the triangle light m and BRDF bin j , as introduced in equation 4.12. See Figure 4.1(c) for an example using $\gamma = \theta_r$.

First, we consider the overlap between a planar triangle and a region bordered by horizontal lines in the plane. In this case, the vertical or y -coordinate is the relevant parameter (e.g. θ_r). The key idea is that we only need this parameter at each vertex, which is known, given that a light's vertex represents a particular value of ω_i .

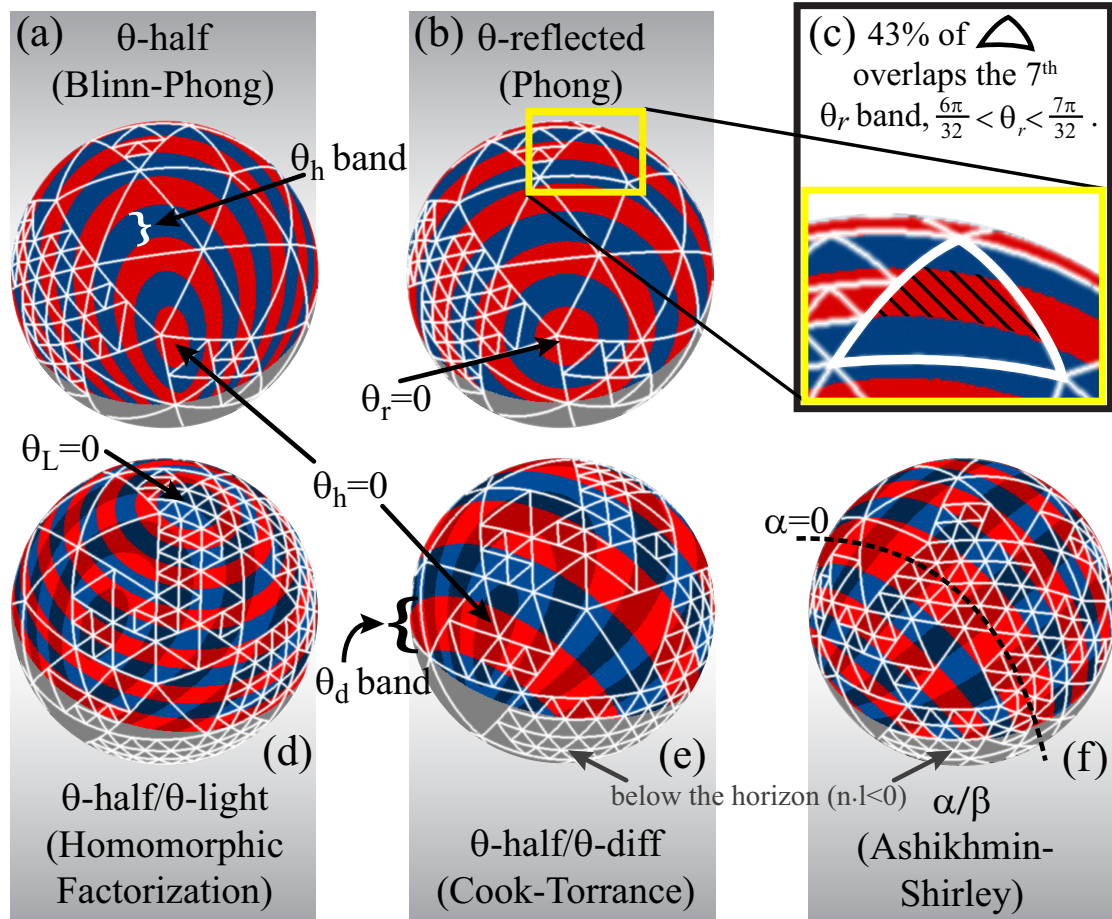


Fig. 4.1: **Light-Bands Overlaps:** The box functions of the BRDF parameterization create bands when visualized on the sphere of incoming directions. The triangular lights that approximate the environment are also shown to illustrate the overlaps that must be computed in equations 22 and 23. Note that the orientation of the bands depends on the image pixel (surface normal and view).

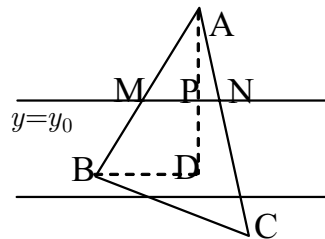


Fig. 4.2: **Triangle-Range Overlap:** Construction for finding the overlap percentage using only y-values.

The fraction of the area of δABC above a line $y = y_0$ is given by:

$$\frac{\delta AMN}{\delta ABC} = \frac{\sin A |AM| |AN|}{\sin A |AB| |AC|} = \frac{|AM| |AN|}{|AB| |AC|} \quad (4.9)$$

We construct δABD so that $y_D = y_B$. By similar triangles, and an analogous construction for the right half, we find that the fraction depends only on the y values (γ values) of the vertices:

$$\frac{AM}{AB} = \frac{AP}{AD} = \frac{y_A - y_0}{y_A - y_B}; \frac{\delta AMN}{\delta ABC} = \frac{(y_A - y_0)^2}{(y_A - y_B)(y_A - y_C)}. \quad (4.10)$$

We compute the percentage above and below the boundary lines of a band to find the overlap within the band. It is important to be able to calculate the overlaps based on a single coordinate because our parameterizations are 1D, and there is no notion of 'x' values for the vertices of triangular lights. Even in the simple Phong parameterization, $\gamma = \theta_r$, a perpendicular axis (possibly ϕ_r) does not exist, and would be ambiguous to define.

The remaining question is what to do when the assumption of local linearity in the parameterization breaks down (as it will for example, near the pole of the parameterization, causing the lines of constant γ to be curved, relative to the triangle.) To account for this, we simply subdivide a triangle into four smaller triangles. If the fractions of the area, added up from these four, agree with the coarser calculation to some tolerance, we consider the parameterization locally linear and stop there. Otherwise, we subdivide again recursively. Figure 4.1 shows the original subdivision of the lighting environment for one image pixel, and the reader may wish to find large triangles that would require subdivision, and smaller ones that would not.

4.2.2 Initial Approach

We now describe two simple ideas which seem like a straightforward solution, but turn out to be inappropriate for BRDF editing. In standard PRT, equation 4.8 simply involves a spherical harmonic or wavelet transform, where projection onto the basis is a well-studied operation. In our case, the box functions b_j use a different parameterization than the lighting, with γ implicitly depending on the local view direction, and therefore *changing* for each pixel. As seen in Figure 4.1, the box functions form highly irregular projections onto the sphere of lighting directions. To address this issue, one might imagine making a change of variables so that the lighting and the integral are also specified over the domain of γ , where b_j is a well-behaved box-function:

$$T_j = \int_o^1 L'(\gamma^{-1}(\omega_i)) b_j(\gamma) |Jac(\gamma^{-1})| d\gamma, \quad (\text{Initial.1})$$

where Jac is the Jacobian necessary for a multi-dimensional change-of-variables. It turns out that most parameterizations, γ , are not invertible, so Initial.1 cannot be evaluated analytically. Moreover, we do not want be restricted to a subclass of mappings that are invertible, and whose Jacobian can be written explicitly. Hence, we discretize equation 4.8 directly, without reparameterizing.

The straightforward approach for discretization is to reduce the continuous environment map into point samples. We can select a number of incoming directions, such as with structured importance sampling (Agarwal *et al.*, 2003), and replace the environment map with a light for each of these directions. The radiance of the solid angle around each light is preintegrated, and attributed to the directional light, thus allowing it to act as proxy for the illumination of its neighboring directions. Equation

4.8 now becomes:

$$T_j = \sum_{m=1}^M L'(\omega_m) \Delta_{m,j}, \quad (4.11)$$

where the environment is transformed into M discrete light sources at locations ω_m , and $\Delta_{m,j}$ is defined here, but redefined in equation 4.12:

$$\Delta_{m,j} = b_j(\gamma(\omega_m)), \quad (\text{Initial.2})$$

meaning that the energy of light m contributes to T_j if its location (center) falls inside band j .

Figure 4.3 shows the shortcoming of this approach. The difference between area-based illumination, and point samples is obvious for highly specular materials, such as the Phong sphere in Figure 4.3(d-f). Furthermore, the concentration of energy from large areas results in abnormally bright points where the environment is relatively dim (compare Figure 4.3(a vs. d)). A proposed solution is simply to increase the number of samples, but as seen in Figure 4.3(f), even 10,000 lights results in noticeable artifacts. We conclude that point lights are only viable for BRDFs that blur the environment with a kernel larger than the distance between any two such points.

4.2.3 Our Algorithm

The problem with point-sampled environment maps for the lighting is that, in editing BRDFs, we care about the angular distribution of energy as much as its strength. In order to retain this information, we use a basis of spherical triangles to approximate the lighting. Similar to the scheme used in Spherical Q2-Trees (Wan *et al.*, 2005), we hierarchically subdivide the sphere into polygonal domains, breaking each one into four children as required by the metric in (Agarwal *et al.*, 2003). However, we use triangles instead of rectangles, because they are more efficient, and also amenable to

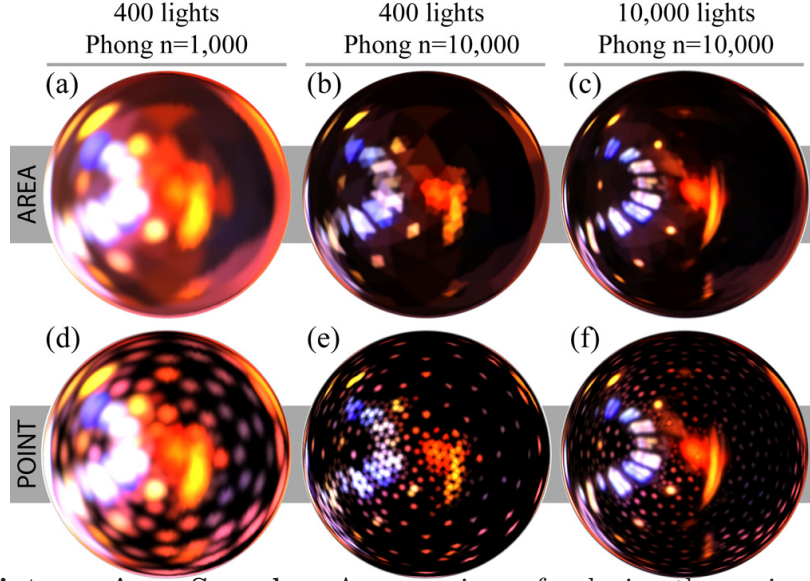


Fig. 4.3: **Point vs. Area Samples:** A comparison of reducing the environment to point lights (d-f), versus preserving the solid angles and representing the environment as triangular area lights (a-c).

the optimized overlap test described in Section 4.2.1.

Figure 4.1 shows how these triangle lights overlap several common parameterizations. We still evaluate the visibility, V , and quotient BRDF, S , at the light centers, as in equation 4.11. However, we use a more sophisticated technique for calculating the overlap (projection) between a given light and the bands created by the box functions of γ , as shown in Figure 4.1(c). Hence, we redefine $\Delta_{m,j}$ from equation 4.11:

$$\Delta_{m,j} = \frac{1}{\Omega_m} \int_{\Omega_m} b_j(\omega) d\omega \quad (4.12)$$

where Ω_m is the solid angle preintegrated for light m , and the integral in equation 4.12 sums to 1 when light m falls entirely within band j , and a fraction thereof otherwise. This overlap can be computed in constant time, as detailed in section 4.2.1.

By accurately computing angular overlaps, we are able to use fewer light samples to speed up precomputation. The final images are accurate up to the variation lost

in averaging the radiance over the area of the triangle, as seen in Figure 4.3(b vs. c). This should be contrasted with the precomputation in standard PRT relighting, which must consider all (usually about 25,000) lighting directions.

Efficiency: A direct implementation of equation 4.11 at each pixel loops over all lights m , for each j . The cost is therefore $O(JM)$, as it would be for most PRT relighting algorithms. We can reduce this to time linear in the number of light samples by recognizing that a triangle light m , overlaps (on average) a small number of box functions j . Therefore, we reorder the computation to first loop over lights m , and then over coefficients j that overlap m . (This inner loop now has essentially constant complexity). Moreover, we now need only compute visibility (by ray-tracing) once in the outer loop. Pseudocode for the optimized algorithm is shown in Code 1. This reduces the complexity to $O(M)$ at each pixel.

```

1  procedure computeT(pixel  $p$ , parameterization  $\gamma$ )
2    for  $m = 1$  to  $M$  do //loop over all lights
3       $V = \text{raytrace}(p, \omega_m)$ 
4      if ( $V == \text{blocked}$ ) continue
5       $S = \rho_q(p, \omega_m) * (N \cdot \omega_m)$  //quotient BRDF * cos  $\theta_i$ 
6       $L' = V * S * \text{preintegratedRadiance}(m)$ ; //equation 4.4
7       $\text{minBand} = \text{firstOverlap}(\omega_m, \gamma, p)$ 
8       $\text{maxBand} = \text{lastOverlap}(\omega_m, \gamma, p)$ 
9      //loop over relevant bands (usually only 2 or 3)
10     for  $j = \text{minBand}$  to  $\text{maxBand}$  do
11        $\Delta_{m,j} = \text{fractionalOverlap}(m, j, \gamma, p)$  //equation 4.12
12        $T[j] += L' * \Delta_{m,j}$  //equation 4.11
13     endfor
14   endfor
15 endprocedure

```

Code 1: **Precomputation:** Code for precomputing the values $T[j]$ in $O(M)$ time.

4.2.4 Two Factor BRDFs

We now detail the extension to two different curves and parameterizations for editing. We use the fact that a user can only edit one curve at a time, making the other curve temporarily ‘dormant’. Each curve or factor is approximated with a set of box functions. Using a 2-factor form of equation 5.4 through the derivations in Section 4.1, we arrive at a quadratic form of equation 4.7,

$$R = \sum_j c_j^A \sum_k c_k^B U_{jk} = (\mathbf{c}^B)^T \mathbf{U} \mathbf{c}^A, \quad (4.13)$$

where \mathbf{c}^A and \mathbf{c}^B are the coefficient vectors for the two curves, f_A and f_B , as in equation 3.1. The vector \mathbf{T} has been replaced with the matrix \mathbf{U} , defined by the 2D form of equation 4.8:

$$U_{jk} = \int_{\Omega_{4\pi}} L'(\omega_i) b_j(\gamma_A(\omega_i)) b_k(\gamma_B(\omega_i)) d\omega_i. \quad (4.14)$$

Recalling that equation 4.14 takes the form of a projection, we note that \mathbf{U} is a projection of the lighting onto the (non-orthogonal) axes of γ_A and γ_B ⁴. The discrete form is analogous to equation 4.11,

$$U_{jk} = \sum_{m=1}^M L'(\omega_m) (\Delta_{m,j} \Delta_{m,k}) \quad (4.15)$$

As in the one-curve case, we loop first over all the lights, and then find the overlapping bands for each parameterization.

The quadratic form in equation 4.13 cannot currently be evaluated directly in real-time. Instead, either the left or the right matrix-vector multiplication is evaluated in

⁴ For the case of $\gamma_A = \theta_i$ and $\gamma_B = \phi_i$, \mathbf{U} is the lighting, modulated by V and S

a run-time precomputation step we call *curve switching*. Let us say the user chooses curve A (Figure 4.5(a):Pearls θ_h) . Using the (temporarily fixed) values of \mathbf{c}^B for the 'dormant' curve (Figure 4.5(f):Pearls θ_d), we compute

$$\mathbf{T}_A = (\mathbf{c}^B)^T \mathbf{U}; R = \mathbf{T}_A \cdot \mathbf{c}^A, \quad (4.16)$$

where \mathbf{T}_A is the transport coefficient vector for standard 1D curve editing of curve A , given the fixed value of curve B . While not real-time, it takes only a few seconds to compute \mathbf{T}_A (shown in Table 4.2), making it a viable part of an interactive editing session. The second part of equation 4.16 to compute the actual image is the standard rendering dot-product, and one can now edit the first curve with real-time feedback. At some point, the user decides to *switch curves*, fixing \mathbf{c}_A while editing \mathbf{c}_B . The system now computes $\mathbf{T}_B = \mathbf{U}\mathbf{c}^A$, which again takes only a few seconds.

4.2.5 Use of Daubechies 4 Wavelets

low-pass=	[d1 d2 d3 d4]
high-pass=	[d4 -d3 d2 -d1]
d1=	0.4829629131445341
d2=	0.8365163037378077
d3=	0.2241438680420134
d4=	-0.1294095225512603

Tab. 4.1: **Daubechies 4 Coefficients:** Coefficients used for filtering primal band coefficients to wavelet coefficients.

The low-pass and high-pass filter coefficients for Daubechies 4 (Daubechies, 1988) wavelets we used are shown on the right. Like the Haar basis, Daub4 wavelets are orthonormal. Unlike Haar, they are designed for C1, not just C0 continuity.

To obtain the Daub4 coefficients of the user's input curve, we first discretize it

using the box basis functions. It is important to integrate over the domain of each box function (as opposed to taking point samples) in order to avoid popping artifacts during edits. So at each frame, the following operation happens once for each edited curve:

$$c_j = \int_{\frac{(j-1)\pi}{2J}}^{\frac{j\pi}{2J}} b_j(\gamma) f(\gamma) d\gamma, \quad (4.17)$$

where we've assumed $\gamma \in [0, \frac{\pi}{2}]$, as it usually is.

Using column vectors, we use the synthesis matrix for Daub4, \mathbf{S}_{D4} , to write the wavelet transformed versions of \mathbf{c}^A and \mathbf{c}^B :

$$\mathbf{c}_{D4}^A = \mathbf{S}_{D4} \mathbf{c}^A; \mathbf{c}_{D4}^B = \mathbf{S}_{D4} \mathbf{c}^B \quad (4.18)$$

To find the properly transformed \mathbf{U} , \mathbf{U}_{D4} , we refer to equation 4.13:

$$R = (\mathbf{c}_{D4}^B)^T \mathbf{U}_{D4} \mathbf{c}_{D4}^A; \mathbf{U}_{D4} = \mathbf{S}_{D4}^{-T} \mathbf{U} \mathbf{S}_{D4}^{-1} = \mathbf{S}_{D4} \mathbf{U} \mathbf{S}_{D4}^T, \quad (4.19)$$

where the last equality uses the fact that the Daubechies 4 synthesis matrix is orthonormal. Therefore, we must use the standard 2D transform of the matrix \mathbf{U} , which first uses the 1D transform to convert the rows, and then uses it to convert the columns (or vice-versa). Though the non-standard 2D transform often provides better compression, it is not applicable for our use.

4.3 BRDF Editing Results

We first show some BRDF edits that are possible in real-time, with complex lighting and shadows. We then evaluate the performance of our precomputation and rendering methods quantitatively.

4.3.1 Real-time Edits

While this part of the thesis focuses on the rendering algorithms, rather than specific editing techniques, our experience has been that one can quickly create a variety of effects and precisely choose material properties with our system - tasks that would often be difficult without final-placement, real-time feedback. Figure 4.4 and Figure 4.5 show a number of edits using our system, that indicate some of the richness of effects that can be produced. We will refer to these as Earrings and Teatray, respectively.

Specular highlight adjustment: One basic edit, shown in Earrings(a-b) and in Teatray(c), is adjusting the width of a specular highlight, by modifying analytic parameters like surface roughness (σ), which in turn specifies the half-angle distribution $f(\theta_h)$. Compare the appearance of the specular BRDF in complex lighting for the handles of the Teatray(d) to the same BRDF in simple lighting (d.1). Complex lighting adds the richness of encompassing illumination that is lost under a point source. Note how the two main light sources blend together in Earrings(a), and are made distinguishable in Earrings(b). Finding the right σ for a desired level of interaction between different parts of the environment can only be accomplished with interactive feedback.

Anisotropic highlights: We can also edit anisotropic BRDFs like Ashikhmin-Shirley (see Teatray). When comparing the teapot in Teatray(b) to Teatray(b.1), we see that

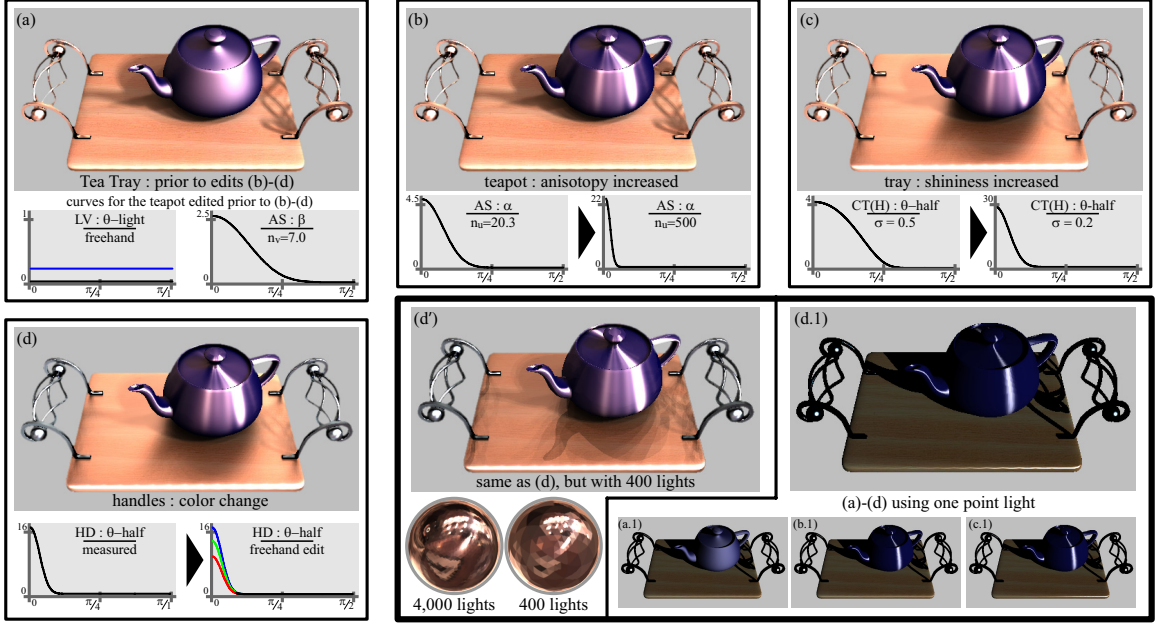


Fig. 4.4: Teatray Results: The Utah teapot on a wooden tray before (a) and after (d) an editing session with our system. The scene starts with a teapot rendered using an Ashikhmin-Shirley specular term + LV diffuse term on a tray with a Cook-Torrance specular term + LV with wood-grain texture term. The handles use the HD parameterization with nickel data from the Matusik database. The illumination is a 4,000 light approximation of the Galileo HDR probe. After setting initial values for the teapot (a), the user switches from editing the b curve, to editing the a curve of the AS model. The anisotropy is increased (b), giving a brushed look. Next (c), the tray's specular term is made sharper. For a planar surface, the most noticeable effect of this edit is a change in the shadows. Now that the diffuse term is less prominent (as evidenced by the washed out texture), the shadows caused by lights in the reflection direction are more visible than those in the incident direction. Finally, the user notices that even though the data loaded for the handles is that of nickel, they appear golden due to the overall color of the environment. To regain the feel of a silvery metal, the user edits the left portion of the curve with a freehand adjustment (d) to offset the coloration given by the environment. Each of these operations is compared under a point source (a.1-d.1), and it is clear that the feel of the final materials cannot be understood by making edits in such a different light setting. The strength and position of the point light were set by hand to match the most noticeable aspects of the environment. Finally, we compare (d) with the same image rendered using only 400 area lights (d'). The overall look and feel are preserved, with the most noticeable difference being a loss of softness in the shadows.

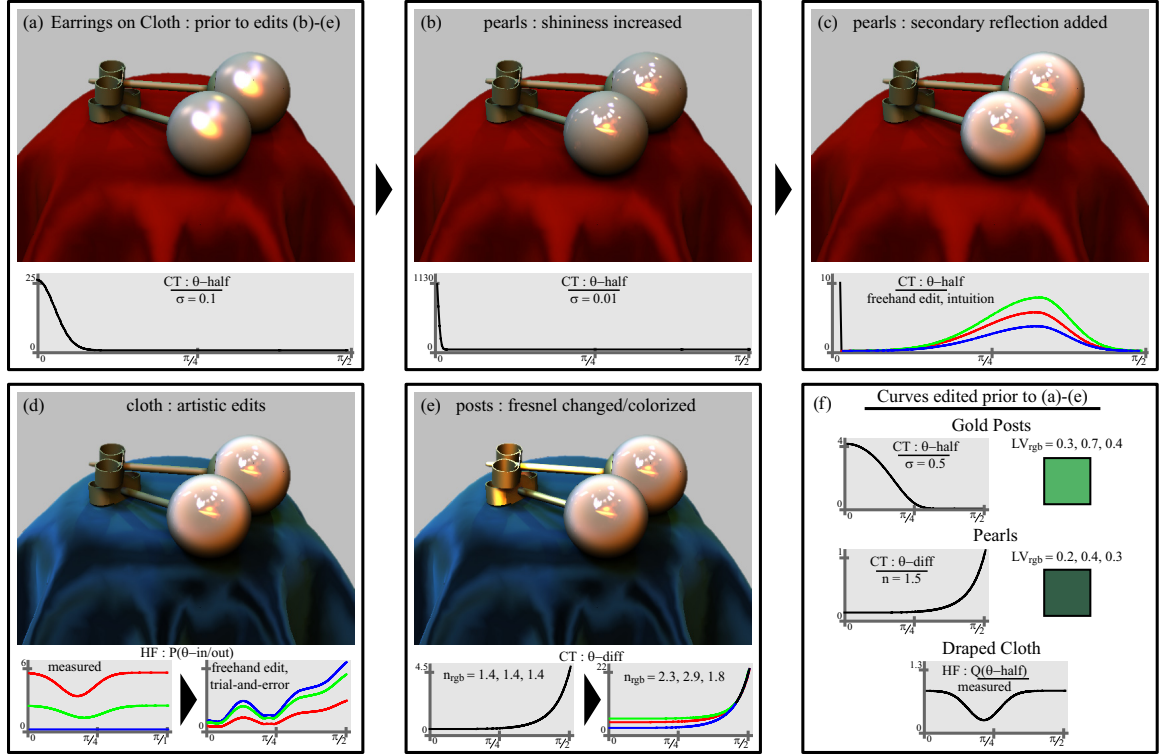


Fig. 4.5: Earrings Results: A sample editing session shows before (a) and after (e) of a scene with pearl earrings on a cloth draped over a pedestal, as illuminated in Grace Cathedral. The pearls and posts use a Cook-Torrance specular term + LV diffuse term, and the cloth uses homomorphic factorization. The session begins by setting some initial values for the UCVs (f), and loading data for red velvet, based on the factorization presented in [McCool et al. 2001]. First (b), the pearls are given sharper reflections by decreasing σ , which in turn defines the curve. The reader is encouraged to compare this with a real pearl and notice that indeed, the reflections are near mirror-like. The ‘hazy’ effect of pearls comes from a secondary reflection, and this is added in (c) by adjusting the curve with a freehand edit. In (d), the user plays around with the shape of the function, and arrives at a desirable blue cloth material. Finally (e), the index-of-refraction for the Fresnel term of the posts is set to give them a metallic gold appearance.

the point-lit teapot is much darker. This is not due to a dimmer light, but rather to the fact that the point light interacts only with surfaces oriented in the specular direction, relative to the viewer. By contrast, the full complex illumination affects the entire surface of the teapot. If we attempt to set n_u under a point source, we would be misled to believe that a larger value creates a darker appearance for the teapot.

Retroreflection and the Fresnel effect: Earrings(e) shows how changing the index of refraction at each color channel generates a new set of curves for $f(\theta_d)$, according to the Fresnel equation. Values of $f(\theta_d)$ near $\theta_d = 0$ control retroreflection, since the view and light directions are close to each other. As seen in the separate RGB curves, the color distinction is large near this part of the curve, and dissipates as θ_d grows. This will give the desired Fresnel effect of desaturating the color of lights near grazing angles.

Interaction of Materials and Shadows: Shadows can often depend on the material properties, since a diffuse BRDF draws lighting energy from a different area than a glossy BRDF. In Teatray(c), we edit the tray’s material from diffuse to glossy, which also allows us to create different shadowing effects. Notice that occluding geometry in the reflection direction is now much more important to the creation of shadows than in the diffuse case, Teatray(b).

Artistic edits: We can also go beyond physically-based BRDFs, to create flexible artistic effects. Earrings(c) shows how intuition about a secondary reflection can be used to edit a portion of a curve otherwise specified by the user’s choice of σ . In Earrings(d), we used trial-and-error to manipulate a curve loaded from factored data in order to design a desirable material. A simpler type of artistic license is used on the handles in Teatray(d), where we compensate for the color of the environment to force the silver appearance.

4.3.2 Precomputation Numbers

Table 4.2 analyzes the time and memory usage for the precomputation phase, for the Earring and Teatray scenes, itemized for each object. Since each pixel is treated separately, the totals are proportional to the number of pixels, and would be comparable for any scene with the same (512x512) image resolution. These tests were run on an Intel Xeon 3.2Ghz 64bit processor with 8GB of RAM.

We proceed from left to right in Table 4.2. For each object, we show the BRDF model used. In many cases we use a sum of diffuse and specular lobes, such as LV + CT for the posts, and LV + AS for the teapot. We typically use 128 or 256 bands to accurately represent specularities. In some cases, lower resolutions of 32 or 64 are used for in HD or CT representations. For two-curve models, the time to switch curves for editing is only one or two seconds.

To reduce storage of the transport coefficient vectors and matrices, we zero out values below a small threshold and do not store them explicitly. Unlike relighting, we must be much more careful about compression since the BRDF is visualized directly, and we are able to achieve compression rates of only 5:1 - 6:1 (and only about 2:1 in some cases). Even so, our final storage requirements, while large (hundreds of megabytes), are comparable with all-frequency relighting PRT methods. Most of the storage is for two-curve BRDF models, where we must store transport matrices U at each pixel (for example, AS in teapot, CT in pearls and HF in cloth). The storage for transport vectors from one-curve BRDF models like LV is usually much smaller.

We separate the precomputation time into visibility and our precomputation. The visibility is precomputed once for the scene, using a standard ray-tracer. The band-light triangle overlap and fixed BRDF shading computations are performed as per Code 1. The time for both computations is approximately linear in the number of

lights and pixels. For a reasonable 400 triangular light approximation, we require only 2-4 minutes, making quick previewing of a variety of different scenes possible. As shown in Figure 4.4(d), our robust area-preserving method for precomputing with triangular lights allows a small number of lights to be used with only a small loss in the quality of rendered images. If precomputation time is not a bottleneck, we can obtain higher quality approximations with thousands of lights, as we did for the figures in this chapter, while still only needing 15-30 minutes. Further optimization of the ray-tracer would yield even faster precomputation.

object	pixels	model	bands	curve	storage (MB)		precompute (min)		visibility (min)	
				switching	raw	compressed	4000 lts	400 lts	4000 lts	400 lts
Tray	40,529	LV	256	NA	121	36.3 (0.3)	1.5	0.2	6.6	0.7
		CT(H)	256	NA	243	121.5 (0.5)	1.6	0.3		
Teapot	17,709	LV	256	NA	53	15.9 (0.3)	0.7	0.1	2.8	0.3
		AS	128/128	2.1 sec	3,400	408 (0.1)	1.7	0.2		
Handles	8,629	HD	256/32	0.7 sec	330	60.2 (0.2)	0.8	0.1	1.4	0.2
Figure 14	66,867				4,147	641.9(0.15)	6.3 min	0.9	10.8 min	1.2
Pearls	22,505	LV	256	NA	4	2.3 (0.6)	0.9	0.1	3.9	0.5
		CT	256/64	2.3 sec	1,464	248.9 (0.2)	2.6	0.3		
Posts	7,626	LV	256	NA	2	0.9 (0.5)	0.3	0.1	1.3	0.2
		CT	256/64	0.3 sec	92	18.4 (0.2)	0.8	0.2		
Cloth	98,890	HF	128/128	1.8 sec	1,186	260.9 (0.2)	6.3	0.8	18.8	2.1
Figure 15	129,021				2,748	531.4(0.19)	10.9 min	1.5	24 min	2.8

Tab. 4.2: **Timings and Memory for Complex Lighting:** The precomputation time and memory overhead for each object in the two scenes, Earrings and Teatray. The images are rendered at 512x512, with a fixed view. Notice that the compression ratios are somewhat low because we cannot afford to compress aggressively, as mentioned in Section 7.2. Also note that even though the memory overhead grows geometrically with the dimensionality (1D vs. 2D), the precomputation time only grows linearly. CT(H) refers to a fixed Fresnel factor for the Tray.

5. BILINEAR RENDERING OPERATORS FOR BRDF EDITING

5.1 *Theory of BRDF Editing in Global Illumination*

In this Chapter, we develop a precomputation-based method for interactive BRDF editing with global illumination (see results in Fig. 6.1). The main challenge arises from the fact that final scene radiance (an image) is *not even linear* in the objects' BRDFs. It is well known that albedo, or more generally a BRDF, has a non-linear effect because it multiplies the light at each bounce. We develop a higher-order representation of the image as a function of the scene's BRDFs. We will show in Chapter 6 how to precompute a tensor at each pixel, fixing the lighting and view for a static scene, but leaving the BRDFs unspecified until runtime, when they can be edited.

We first describe a general theoretical framework for BRDF editing, based on a bilinear formulation of the reflection operator, that extends the linear operator formulation of rendering (Arvo *et al.*, 1994). We show how the precomputed matrix of previous methods must be extended to a multi-bounce tensor of polynomial coefficients. The operator notation is compact, and alleviates the need for complex integrals in the equations. The derivations in the chapter are independent of any specific implementation. In Sec. 5.2, we discuss our approximations within this framework, that make the computation tractable.

The full multi-bounce tensor is a complete representation of the image as a func-

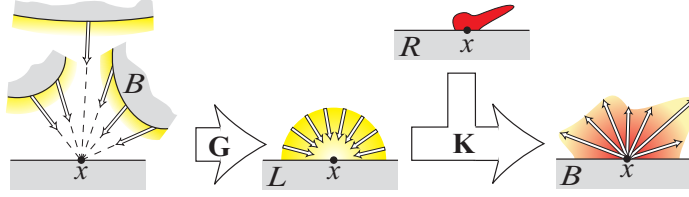


Fig. 5.1: Rendering Operators: Schematic of the rendering equation. Outgoing radiance from all surfaces in the scene (left) is converted by the \mathbf{G} operator into a local incident radiance field at each location (middle), which is then reflected via the bilinear reflection operator \mathbf{K} , that takes as input both the incident lighting and the BRDF.

tion of scene BRDFs, but is computationally too expensive to treat in full generality. We consider frequency characteristics, developing a tractable approximation that preserves most important perceptual effects (Sec. 5.2). Specifically, the first bounce from the eye usually uses the full BRDF, to capture glossy reflections, while subsequent bounces use a lower-frequency approximation to capture overall shading effects. Within this general framework, we show two possibilities—where further bounces (up to the fourth-bounce) are treated as purely diffuse (Figs. 6.1, 5.5 and 6.3), and where additionally, the second bounce from the eye uses a lower-frequency approximation to achieve accurate indirect reflections in glossy surfaces (Fig. 5.3), or even intricate effects like caustics (Fig. 6.5).

The diagram shows a sequence of basis BRDFs (yellow and blue) being combined into a polynomial. The equation is:
$$\langle \cdot \rangle = \langle \cdot \rangle c_1^1 + \langle \cdot \rangle c_2^1 + \langle \cdot \rangle c_1^1 c_1^2 + \langle \cdot \rangle c_2^1 c_1^2 + \langle \cdot \rangle c_1^1 c_2^2 + \langle \cdot \rangle c_2^1 c_2^2 + \langle \cdot \rangle (c_1^1)^2 + \dots$$
Each term is represented by a small diagram showing the interaction of light rays with the surfaces.

Fig. 5.2: BRDF Polynomial: The final value of each pixel is a polynomial in the BRDF coefficients. Here we show an example with 2 surfaces and two basis BRDFs shown in yellow (diffuse and specular). Note that the BRDFs we use in practice are different. The combinatorics of multivariate polynomial coefficients make a tensor notation particularly useful.

$B(x, \omega_o)$	Outgoing radiance (image)
$E(x, \omega_o)$	Emissive radiance of light sources
$L(x, \omega_i)$	Local incident radiance
$R(x, \omega_i, \omega_o)$	BRDFs of all points in the scene
$T^N(x, \omega_o)$	Precomputed multi-bounce tensor
$\rho^m(\omega_i, \omega_o)$	BRDF of object m
$b_j^m(\omega_i, \omega_o)$	Basis function j for the BRDF of object m
$H^m(x)$	Spatial weight map or texture for object m
\mathbf{G}	Linear geometric operator
	$\mathbf{G} : B(x, \omega_o) \mapsto L(y, \omega_i)$
	$(\mathbf{G}B)(x, \omega) \equiv B(x'(x, \omega), -\omega)$
$\mathbf{K}(R)$	Reflection operator (equation 5.2)
c_j^m	BRDF coefficients (equation 5.4)
d_m	Equivalent albedo of object m (section 5.2.4)
\bar{d}_z	Product of albedos d_m
\vec{X}_i	Light path with contribution $f(\vec{X}_i)$
F_{jn}^i	Tensor coefficient after freezing BRDFs
J	Number of BRDF bases (usually 64 or 128)
M	Number of objects ($M \sim 5$)
W	Total basis functions ($W = JM \sim 500$)
Z	Number of terms for diffuse \bar{d}_z ($Z \sim 64$)

Tab. 5.1: **Table of Notation for Chapters 5 and 6:** This table give the notation used for developing the global illumination solution to BRDF editin, presented in Chapters 5 and 6

5.1.1 Basic Framework using the bilinear \mathbf{K} operator

We begin by writing the rendering equation (Kajiya, 1986) as a linear operator equation, similar to (Arvo *et al.*, 1994):

$$B = E + \mathbf{K}(R) \mathbf{G}B, \quad (5.1)$$

where $B(x, \omega_o)$ is the outgoing surface radiance, $E(x, \omega_o)$ is the emission and \mathbf{G} is the linear geometric operator that converts the outgoing radiance from distant surfaces to a local incident radiance field as in (Arvo *et al.*, 1994). Figure 5.1 shows a schematic,

and Table 5.1 summarizes notation.

Arvo et al. (1994) define \mathbf{K} as a linear reflection operator on the local incident radiance field L . In our first departure from previous representations, we make explicit \mathbf{K} 's dependence on the BRDFs R of scene objects. We write \mathbf{K} as a bilinear operator that takes an additional input $R(x, \omega_i, \omega_o)$ which describes the BRDF at each point in the scene and is the kernel of integration in \mathbf{K} ,

$$\begin{aligned} \mathbf{K} : L(x, \omega_i), R(x, \omega_i, \omega_o) &\mapsto B(x, \omega_o) \\ (\mathbf{K}(R) L)(x, \omega_o) &\equiv \int_{\Omega_{2\pi}} R(x, \omega_i, \omega_o) L(x, \omega_i) \cos \theta_i d\omega_i. \end{aligned} \quad (5.2)$$

Note that \mathbf{K} is bilinear, or linear with respect to *both* inputs—incident lighting L , and the BRDFs of objects in the scene R . That is, for scalars a and b ,

$$\begin{aligned} \mathbf{K}(aR_1 + bR_2) L &= a\mathbf{K}(R_1) L + b\mathbf{K}(R_2) L \\ \mathbf{K}(R) (aL_1 + bL_2) &= a\mathbf{K}(R) L_1 + b\mathbf{K}(R) L_2. \end{aligned} \quad (5.3)$$

We now seek to relate R (and hence \mathbf{K}) to editable BRDFs of individual objects. We assume there are M objects in the scene, and for now that each object has a single BRDF. Let object m have¹ BRDF ρ^m . We assume the BRDF can be represented as a linear combination of functions over the domain (ω_i, ω_o) ,

$$\rho^m(\omega_i, \omega_o) = \sum_{j=1}^J c_j^m b_j^m(\omega_i, \omega_o). \quad (5.4)$$

The BRDF basis functions b could be spherical harmonics, wavelets or any other linear basis. We follow previous BRDF editing methods (Ben-Artzi *et al.*, 2006b;

¹ We use superscripts to denote some property of the function, and parentheses for explicitly raising to a power, so ρ^2 is the second in a series, whereas $(\rho)^2$ is ρ squared.

Lawrence *et al.*, 2006), that have used box functions over a suitable 1D parameterization such as the half-angle, as described in Chapter 3. They have shown that a 1D parameterization is appropriate for most BRDF edits, as well as being compatible with parametric edits of most common BRDF models.

Our goal is to use these basis BRDFs to create a method that allows us to alter the kernel of integration in the \mathbf{K} operator by specifying different weights c_j^m . We first need to use the b_j^m s to describe R over all surfaces. In order to encode per-object BRDFs, we define a surface mask $H^m(x)$ that is 1 if x is on object m , and 0 otherwise.²

$$R(x, \omega_i, \omega_o) = \sum_{m=1}^M H^m \rho^m = \sum_{m=1}^M H^m(x) \sum_{j=1}^J c_j^m b_j^m(\omega_i, \omega_o) \quad (5.5)$$

The super/subscripts in the above equation implicitly define basis functions for the full spatially varying R ,

$$R = \sum_{m=1}^M \sum_{j=1}^J c_j^m R_j^m; \quad R_j^m(x, \omega_i, \omega_o) = H^m(x) b_j^m(\omega_i, \omega_o). \quad (5.6)$$

For simplicity, we will often use a single index w (or u or v) to refer to the double script j^m , with $w \in [1, W] : W = MJ$.

5.1.2 Polynomial Representation for Multi-Bounce

The solution of equation 5.1 can be expressed as the expansion

$$B = E + \mathbf{K}(R) \mathbf{G} E + \mathbf{K}(R) \mathbf{G} \mathbf{K}(R) \mathbf{G} E + \dots \quad (5.7)$$

² H can also take on non-binary values to encode spatial weight maps for combining BRDFs (Lensch *et al.*, 2001; Lawrence *et al.*, 2006), and/or for describing textures.

where each term N has an intuitive interpretation, as corresponding to N bounces of light from source(s) to viewer.

All current relighting methods rely on the linearity of B with respect to E . Previous BRDF editing methods also take advantage of the linearity of the 1-bounce term in B (i.e., $\mathbf{K}(R)\mathbf{G}E$) with respect to \mathbf{K} (and hence with respect to R), requiring them to render using only direct lighting.

However, *this linearity no longer applies* for BRDF editing with global illumination because the operator $\mathbf{K}(R)$ is applied multiple times. Even for 2-bounce reflections, the system becomes quadratic, and must be represented with a quadratic multivariable polynomial in the c_w s. The N -bounce solution is an order N polynomial. We now show how to extend the general PRT approach to these polynomial equations. We start by considering 2-bounce reflection,

$$B^2 = \mathbf{K}(R)\mathbf{G}\mathbf{K}(R)\mathbf{G}E \quad (5.8)$$

$$= \mathbf{K}\left(\sum_u c_u R_u\right)\mathbf{G}\mathbf{K}\left(\sum_v c_v R_v\right)\mathbf{G}E \quad (5.9)$$

$$= \sum_u c_u \left(\mathbf{K}(R_u)\mathbf{G} \sum_v c_v (\mathbf{K}(R_v)\mathbf{G}E) \right) \quad (5.10)$$

$$= \sum_u \sum_v c_u c_v \mathbf{K}(R_u)\mathbf{G}\mathbf{K}(R_v)\mathbf{G}E. \quad (5.11)$$

The bilinear nature of \mathbf{K} is crucial here. We use the linearity of \mathbf{K} with respect to the BRDF to get equation 5.10, and the linearity of \mathbf{K} and \mathbf{G} with respect to radiance to get equation 5.11.

For BRDF editing, the coefficients (c_u and c_v) become the variables of our computation. The fixed quantities for precomputation are the basis BRDF distributions (R_u and R_v), that depend only on the parameterizations of the various BRDFs. \mathbf{G} and E are also known, defined by the geometry and lighting of the scene, respectively.

We precompute a 2-bounce transport function T_{uv}^2 and calculate B^2 as

$$\begin{aligned} T_{uv}^2 &= \mathbf{K}(R_u) \mathbf{G} \mathbf{K}(R_v) \mathbf{G} E \\ B^2 &= \sum_u \sum_v T_{uv}^2 c_u c_v. \end{aligned} \quad (5.12)$$

Most generally, T_{uv}^2 and B^2 are defined over all spatial locations and view directions. In practice, since we fix the view, $T_{uv}^2(x, \omega_o(x))$ is an order 2 tensor (that is, a matrix for 2-bounce reflection) at each pixel. $B^2(x, \omega_o(x))$ at each pixel is a quadratic polynomial in the variables c , with coefficients given by T^2 . When evaluated, it becomes the radiance function for the scene due to light that has reflected off exactly 2 surfaces. More explicitly,

$$B^2 = T_{11}(c_1)^2 + T_{12}c_1c_2 + \dots T_{uv}c_uc_v + \dots T_{WW}(c_W)^2. \quad (5.13)$$

Figure 5.2 illustrates such polynomials for paths of length 1 and 2. All 1- and 2-term combinations of the BRDFs are possible, including repetitions of the same basis function, as in $T_{11}(c_1^1)^2$, since concave objects can reflect onto themselves. Following the same logic, the N -bounce energy is

$$B^N = \sum_{w_N} \sum_{w_{N-1}} \dots \sum_{w_1} T_{w_N w_{N-1} \dots w_1}^N c_{w_N} c_{w_{N-1}} \dots c_{w_1}, \quad (5.14)$$

where T^N is an order N tensor for each pixel, whose size varies with the number of objects and BRDF basis functions per object. We are evaluating a multi-variable, degree N polynomial where each w runs over all W values (all basis functions). The

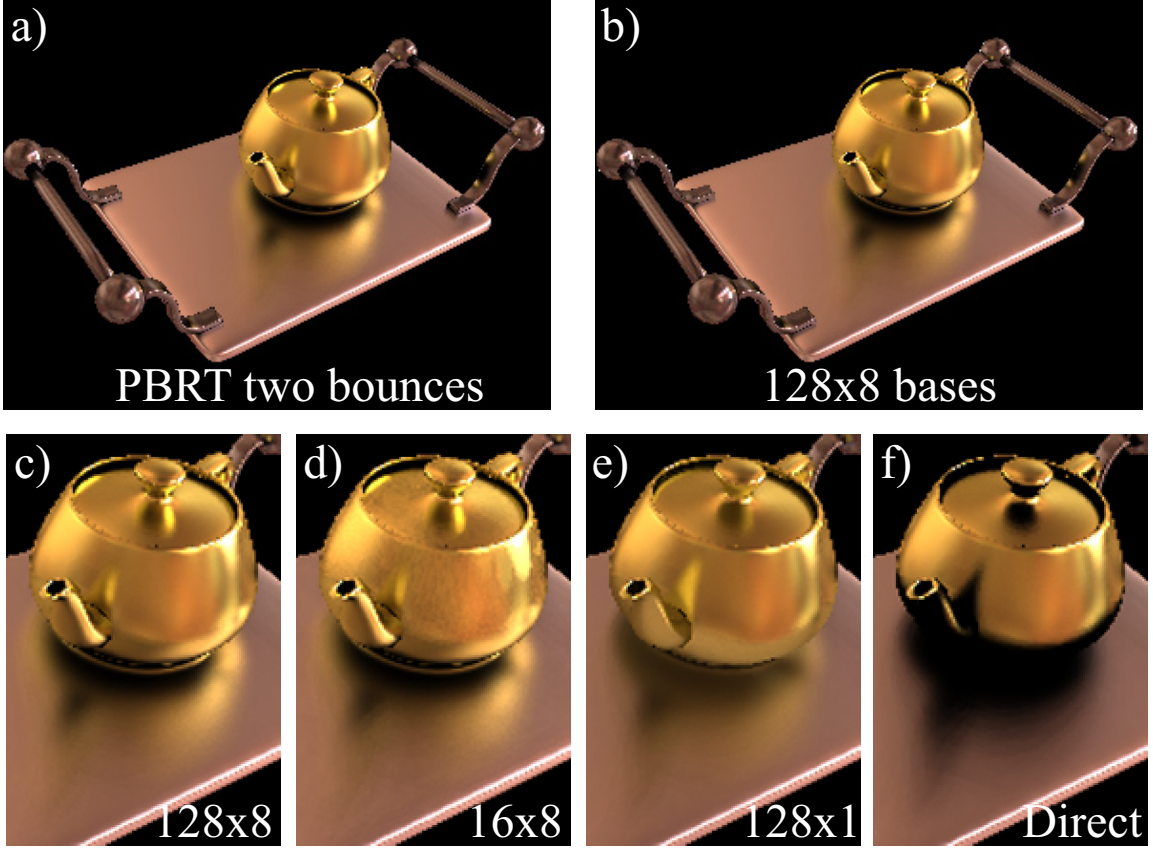


Fig. 5.3: Comparison of Bounce Resolution: An evaluation of the accuracy of different two-bounce decreasing-frequency BRDF series, precomputed and rendered in our editing system. (a) Full resolution BRDFs rendered offline in PBRT—hence a series of (∞, ∞) (b) A $(128, 8)$ series (c) $(128, 8)$ (d) $(16, 8)$ (e) $(128, 1)$ or diffuse for second bounce from the eye (f) Direct lighting only (Ben-Artzi et al. 06). We see that a very low-frequency second-bounce BRDF approximation $(128, 8)$ in (b) and (c) is essentially exact. Moreover, even a diffuse second bounce approximation $(128, 1)$ in (e) provides the correct shiny material perception of the tray and indirect reflections of tray and teapot. By contrast, direct lighting shows only a black shadow of the teapot on the shiny tray, and the spout and handle do not reflect in the teapot.

variables of are the unknown scalar c 's. The coefficients are stored in T^N ,

$$T_{w_N w_{N-1} \dots w_1}^N = \mathbf{K}(R_{w_N}) \mathbf{G} \mathbf{K}(R_{w_{N-1}}) \mathbf{G} \dots \mathbf{K}(R_{w_1}) \mathbf{G} E. \quad (5.15)$$

Finally, we construct the image to be displayed by adding the contributions from

the different path-lengths:

$$B = E + B^1 + B^2 + \dots + B^{\overline{N}}, \quad (5.16)$$

where we cut off the expansion in equation 5.7 to $\overline{N} + 1$ terms.

5.2 Managing Complexity

Section 5.1 has presented a theoretical framework that is fully general. However, we need to manage complexity in order to develop a tractable algorithm. In particular, the number of terms in equation 5.14 is $(W)^N$. Recall that W is already JM , the number of basis BRDFs times the number of objects. For typical values of M (5) and J (64-128), (so $W \sim 500$) the computational and storage cost for all but the first bounce become prohibitive. In this section, we derive efficient and perceptually accurate approximations.

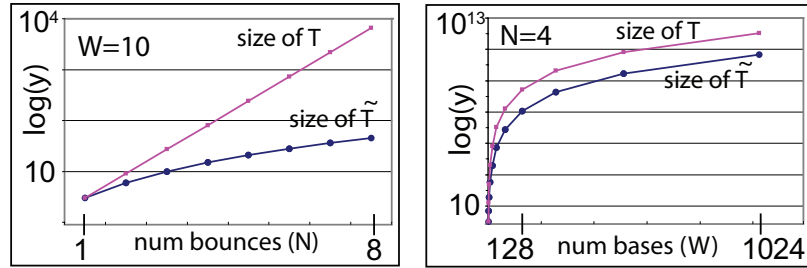


Fig. 5.4: Exponential Growth wrt BRDF Bases: Rate of growth for the precomputed data-structure, T with and without taking symmetry into account. Left: The base of the exponential growth is effectively reduced due to symmetry (smaller slope). Right: The slope of both growth rates is the same; symmetry offers only a constant offset, corresponding to a scale factor (90 here) for the same asymptotic behavior.

Some efficiency is obtained by taking advantage of symmetry in the terms of our polynomial. Referring back to equations 5.12 and 5.13, we note that T_{12} and T_{21} both get multiplied by $c_1 c_2$. We can therefore define \tilde{T} by summing all T entries which

differ by only a permutation of indices. It can be shown that the number of terms now grows as $\binom{N+W-1}{N}$ which is slower than $(W)^N$ (Fig. 5.4(left)). We can therefore consider a larger number of bounces—in practice, we use up to $N = 4$ bounces, which we find sufficient for convergence.

However, as Fig. 5.4(right) shows, symmetry only reduces the complexity by a constant factor for growth with respect to W . While this factor is about 90 for the case of $N = 4$ shown, the $O((W)^4)$ behavior has not been reduced.

5.2.1 Low-Frequency BRDF approximations

To deal with the explosion in the number of bases for R , we make the important observation that equation 5.15 does not require us to use the same set of BRDFs R for every occurrence of $\mathbf{K}(R)$. We can define a hierarchy of BRDFs³ for each object, using less bases to represent the BRDF when considering light-surface interactions that are not directly visible to the viewer. For any bounce n on object m ,

$$\begin{aligned} \rho_n^m(\omega_i, \omega_o) &= \sum_{j=1}^{J_n} c_j^m b_j^m(\omega_i, \omega_o) \\ J &= J_N \geq \dots \geq J_n \geq \dots \geq J_1 \geq 1, \end{aligned} \tag{5.17}$$

where c_j^m and b_j^m correspond to the appropriate hierarchy (and are not the same for different ρ_n .) This creates lower-frequency BRDF approximations, motivated by the often-discussed low-frequency nature of indirect lighting and by experiments by (Nayar *et al.*, 2006) and theoretical work that shows that, at each bounce, the BRDFs act as a low-pass filter (Durand *et al.*, 2005).

³ This hierarchy is never directly exposed to the user. The user simply edits BRDFs in the usual way, by adjusting parameters or editing high-resolution 1D curves. The system automatically filters these to lower-frequency versions where needed, or computes diffuse equivalents as described in Section 5.2.4.

We now denote the distribution of BRDFs R with a superscript indicating the number of bases used per object,

$$B^N = \mathbf{K}(R^{J_N}) \mathbf{G} \mathbf{K}(R^{J_{N-1}}) \mathbf{G} \dots \mathbf{K}(R^{J_n}) \mathbf{G} \dots \mathbf{K}(R^{J_1}) \mathbf{G} E. \quad (5.18)$$

When equations 5.14 and 5.15 use the corresponding hierarchy of R 's, the subscripts are modified to run over a smaller domain such that $w_n \in [1, MJ_n]; MJ_n = W_n$. Specifically,

$$T_{(m_N j_N) \dots (m_n j_n) \dots (m_1 j_1)}^N(x, \omega_o) = \mathbf{K}(R_{m_N j_N}^{J_N}) \mathbf{G} \dots \mathbf{K}(R_{m_n j_n}^{J_n}) \mathbf{G} \dots \mathbf{K}(R_{m_1 j_1}^{J_1}) \mathbf{G} E, \quad (5.19)$$

where the subscripts $w_n = (m_n j_n)$ explicitly denote the object m_n and BRDF basis function j_n . This is the most general form of the multi-bounce tensor T^N .

A variety of decreasing-frequency BRDF series within our editing system are illustrated in Fig. 5.3. The scene is lit by an environment map and modeled after a figure in (Ben-Artzi *et al.*, 2006b). For clarity in the comparisons, we use only two bounces, and show series $(J_2, J_1) \equiv (128, 8); (16, 8); (128, 1)$. Figure 5.3f shows direct lighting only, as in (Ben-Artzi *et al.*, 2006b)—this omits important effects for the perception of materials and shininess, like the reflection of the teapot in the shiny tray (a black shadow results instead), or the reflection of the spout in the teapot. Figure 5.3a is ground truth, rendered offline (with two bounces) in PBRT (Pharr & Humphreys, 2004).

Figure 5.3 underscores that *further bounces can be represented with very low-frequency BRDFs*. The ground truth in (a) is essentially identical to the $(128, 8)$ BRDF series in (b) and (c), that uses only 8 BRDF bases for the second bounce. In

fact, our direct material perception primarily responds to the glossy tray reflecting nearby objects like the teapot. Therefore, even the purely diffuse approximation for further bounces from the eye $(128, 1)$ in (e) is usually adequate. In that case, the teapot appears diffuse in the reflection in (e), but this approximation is only for objects seen indirectly through reflections, and not easily noticeable.

Often, our choices are dictated by available computational resources. For a given complexity $J_2 J_1 = 128$, two possible options are $(16, 8)$ in (d) and $(128, 1)$ in (e). Both images are quite accurate, and can be edited within our system. They make different tradeoffs. The glossy reflection of the teapot in the tray is slightly more accurate in $(16, 8)$ because of a better BRDF approximation for the second bounce. However, the first bounce is represented at lower frequency than $(128, 1)$ —for example, direct reflections on the teapot are somewhat less sharp; this can become more noticeable for measured BRDFs and very shiny materials.

The observations from Fig. 5.3 indicate that using a diffuse equivalent for further bounces is a reasonable and efficient approximation, and we use it for some of our examples (Sec. 5.2.2). For more complex effects like caustics, we explore instead a series where the second bounce from the viewer uses a low-frequency approximation (Sec. 5.2.3).

5.2.2 Diffuse approximation for further bounces

In the limit, $J_{N-1} = 1$, and we approximate each object’s BRDF using a single basis function that is a diffuse lobe, scaled by the “equivalent albedo” of the BRDF. See Section 5.2.4 for a derivation of the equivalent albedo, d_m . We usually use four

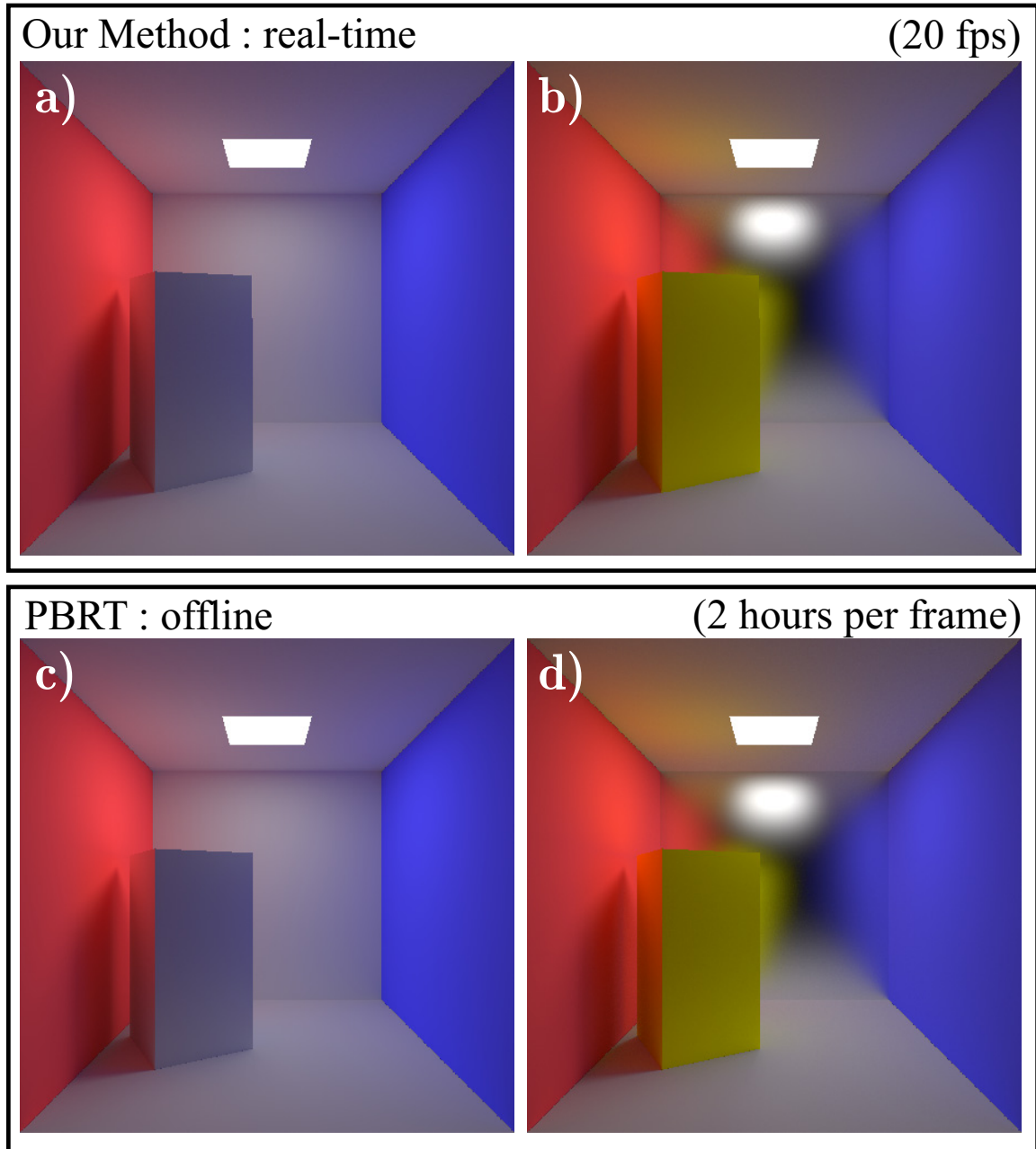


Fig. 5.5: BRDF Edits in Cornell Box: We edit the color of the small box, and also make the back wall shinier, all while rendering interactively with four bounces of global illumination. Notice the correct color bleeding in the scene, and correct glossy reflections of the scene in the back wall of (b). We compare our results to offline ground truth with PBRT in (c) and (d).

bounces $(J, 1, 1, 1)$ (with typically $J=64$).

$$B^N \approx \mathbf{K}(R^J) \mathbf{G} \mathbf{K}(R^1) \mathbf{G} \dots \mathbf{K}(R^1) \mathbf{G} E \quad (5.20)$$

$$T_{(j)(m_{N-1}) \dots (m_1)}^N = \mathbf{K}(R_j^J) \mathbf{G} \mathbf{K}(R_{m_{N-1}}^1) \mathbf{G} \dots \mathbf{K}(R_{m_1}^1) \mathbf{G} E,$$

where we have simplified the index pairs $(m_n j_n)$ in the general tensor of equation 5.19 as follows. We drop the m_N subscript in the first index pair, since only one object, $m_N(x)$, is visible through a pixel. We also drop the j_n subscripts for further bounces, since there is only one basis BRDF when using R^1 . Thus, we also simply use ‘ j ’ instead of ‘ j_N ’ for the bounce closest to the eye.

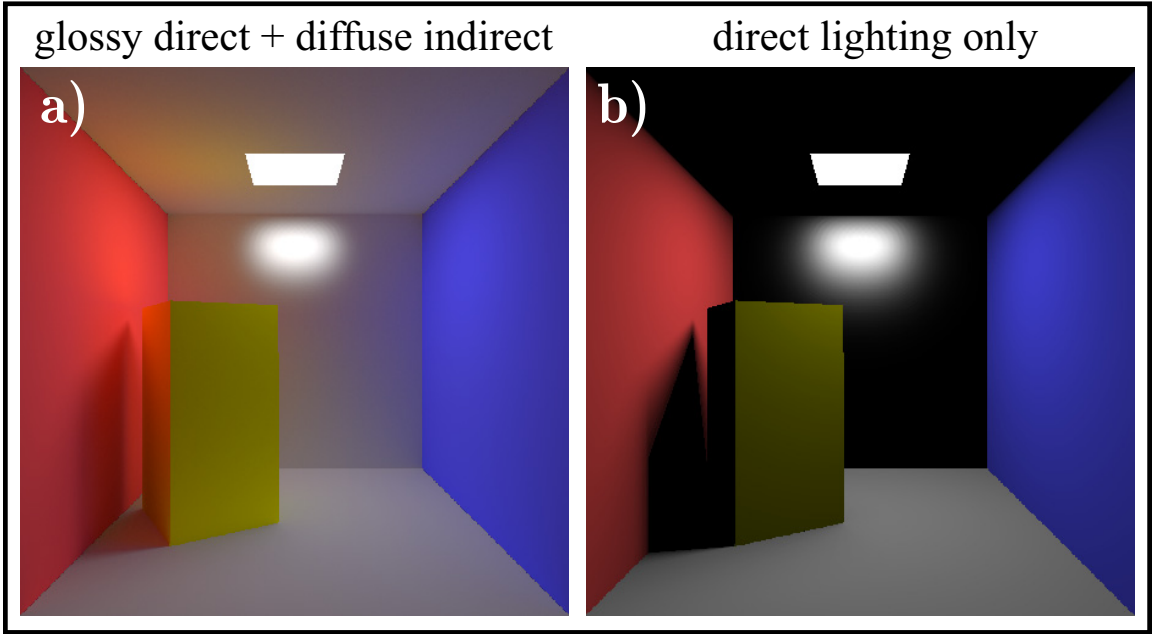


Fig. 5.6: Errors in Simpler Approximations: Compare (a) and (b) to Fig. 5.5b. (b) is the direct lighting approximation of Ben-Artzi et al., which fails to capture many important global illumination effects. (a) is a sum of (b) and diffuse indirect lighting, capturing some global effects but providing an inconsistent material perception for glossy objects like the back wall.

This approximation fully treats the first bounce from the viewer, including glossy reflections of the nearby scene. Bounces further from the viewer (and hence reflections

of objects not seen directly) are treated as diffuse. The complexity at each pixel reduces from $O((W)^N)$ to $O(J(M)^N)$. We will later see how this reduces further to $O(J)$ for rendering, since we edit only one object or material at a time.

Evaluation: Figures 5.5a and 5.5b are produced with our system. In (a), all surfaces are diffuse, while in (b) we edit the back wall to make it a glossy material, and change the color of the inner box to yellow. It is clear our method enables perception of material appearance, because objects correctly reflect other nearby objects (see the glossy interreflections of the room in the back wall in (b)), while also accurately preserving global effects like color bleeding onto the large inner box. We compare to ground truth using offline path tracing with PBRT in Figs. 5.5c and 5.5d, which confirms the accuracy of the approximation.

By contrast, Fig. 5.6b shows the direct lighting approximation of (Ben-Artzi *et al.*, 2006b) for the configuration in Fig. 5.5b. Not only is it missing a lot of energy, but it also lacks the reflections of the room in the back wall, which makes it difficult to assess the desired glossiness while editing.

Note that our method treats the first bounce from the eye with the full BRDF to get glossy reflections of nearby objects,

$$B \approx E + \mathbf{K}(R^J) \mathbf{G} E + \mathbf{K}(R^J) \mathbf{G} \sum_{N=2}^{\overline{N}} (\mathbf{K}(R^1) \mathbf{G})^{N-1} E.$$

Figure 5.6a compares to an alternative coarser approximation we originally tried using our framework. This simply adds a diffuse indirect solution to the full direct lighting result. It is essentially a series $(1, 1, 1, 1)$ for the indirect illumination, and therefore

the most efficient technique,

$$B \approx E + \mathbf{K}(R^J) \mathbf{G} E + \mathbf{K}(R^1) \mathbf{G} \sum_{N=2}^{\overline{N}} (\mathbf{K}(R^1) \mathbf{G})^{N-1} E,$$

where the main difference is that $\mathbf{K}(R^1)$ is used instead of $\mathbf{K}(R^J)$ for the leftmost operator of the multiple-bounce terms. Figure 5.6a is clearly better than direct lighting only—some global illumination is usually better than none.

However, a comparison with Fig. 5.5b shows that while further bounces can be approximated as diffuse, the first bounce from the eye does need the full high-frequency BRDF. Unlike our method, Fig. 5.6a gives an inconsistent material appearance of the back wall, that may be difficult to interpret while editing. While the direct reflection of the light source is glossy, the indirect reflections of the room appear diffuse.

5.2.3 Slower Decay of BRDF Series

Treating later bounces as diffuse works well in most scenes (see Figs. 6.1, 5.3e, 5.5 and 6.3). However, in some configurations like concave curved reflectors, higher frequency indirect effects like caustics are visually important (Durand *et al.*, 2005).

To handle such challenging situations, we need to reduce the BRDF frequencies more slowly, using more (8-16) basis functions for the second bounce from the eye. (Cases where even the third or higher bounces away from the eye need to be high-frequency are quite rare, though our general framework does not preclude taking them into account.) We compensate for the extra memory cost either by reducing the number of bases for the first bounce (Fig. 5.3d) or by using fewer bounces (Fig. 6.5 has only two bounces).

We have already seen an example of using 8 BRDF basis functions for the second bounce in Fig. 5.3(b-d), that gives a more accurate reflection of the teapot in the

shiny tray. In practice, our editing system also includes diffuse approximations for the third and fourth bounces to augment the series in Fig. 5.3 (see table 6.1). An even more challenging example is Fig. 6.5, that involves caustic effects. In this case, we use 16 BRDF basis functions for the second bounce, with a BRDF series of the form $(J, J/4) \equiv (64, 16)$.

5.2.4 Equivalent Albedo

We choose the equivalent albedo to match the average BRDF value, or more exactly, the output power for a uniform incident radiance field. This also corresponds formally to choosing the best perturbed \mathbf{K} operator, as in (Arvo *et al.*, 1994). In other words,

$$\begin{aligned} d &= \frac{1}{\pi} \int \int \rho(\omega_i, \omega_o) \cos \theta_i \cos \theta_o d\omega_i d\omega_o \\ &= \frac{1}{\pi} \sum_j c_j \int \int \rho_q(\omega_i, \omega_o) b_j(\gamma(\omega_i, \omega_o)) \cos \theta_i \cos \theta_o d\omega_i, d\omega_o. \end{aligned} \quad (5.21)$$

The term in the integral now depends only on known quantities—the quotient BRDFs and the basis functions, and can therefore be evaluated by dense Monte Carlo sampling (this needs to be done only once for a given parameterization, not even for each scene). Call this e_j . Finally, at run-time, we simply need to compute

$$d = \frac{1}{\pi} \sum_j c_j e_j, \quad (5.22)$$

with the predetermined e_j and the dynamically chosen c_j .

Note that the quotient BRDF must be included in equation 5.22 for energy conservation.

This section describes how we approximate a BRDF with an energy-conserving equivalent diffuse BRDF. (Arvo *et al.*, 1994) introduced a norm for the reflectance

operator, \mathbf{K} . In that work, they show that minimizing the norm of the difference between the true reflectance operator \mathbf{K} and a perturbed operator $\tilde{\mathbf{K}}$, will minimize the error in the solution. In our case, $\mathbf{K} = \sum c_w \mathbf{K}_w$ and $\tilde{\mathbf{K}} = \mathbf{K}^{\mathbf{D}}$. Using the L^1 norm presented in (Arvo *et al.*, 1994) (which measures the total power in the exit radiance field produced by applying the BRDF to the incoming light), we want to minimize:

$$\int_S \iint_{\Omega^2} \sum_w c_w \mathbf{K}_w - d \mathbf{K}^{\mathbf{D}} d\omega_i d\omega_o dx \quad (5.23)$$

This is equivalent to stating that the total power attenuation for a uniform incident radiance field should be the same for the diffuse case, as for the full \mathbf{K} . Since we perform this operation once for each object, and the BRDF across an object's surface doesn't vary, we neglect the variation with respect to x . By pulling the sum outside the double integral, and then distributing the double integral to the two terms, we want to minimize:

$$\sum_j \left[c_j \iint \mathbf{K}_j - d \iint \mathbf{K}^{\mathbf{D}} \right] \quad (5.24)$$

While it is simplest to think of \mathbf{K}_j as a box basis function of the BRDF, including the quotient BRDF into it is important for energy conservation. For example, a Blinn BRDF does not approach a Lambertian BRDF when the exponent $\rightarrow 0$ because the \mathcal{GAF} (geometric attenuation factor) is always present, and reduces the energy. Our analysis shows that a Blinn with exponent 0 is equivalent to a diffuse albedo of 0.56. If we set each term in equation 5.24 to be 0, we know that $c_j \iint \mathbf{K}_j$ must equal $d \iint \mathbf{K}^{\mathbf{D}}$. Dividing by $\iint \mathbf{K}^{\mathbf{D}}$, we can define e_j and get an expression for d

$$e_j = \frac{\iint \mathbf{K}_j}{\iint \mathbf{K}^{\mathbf{D}}} \quad ; \quad c_j e_j = d \quad (5.25)$$

At precomputation time, we compute the e_j for the BRDF of each object by gener-

ating several thousand light and view direction using the path-tracer's Monte Carlo method. At run-time, we compute d , given the user-defined c_j . This gives us the albedo of a diffuse object that would reflect the same percentage of incoming radiance as the user-defined BRDF. This technique can be generalized to approximations with more than a single basis function.

6. INTERACTIVE RENDERING WITH GLOBAL ILLUMINATION

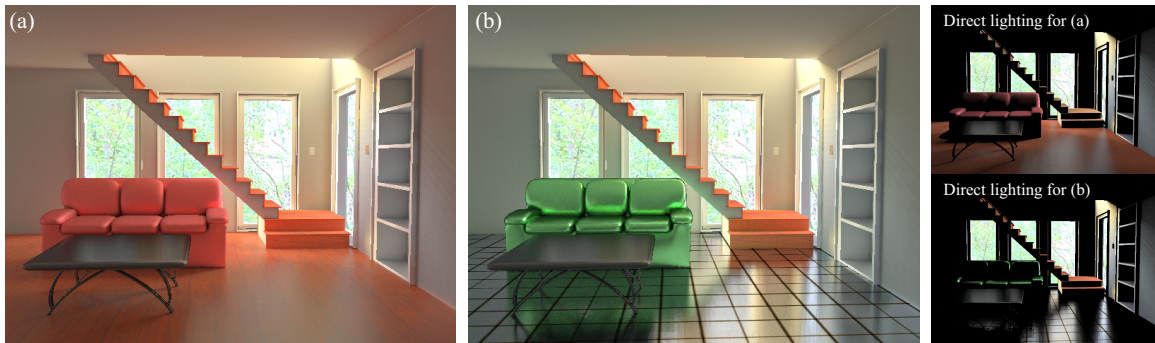


Fig. 6.1: Interior Design Session: We simulate an interior design session in which we edit the BRDFs of the couch and floor. The couch’s red fabric in (a) is loaded from measured data, and edited to a more specular green material in (b). The floor is re-textured and made very glossy in (b). The reflections of objects in the near-mirror floor, and color bleeding from the couch to the staircase can be seen here as well as in closeups in Fig. 6.4. The scene is lit only from the large windows, using the top half of an exterior environment map (campus). We can see in the comparisons to direct lighting (far right) that most of the room’s lighting is a result of indirect illumination.

While relighting systems have long provided feedback with global illumination in complex scenes (Dorsey *et al.*, 1995), BRDF editing has been limited to simplified settings such as point lights.

In Chapter 4, we introduced the ability to edit BRDFs under natural illumination, albeit only with direct lighting. This is a significant limitation in some scenarios since indirect illumination and glossy reflection are essential to the realism of today’s renderers, and are often critical to correctly perceive and choose material properties.

For precomputation (Sec. 6.1.1), we show how Monte Carlo path tracing can be extended to precompute the multi-bounce tensors needed. For rendering (Sec. 6.1.2), since only one object’s BRDF is edited at a time, we show that the tensor can be reduced to a few vector dot products at each pixel, whose coefficients can be computed in a very fast runtime preprocess. Our results show a variety of BRDF edits, with interactive feedback rendered with global illumination.

6.1 *Implementation*

We now describe our implementation, starting with our precomputation method (Sec. 6.1.1), followed by the rendering algorithm (Sec. 6.1.2), and some practical extensions (Sec. 6.1.3).

6.1.1 *Monte Carlo Precomputation*

We need to precompute the tensors defined by equation 5.19 at each pixel. The important special case for diffuse approximation in further bounces is given by equation 5.20. Recall that the \mathbf{K} operators involve integrals over the hemisphere, which means that each $T^N(x)$ requires nested (high-dimensional) integrals over ray paths. This is similar to traditional global illumination.

We adapt Monte Carlo path tracing (Kajiya, 1986) for precomputation because of its flexibility, ease of implementation and negligible memory overhead.

Each value in the different tensors can be seen as a separate integral over the space of paths. However, it is easier and more similar to traditional path tracing to sample path space for all integrals at the same time. We generate random paths, and for each path update the appropriate tensor integrals. We must modify three basic aspects of the path tracer. First, we cannot generate rays by sampling the BRDF

(since it is unknown). Second, we must add each path’s contribution to the correct tensor element, as opposed to simply contributing to the final pixel radiance. Third, we must compute the contribution of each path using basis BRDFs.

Consider a given path \vec{X} from a point on a light source (ℓ) to the eye (e), that passes through points x_N, x_{N-1}, \dots, x_1 on objects m_N, m_{N-1}, \dots, m_1 , as illustrated in Fig. 6.2.

Sampling path space: According to Monte Carlo theory, any random sampling can be used to generate new directions when building up the path. We follow standard path tracing and generate rays recursively from the eye. We sample the light source at each bounce to generate all path lengths.

Path tracers usually importance sample the BRDF to select a new direction at each intersection. Unfortunately, we have no knowledge of the final BRDF. We cannot sample according to the basis BRDFs either because they will be combined with arbitrary weights at runtime. The simplest approach would be to sample the cosine-weighted hemisphere uniformly, but this would yield high variance when sharp specular lobes are used. Instead, we take advantage of the general form of typical BRDFs and sample according to a mixture of 70% diffuse, and 30% high-gloss (Blinn exponent of 200). This places more importance on specular directions and enables low-noise results for the full range of glossy to near-mirror BRDFs in a practical editing session.

Tensor update: For each random path, we need to accumulate a contribution to the appropriate tensor element. In effect, we are computing coefficients in a symbolic polynomial representation of the basis BRDFs (in the spirit of symbolic rendering by (Séquin & Smyrl, 1989)). In our case, we have chosen bases that do not overlap, and therefore a given path requires updating exactly one tensor element. The j index

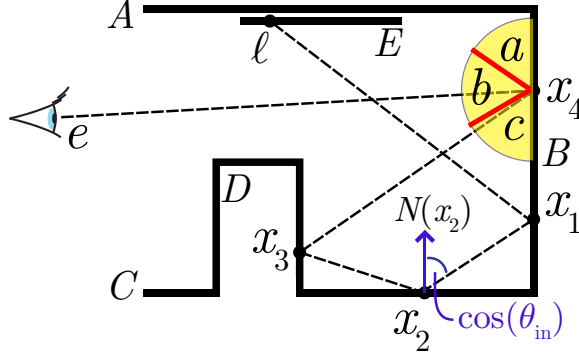


Fig. 6.2: **Light Paths:** Consider a path \vec{X} from the light (ℓ) to the eye (e). The light hits objects B , C , D , B before reaching the eye. At the final bounce (x_4), the view direction defines how the basis functions of B 's BRDF divide the incoming light directions. Of the three bases (a , b , and c), the configuration of the last bounce places it in c . Therefore, this path contributes to $T_{Bc,D,C,B}^4$.

in equation 5.20 is determined by the basis function that contains the configuration of incoming and outgoing directions at the last intersection point (in Fig. 6.2 this is x_4). The outgoing direction for the bounce to the eye ($x_4 - e$) partitions the space of incoming directions into bands corresponding to our different box-basis functions (a , b , and c in Fig. 6.2). In the example, band c contains the incoming path direction, which determines j . More generally, we would choose the j for which $b_j(\omega_i, \omega_o)$ is non-zero. In the case of the diffuse approximation for further bounces, we only care about the objects containing the further bounces (the indices $m_{N-1} \dots m_1$ in equation 5.20). In Fig. 6.2, these are DCB . The more slowly decaying BRDF series with multiple specular bounces would use a similar band selection for the second bounce from the eye, as for the first.

Tensor values involve a standard Monte Carlo sum,

$$T_{w_N w_{N-1} \dots w_1}^N(x) = \frac{1}{Q} \sum_i \frac{f(\vec{X}_i)}{p(\vec{X}_i)}, \quad (6.1)$$

where Q is the number of paths through pixel x , and the sum runs only over paths

\vec{X}_i that correspond to the specific subscripts (bands and objects) in T^N . $f(\vec{X}_i)$ is the contribution of \vec{X}_i , and $p(\vec{X}_i)$ is the probability of generating it.

Path Contribution: In standard path tracing, the path contribution $f(\vec{X}_i)$ is the direct visible lighting at x_1 , multiplied by the product of BRDFs (corresponding to \mathbf{K}) and cosine terms at each intersection (the visibility in \mathbf{G} is already considered when creating valid paths). In our case, we must instead multiply by the appropriate *basis* BRDFs.

For the first bounce from the eye, we use

$$\tilde{b}_j(e, x_N, x_{N-1}) = b_j^{m_N(x_N)}(\omega_i, \omega_o) \cos \theta_i, \quad (6.2)$$

where $\omega_i(x_N, x_{N-1})$ and $\theta_i(x_N, x_{N-1})$ depend on the direction of the incident ray, and $\omega_o(e, x_N)$ on the outgoing view direction. For the slowly decaying series, a very similar form can be used for the second bounce from the eye, simply considering a lower-frequency $\tilde{b}_j(x_N, x_{N-1}, x_{N-2})$. For the other bounces, we use the single diffuse basis:

$$D(x_n, x_{n-1}) = \frac{1}{\pi} \cos(\theta_i(x_n, x_{n-1})). \quad (6.3)$$

Finally, $f(\vec{X})$ is a product of the terms at each bounce. For the diffuse approximation for further bounces, this is

$$f(\vec{X}) = \tilde{b}_j^{m_N(x_N)}(\omega_i, \omega_o) D(x_{N-1}, x_{N-2}) \dots D(x_1, \ell) E(\ell). \quad (6.4)$$

Optimizations: Our precomputation is essentially the same complexity as rendering a single image with MCPT. Moreover, many standard path tracing optimizations can still be applied. For example, we have adapted irradiance caching (Ward *et al.*,

1988)—instead of generating paths that terminate at the light source, we find the direct lighting at the last surface point x_1 . We cache the irradiance in a preprocess that samples visibility on a grid within each triangle.

6.1.2 Rendering in Real Time

We now focus on the runtime rendering computation for each pixel. For compactness of notation, this subsection will deal primarily with the diffuse approximation for further bounces, but more slowly decaying series use very similar methods, and are discussed briefly at the end.

To simplify notation, we denote the tensor as $T_{jz}^N(x)$, where the single “super-index” z is a short-hand for writing out $m_{N-1} \dots m_1$ in equation 5.20 (viewed as an index, $z \in [1, Z = (M)^{N-1}]$). Similarly, we also denote the product of diffuse equivalents d_m of each object by \bar{d}_z ,

$$\begin{aligned} z &\equiv \{m_{N-1}, \dots, m_n, \dots, m_1\} \\ \bar{d}_z &\equiv d_{m_{N-1}} d_{m_{N-2}} \dots d_{m_n} \dots d_{m_2} d_{m_1}. \end{aligned} \tag{6.5}$$

Note that z represents a list of indices, while \bar{d}_z is a single number, corresponding to the product of the albedos d_m .

Finally, we can adapt equation 5.14 for rendering,

$$B^N(x) = \sum_{j=1}^J \sum_{z=1}^Z T_{jz}^N(x) c_j \bar{d}_z. \tag{6.6}$$

During the edit, the user specifies the BRDF coefficients c_j (either directly by editing a curve, or implicitly by editing a parametric model). The diffuse equivalents d (and hence \bar{d}_z) are then computed as described in Section 5.2.4. Finding the c_j and

\bar{d}_z occurs once per frame for the whole image. Using the precomputed $T_{jz}^N(x)$, the double summation in equation 6.6 must now be evaluated at each pixel.

Object Freezing: Equation 6.6 requires $O(JZ)$ operations per pixel. On modern hardware, this is fast, but still not real-time (requiring a couple of seconds per update). To reduce complexity, we observe that a user edits the BRDF of only one object at a time. We use a run-time precomputation that performs the bulk of the calculations in Equation 6.6 by temporarily “freezing” the BRDFs of the remaining objects.

Recall that \bar{d}_z represents a multi-variable polynomial in the d ’s of the objects in the scene. For example, if we have $z = \{1, 3, 2, 1, 5, 3\}$, $\bar{d}_z = (d_1)^2 d_2 (d_3)^2 d_5$. However, if all but one of the d ’s are fixed, this becomes just a single-variable polynomial in the unfrozen d of the object being edited. For example, if all but object 1 are “frozen”, we can define a constant $A = d_2 (d_3)^2 d_5$, so that \bar{d}_z becomes a simple quadratic polynomial, $\bar{d}_z = A \cdot (d_1)^2$ in only the edited variable d_1 .

To implement this scheme more formally, we need a helper function $n(z, i)$ that tells us how many times a given edited object i appears in z . In the above example, $n(z, i) = 2$ (for $i = 1$) that tells us \bar{d}_z is a quadratic polynomial in d_i alone. We can also define the constant A more formally as $A = \bar{d}_z / (d_i)^n$. Finally, we compute at run-time a new tensor of coefficients at each pixel, where each row represents a single-variable polynomial in d_i ,

$$F_{jn}^i(x) = \sum_N \sum_z \begin{cases} n(z, i) \neq n : & 0 \\ n(z, i) = n : & T_{jz}^N(x) \frac{\bar{d}_z}{(d_i)^n} \end{cases} \quad (6.7)$$

Our real-time rendering step is now a direct evaluation of

$$B(x) = \sum_{n=0}^{\overline{N}-1} (d_i)^n \sum_{j=1}^J c_j F_{jn}^i(x). \quad (6.8)$$

For each power $(d_i)^n$, we simply evaluate a dot-product $c_j F_j$, essentially as in a standard linear PRT formulation.

The computation in equation 6.7 requires $O(JZ)$ operations per pixel, comparable to simply evaluating equation 6.6 directly once. This requires a short (usually 5-10 seconds) mode switch each time the user begins editing a different object. The real-time rendering in equation 6.8 is now $O(J)$ (the number of bounces \overline{N} is a small constant, usually four.)

Two further optimizations are possible. In a practical editing session, the coefficients c_j change slowly from frame to frame, especially if we transform into a wavelet representation. This temporal coherence can be directly exploited using the incremental wavelet rendering method described in (Ben-Artzi *et al.*, 2006b). Finally, if we are rendering a pixel of an object that is not being edited, the c_j do not change at all. (Note however, that the object's appearance will still be affected because of global illumination.) This makes it possible to further precompute $V_n^i = \sum_j F_{jn}^i c_j$, reducing the cost to evaluating a simple polynomial in d_i .

Slower Decaying Series: We briefly describe the generalization to more slowly decaying series, as in Sec. 5.2.3. The general rendering operation of Equation 6.6 is now best described as a triple-summation, since we are dealing with three distinct representations of R : R^{J_N} , $R^{J_{N-1}}$, and R^1 .

$$B^N(x) = \sum_{j=1}^{J_N} \sum_{w=1}^{MJ_{N-1}} \sum_{z=1}^Z T_{jwz}^N(x) c_j \hat{c}_w \bar{d}_z, \quad (6.9)$$

with \hat{c}_w denoting the lower-frequency BRDF coefficients for the second bounce (J_{N-1} bases on each of the M objects).

Object freezing is a bit more difficult, theoretically requiring the creation of a three-dimensional F_{jkn}^i , where $j \in [1, J_N]$, $k \in [1, J_{N-1}]$, $n \in [0, \overline{N}-2]$. In practice, we think of the j as one dimension, and $k' \equiv kn$ as the other.

6.1.3 Extensions

We briefly describe two important practical extensions.

Objects with Fixed BRDFs: Large scenes can contain many small objects that cause an exponential increase in memory requirements. Such scenes usually do not require editing the BRDFs of *all* objects. When designing the materials in a room, one typically does not want to change the BRDFs of small “placeholder” objects like books or toys. We extend our algorithm by implementing the ability to fix the BRDFs of certain objects at precomputation. Note that their shading is still updated, based on global illumination from other surfaces. This should also not be confused with run-time object freezing above, which occurs temporarily during an editing session.

In precomputation, instead of using diffuse equivalents D or BRDF bases \tilde{b} , we must use the full known BRDF $\rho^m(\omega_i, \omega_o) \cos \theta_i$ for reflections from fixed object m . Rendering is unchanged for editable objects, since there are no new BRDF bases. For the fixed object, we still use $T_z^N(x)$ and multiply by the diffuse albedos of editable objects. However, the BRDF bases (and index j) need not be considered. In Fig. 6.1, the table and its legs have fixed BRDFs.

Spatial Weight Maps: So far, we have focused on BRDF effects. Spatial variation can be handled with textures to modulate the BRDF over an object’s surface. If we do

not seek to edit them, the textures can be directly incorporated into the BRDF basis functions (as the multiplicative $H^m(x)$ terms in equation 5.5). Finally, while we have discussed a single BRDF per object for clarity, our framework and implementation can handle multiple co-located BRDFs for each object. If we also seek to modify the spatial blending of BRDFs, as we do for the floor in Fig. 6.1, we can simply modulate the directly viewed surfaces in image-space by the multiplicative spatial blending weights (weight maps) or texture. For global illumination, we are concerned only with the low-frequency behavior of the weight maps or textures, and we modulate the diffuse equivalent albedos by the average value of the weight map for each BRDF layer.

6.2 Global Illumination Results

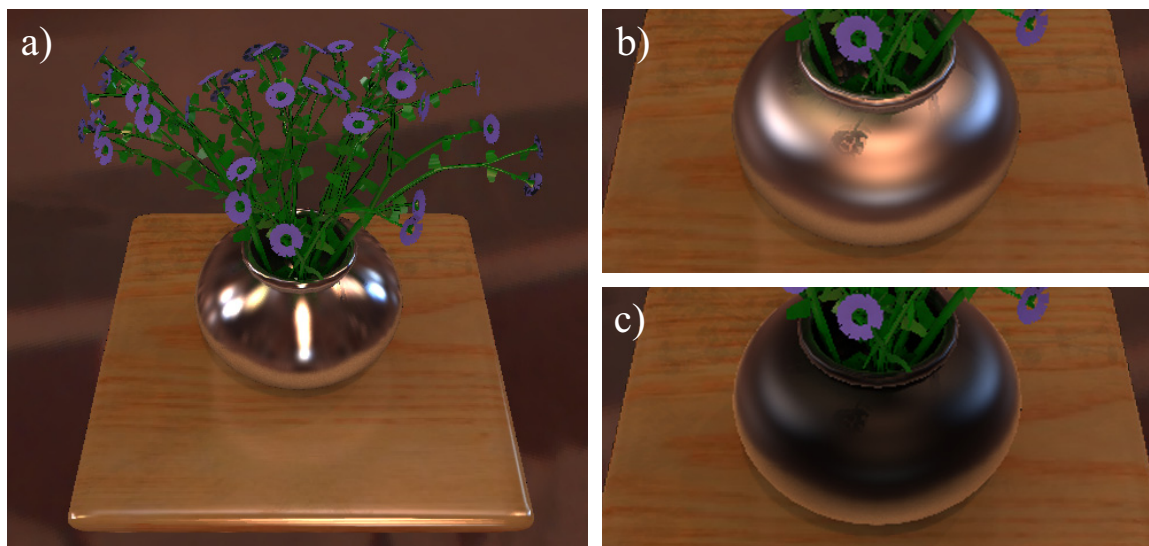


Fig. 6.3: **Vase with Flowers:** a) original, b) closeup after anisotropic edit of vase in (a), c) closeup after fresnel effect strongly increased for (b).

Section 8.7.1 briefly discusses the types of edits performed on the scenes in Figs. 6.1, 5.3, 5.5, 6.3 and 6.5, and the global illumination effects involved. Then, Sec. 8.8.3 gives performance results for precomputation time and memory usage, while Sec. 6.2.3 discusses rendering frame rates.

6.2.1 Editing and Visual effects

Cornell Box: Figure 5.5 shows the Cornell box, where we edit the parameters of a Blinn-Phong and diffuse reflectance model. In going from (a) to (b), we make the back wall glossy with correct interreflections of the nearby scene, and change the color of the inner box, demonstrating accurate color bleeding. Note that even the simple color adjustment was not possible interactively with global illumination in previous methods.*

* Since the Cornell Box was designed for verifiable comparison to PBRT, it was precomputed with the floor and back wall using Blinn-Phong with 64 bands, while the other objects use 2 bands to

Teatray: Figure 5.3 shows a teatray scene. The teapot has a Cook-Torrance BRDF with specular and diffuse components, and the handles and tray a Blinn-Phong model. While only a single set of BRDFs for the objects is shown in Fig. 5.3 for brevity, we can freely edit the teapot, tray and handles in real-time, using any of the BRDF series shown in the figure (extended to three or four bounces).

Vase: Figure 6.3 shows a variety of flexible BRDFs possible within our system. The flowers are diffuse, while the stems are a diffuse+specular BRDF to enable a glossy coating. The table has diffuse and specular BRDFs, with the diffuse shown textured in Fig. 6.3. The vase uses a diffuse and a specular layer. The specular BRDF is an Ashikhmin-Shirley BRDF with fixed exponent of 225, but adjustable ratio to adjust the direction and amount of anisotropy (see Section 3.3.2). A second specular layer allows for edits to the Fresnel term. Section 3.3.1 describes how to enable Fresnel control for any BRDF via a second additive layer.[†]

Room: Figure 6.1 shows one potential application of our system to design interiors. In this case, indirect light is critical, since a large part of the scene, like the back wall and much of the couch, would be black with direct lighting only. Most of the indirect illumination is low-frequency, so we use our diffuse approximation for further bounces. We only use three bounces, with a BRDF series (64, 1, 1), since we found that the fourth bounce contributed relatively little to the qualitative appearance of the scene.

This is a complex scene with 8 objects (6 editable), environment lighting from the windows, and a variety of materials including measured reflectance and textures, which can all be edited. Our system renders interactive feedback with global illu-

represent a pure diffuse BRDF with editable albedo.

[†] The glossy layers of the stems and table use 64 bands. The Ashikhmin-Shirley layers of the vase use 256 bands. All diffuse layers use 2 bands.



Fig. 6.4: Room Closeup: Closeups for Fig. 6.1. Note the color bleeding of the couch onto the stairs. Note also the glossy reflection of the couch in the shiny floor on the right, which is a third-bounce effect (since the bottom of the couch is lit only by indirect lighting).

mination, enabling the user to correctly perceive and edit materials to design the interior of the room.

The effects of global illumination are clearly seen in the closeup views of Fig. 6.4, where the couch color bleeds onto the stairs. Notice also the glossy reflection of the green couch in the highly shiny floor on the right. The lower portion of the couch is lit only by indirect illumination, so this is actually a glossy reflection of indirect light, and needs at least 3 bounces to simulate properly.[‡]

Ring: Figure 6.5 shows how our system can be used even to choose materials to create the desired complex global illumination effects like caustics. In this case, we use a slower series decay (64, 16) that includes two specular bounces, to obtain accurate

[‡] Different objects in the room were computed with different BRDF parameterizations and resolutions. The couch and floor have 64 bands for specular BRDFs. The floor also has a diffuse layer with 2 bands. The stairs and sills use 16 bands for glossy BRDFs. The walls and ceiling have only a diffuse BRDF with 2 bands, allowing color and intensity edits, but not glossy.

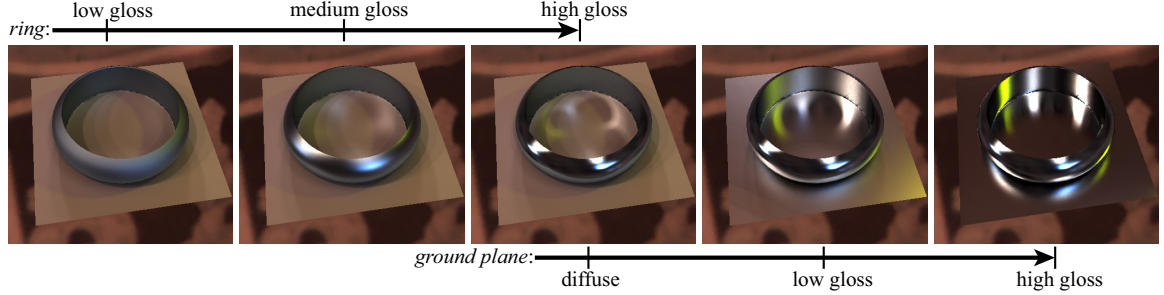


Fig. 6.5: Two-Bounce Glossy Edits: To obtain accurate indirect reflections, we enable second bounce glossy reflections. We show results with an implementation that uses $J = 64$ for the first bounce from the eye and $J = 16$ for the second. With this approximation, we can not only get caustics on the floor from the ring, but also see the reflection of the floor in the ring accurately, as shown in the rightmost image.

indirect specular reflections. Our system interactively updates both the caustics on the floor from the ring, and the reflection of the floor in the ring accurately, as the BRDFs are edited. Note that the sharpness of the caustics and indirect reflections are maintained even though the second bounce BRDF is still quite low frequency. Without our system, it would be quite difficult to interactively select the glossiness of the plane and ring, to explore possible appearances of the caustics.

6.2.2 Precomputation Times and Memory Usage

Table 6.1 describes precomputation times and memory usage. The rows show the scenes (including multiple BRDF series for the teatray). The image resolutions were chosen primarily to fit the resolution of the accompanying video—the teatrays were computed at lower resolution for faster testing and comparison to PBRT renders (which took hours to generate low-noise still images at these resolutions).

Precomputation Time: The precomputation time (on an Intel Xeon 3.2GHz 64 bit machine) ranges from one to several hours, depending linearly on the number of path tracing samples used, and also varying with the number of point lights to

sample the environment. Interestingly, these wall clock times are about as fast (and sometimes faster than) for standard PBRT to render a single (uneditable) image of the scene—this is because the complexity of our precomputation is essentially the same as rendering a single image with path tracing. Thus, our precomputation times, while large, are comparable to those for high quality global illumination image synthesis methods.

scene specifications				precomputation			storage per path-length (MB)					rendering	
<i>name</i>	<i>resolution</i>	<i>M</i>	<i>BRDF series</i>	<i>samples</i>	<i>lights</i>	<i>time(h:m)</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>total</i>	<i>freeze</i>	<i>fps</i>
Cornell box	450×450	6	64,1,1,1*	12K	81/81	5:58	17	104	365	975	1461	10s	22
Vase	512×512	4	64,1,1,1 [†]	4K	10K/400	5:29	65	262	637	—	964	8s	19
Room	640×480	6	64,1,1 [‡]	8K	14K/800	9:25	27	165	563	—	755	7s	20
Teatrays	320×240	3	128,1,1,1	10K	10K/2K	1:07	13	37	75	125	250	3s	25
			16,8,1,1			1:42	2	37	113	333	485	3s	16
			128,8,1			2:10	13	301	853	—	896	15s	5
Ring	400×400	2	64,16	4K	4K/400	6:15	20	607	—	—	627	16s	6

Tab. 6.1: Timings and Storage for Global Illumination: Precomputation time, storage of precomputed data structures, and rendering time for each of our scenes. The scene specification includes image resolution, number of objects (M), and the BRDF series. Precomputation lists the number of samples per pixel for path tracing, and the number of point lights used to sample the environment for direct/indirect lighting (we usually use fewer samples for indirect illumination). Storage is shown separately (in MB) for each path-length, or number of bounces. Rendering time indicates the time for “object freezing” when selecting a single object to edit, and for real-time rendering.

Memory Usage: The memory usage grows for each bounce, since there are more polynomial terms in T^N (as shown in Fig. 5.4 (left)). The growth is relatively slow, being a factor of about 3 for higher bounces. Nevertheless, the highest (usually fourth) bounce requires more than half the total memory. This is an interesting direction for future work, since it is usually the darkest and most amenable to compression. However, in our experiments with direct quantization and wavelet compression, we were not easily able to find a scheme that was free of image artifacts. The problem is different from compression for relighting, since we visualize the BRDF and image directly, making artifacts more apparent. We instead simply drop zero values, and use RGBE compression. At the end, our total precomputed data structures are in

the range of several hundred MB. While this is large, it is comparable, for instance, to all-frequency relighting approaches on similar resolution images.

Note that the storage sizes shown in Table 6.1 are larger than the theoretical runtime memory requirements. Due to our object freezing step, only the smaller F_{jn}^i in equation 6.7 needs to be in main memory. We can avoid preloading all of the data at the cost of higher object-switch times.

In comparing the different BRDF series for the teatray, (128, 8, 1) requires the most memory, because of the extra factor of 8 for second bounce BRDF bases (as opposed to (16, 8, 1, 1) and (128, 1, 1, 1)). To keep memory consumption reasonable, we limit to 3 bounces rather than 4 as with the other series. These numbers are comparable to the ring, that also uses 16 bases in the second bounce. (We similarly limit the number of bounces in the ring to two (64, 16)).

6.2.3 *Rendering Speed*

For simplicity, we pursue a purely software implementation, although the simple dot-product form of equation 6.8 indicates that GPU implementations, similar to those recently developed for relighting should also be applicable.

As can be seen in the video and the last column of table 6.1, our software method achieves frame rates of 15-25fps on most scenes. The time for “object-freezing” when we switch from one object to another for editing is about 5-10 seconds, and isn’t disruptive to typical editing sessions, as seen in the video. The rendering and mode switch times are somewhat larger when there are more BRDF bases for the second bounce (one teatray example and ring), on account of the additional complexity. However, they are still interactive, and our video demonstrates interactive editing of the ring scene. In summary, our results and video for the first time show practical real-time BRDF editing, as interactive feedback is rendered with global illumination.

Part III

NEW CONCEPTS APPLICABLE TO BRDF EDITING AND MORE

The general theoretical framework presented in Part II can be improved with several orthogonal developments presented in this part. These concepts merit an independent discussion because they are easily beneficial to BRDF editing, as well as other applications in computer graphics. Chapter 7 will introduce an improvement of the standard dot-product performed at each pixel during rendering in a precomputation-based context. The improvement relies on the temporal coherence of inputs from frame to frame, without being specific to the meaning encoded in the input vectors. Chapter 8 presents a fast method for computing visibility of a lighting environment. This calculation is a necessary step to shading surface illuminate by environment maps. It is also required by the precomputation steps discussed in Part II. Instead of casting shadow rays to 100's of sample points in an environment map, the visibility function can be constructed with an accurate prediction of about 90% of the shadow rays. This gain is achieved by relying on the spatial coherence of visibility in typical scenes. Chapter 9 presents a novel function editing framework. This method of editing 1D functions is utilized as the user interface for editing 1D BRDF factors as described in Part II. The operations described in Chapter 9 can be equally applied to any graphics application that requires editing 1D functions, such as the curves tool in image-editing software, and time-dependent parameters in animations.

7. TEMPORAL COHERENCE FOR REAL-TIME PRECOMPUTED RENDERING

In Part II, Sections 4.1 and 6.1.2 phrase the real-time rendering operation as a dot-product at each pixel. One of the vectors of the dot-product is constant throughout the editing session, but unique to each pixel. The other vector changes from frame-to-frame, but is uniform across all pixels of an object. The methods described in this chapter are equally applicable to any operation which has this same nature for the two vectors of a dot-product. For a concrete application, we will describe it in terms of equation 4.7 in Section 4.1.

Recall that Chapter 4 described how to obtain the transport vector \mathbf{T} , and vector of function coefficients \mathbf{c} , required by equation 4.7. The number of elements in the dot-product $\mathbf{T} \cdot \mathbf{c}$, can be quite large, and when performed at each pixel, modern CPUs may not be able to render this in real-time. A common solution is to transform both vectors into a wavelet basis, and then use a non-linear wavelet approximation that makes these vectors sparse,

$$R_{perfect}^i = \mathbf{T} \cdot \mathbf{c}_{perfect}^i \quad (7.1)$$

$$R_{effective}^i = \mathbf{T} \cdot \mathbf{c}_{effective}^i \quad (7.2)$$

where $\mathbf{c}_{effective}^i$ is the representation of $\mathbf{c}_{perfect}^i$ after approximation, and $R_{effective}^i$ is the effective (approximate) image displayed. This method has two drawbacks for

us. First, we cannot compress as aggressively as relighting methods, which never directly visualize the lighting ¹. We need to keep many more terms in $\mathbf{c}_{effective}^i$, since both lighting and BRDF retain all frequencies. Second, the standard non-linear wavelet approximation, by definition ignores small coefficients, often giving the user no feedback when small adjustments are made to the BRDF.

We note that \mathbf{c} tends to change by small amounts between frames, since the user is continuously editing the 1D curve that defines it. We take advantage of this by rendering the current frame i , using the change \mathbf{c}_{diff}^i relative to the previous frame $i - 1$,

$$\mathbf{c}_{diff}^i = \mathbf{c}_{perfect}^i - \mathbf{c}_{effective}^{i-1} \quad (7.3)$$

The \mathbf{c} 's of consecutive frames tend to be more similar when they are coefficients of a wavelet basis. In interactive editing, one is usually making local modifications to the curve, so \mathbf{c}_{diff}^i has only a few nonzero wavelet coefficients, even when $\mathbf{c}_{perfect}^i$ would need many terms for accurate approximation. We must also consider occasional cases where \mathbf{c}_{diff}^i has more nonzero wavelet terms than we can afford to use in a single frame. This happens when the user makes drastic changes that affect the entire curve (such as a uniform scaling). To handle these situations within our framework, we use a standard non-linear wavelet approximation, $\tilde{\mathbf{c}}_{diff}^i$, of \mathbf{c}_{diff}^i . For most frames, $\tilde{\mathbf{c}}_{diff}^i$ will be equal to \mathbf{c}_{diff}^i , but allowing for approximation makes our system robust when large changes occur (or if the wavelet budget is very small).

¹ Relighting implicitly assumes the BRDF is lower frequency than the lighting (no near-mirror), thus being more forgiving of lighting approximations.

Our final equations for incremental rendering then become:

$$R_{effective}^i = R_{effective}^{i-1} + \mathbf{T} \cdot \tilde{\mathbf{c}}_{diff}^i \quad (7.4)$$

$$\mathbf{c}_{effective}^i = \mathbf{c}_{effective}^{i-1} + \tilde{\mathbf{c}}_{diff}^i, \quad (7.5)$$





















where we make explicit that the final radiance, $R_{effective}^i$ may not be exact. Equation 7.5 shows how we keep track of the $\mathbf{c}_{effective}^i$, so that \mathbf{c}_{diff}^{i+1} can be calculated in the next frame, per equation 7.3. Note that the expensive operation here is the (per-pixel) dot-product ($\mathbf{T} \cdot \tilde{\mathbf{c}}_{diff}$) just as in standard (non-incremental) PRT rendering.

Even though we never explicitly calculate the dot-product in equation 7.2, our incremental updates in equation 7.4 *effectively* give the same result. The main difference is that our $\mathbf{c}_{effective}$ are not approximated with a predetermined budget, but rather can get arbitrarily close (and often equal) to $\mathbf{c}_{perfect}$. Table 7.1 uses equations 7.3-7.5 to compute successive frames of a didactic editing session.

Incremental wavelet rendering addresses the issues mentioned at the start of this chapter. Since work is focused only in the local region where the user is changing the BRDF, our system almost always renders reference-quality images. In rare cases, when the region of change is large, and cannot be exactly captured in a single incremental update, our system automatically continues to refine the image using equations 7.4 and 7.5 until $\mathbf{c}_{effective}$ equals $\mathbf{c}_{perfect}$.

7.1 Use in BRDF editing

Rendering is real-time, at about 25fps, and with rapid interactive switching between curves if the user desires. We typically use 30 wavelet coefficients for each frame. Our incremental rendering algorithm ensures essentially reference-quality images while

$T=(3, 5, 2)$		<i>Incremental Non-Linear</i>						<i>Non-Linear</i>	
t	$c_{perfect}$	R_{perf}	c_{diff}	\tilde{c}_{diff}	$T \cdot \tilde{c}_{diff}$	R_{effec}	$c_{effective}$	\tilde{c}_{perf}	$T \cdot \tilde{c}_{perf}$
0		0			0	0			0
1		24			12	12			12
2		29			15	27			12
3		22			-5	22			12

Tab. 7.1: Didactic Example of Incremental vs. Non-Linear: In this example, the monochromatic color of a single pixel (with T shown in the upper left) is computed at 4 consecutive frames. For instructive purposes, we use the original box functions, so that the coefficients shown can be used to directly envision the curve. We assume that only 1 multiplication is possible per frame, though in real scenarios, 20-30 wavelet terms can be multiplied at each frame. First, we initialize all values to 0. At $t=1$, the user loads in a curve with geometric decay: (4, 2, 1). Next ($t=2$), the user amplifies the center of the curve. Finally ($t=3$), the user decides to shift all but the left part of the curve down by 1 unit. At each frame, Incremental is able to move closer to the perfect result by concentrating all of the computational budget on the difference in the user’s input. By contrast, the standard Non-Linear approximation always concentrates on the coefficient with largest energy.

editing, as seen in Figure 7.1. As with other precomputation methods, rendering time is independent of scene complexity. In addition, our method decouples rendering time from the lighting and shading complexity. Our incremental rendering technique further breaks dependence on even the BRDF resolution (number of bands).

Figure 7.1(a) compares standard and incremental curve approximations for a typical BRDF edit. Figure 7.1(b,c) compares the image quality. The user’s input in Figure 7.1(a) is better approximated by our method because the energy in the peak on the left (more easily seen in Figure 4.5(b) is static, leaving a full budget of wavelet coefficients to approximate the secondary rise. Note that Standard Non-Linear is negative in some regions, leading to a darkening of the diffuse term near the center of the pearls, as seen in Figure 7.1(c).

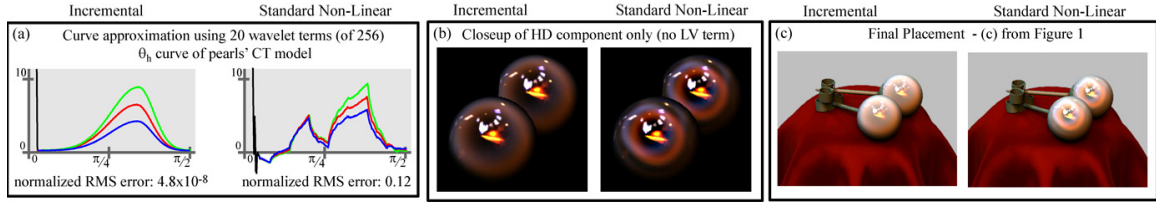


Fig. 7.1: Incremental BRDF Editing: Our incremental method is compared to standard non-linear wavelet approximation. A non-linear wavelet approximation uses most of its wavelet budget for the large peak on the left (visible more clearly in Figure 4.5b), and poorly approximates the colored rise on the right. The incremental images are identical to the reference image when viewed as a 24bit RGB image.

7.2 Use in Relighting

Incremental Wavelet Rendering is particularly useful when a small percentage of the coefficients is updated/edited at each frame. The derivation of IWR at the beginning of this chapter is valid for any rendering operation that uses a dot-product to compose the final images. Relighting systems are therefore poised to benefit from using this technique in conjunction with the existing algorithm. However, when changes are made to a large portion of the coefficients, such as in environment map rotation, a small budget of wavelets may not be enough to update all of those coefficients. Figure 7.2 shows a frames from the rotation sequence of an environment map to demonstrate this.

As we can see from the top and third rows of Figure 7.2, our basic incremental approach is much better than the standard method for approximating lighting with wavelets. However, because rotation is not a local edit, many coefficients change, and the approximation in equations 7.4 and 7.5 falls behind. In order to allow for such global edits, we introduce Per-Band Incremental Wavelet Rendering.

We base our extension to IWR by recognizing that it is sometimes better to build up an approximation from zero, as opposed to incrementally updating by canceling



Fig. 7.2: **Incremental Relighting:** Basic Incremental is the method described in this chapter. Per-Band Incremental is described in (Overbeck *et al.*, 2006)

out coefficients which are no longer valid, and updating others to their new value. However, this is not equally true of all the wavelets in the hierarchy. The largest wavelets (lowest frequency) need to be updated less often. Also, there are fewer of them, so it is easier to keep them up to date. The smallest wavelet band (highest frequency) contains half of the wavelets (1000's) and require a lot of updates. A pathological case would be a checker-board pattern that is shifted by 1 pixel. All of the high-frequency wavelets would have to be updated, but none of the other wavelets coefficients would change.

We separate the basis approximation into the different frequency bands, and dy-

namically decide which bands to treat incrementally, and which ones to treat traditionally. As seen in the bottom row of Figure 7.2, zeroing out the high frequencies and only updating the lower ones successfully produces an approximation without artifacts that is still better than the standard wavelet approximation (top row). The details of selecting how to treat each band, and how many coefficient updates to allocate to it can be found in (Overbeck *et al.*, 2006).

8. VISIBILITY COHERENCE FOR PRECOMPUTATION

Over the past decade, environment maps have increased in popularity as a source for realistic, complex lighting. Today, they are common in computer graphics workflows due to the ease of acquisition and representation. This chapter addresses the problem of efficiently calculating shadows from environment map lighting. Since accurate rendering of shadows from environment maps requires hundreds of samples, the expensive computation is determining visibility from each pixel to each light direction, such as by ray-tracing. We show that coherence in both spatial and angular domains can be used to reduce the number of shadow rays that need to be traced. Specifically, we use a coarse-to-fine evaluation of the image, predicting visibility by reusing visibility calculations from four nearby pixels that have already been evaluated. This simple method allows us to explicitly mark regions of uncertainty in the prediction. By only tracing rays in these and neighboring directions, we are able to reduce the number of shadow rays traced by up to a factor of 20 while maintaining error rates below 0.01%. Figure 8.3 shows how a high-dynamic range environment map is partitioned into areas, with visibility calculated at the centroid of each area. Figure 8.5 shows how visibility information from one pixel may be leveraged to predict visibility of the lighting at a nearby pixel. Our method uses a unanimous vote from 4 neighboring pixels to determine visibility of lights in the environment. Where consensus is not reached, explicit ray-tracing is triggered. For more details, please refer to (Ben-Artzi *et al.*, 2006a)

8.1 Use in BRDF Editing

The previous chapter discussed a new method for improving the performance of the runtime rendering operations described in Part II. This Chapter concentrates on a method that can accelerate the offline portion of the precomputation. We implemented the coherence-based visibility sampling to speedup our precomputation described in Chapter 4. While the visibility calculations were reduced to about 10% of their original time, the entire precomputation with scene loading, acceleration structure construction, and primary rays was reduced to about 30%. While precomputation is assumed to be a lengthy offline process, increasing its speed can greatly add to the practicality of precomputation-based methods.

8.2 Use in Offline Still and Animation Rendering

The method described in Section 8 (and (Ben-Artzi *et al.*, 2006a)) is applicable to any computation that is pixel-based. Extending it to vertex-based precomputation remains for future work. In addition, it can be used directly to generate still images and videos. Figure 8.1 shows comparisons for some scenes as rendered with ground truth, and using approximately 10% of shadow-ray tracing.

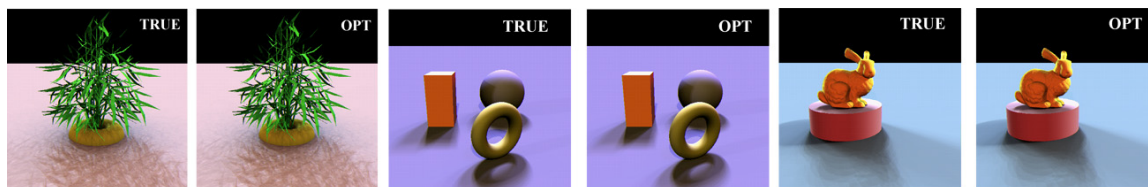


Fig. 8.1: Fast Visibility Ground Truth Comparisons: Rendering comparison using Coherence Based Visibility Testing

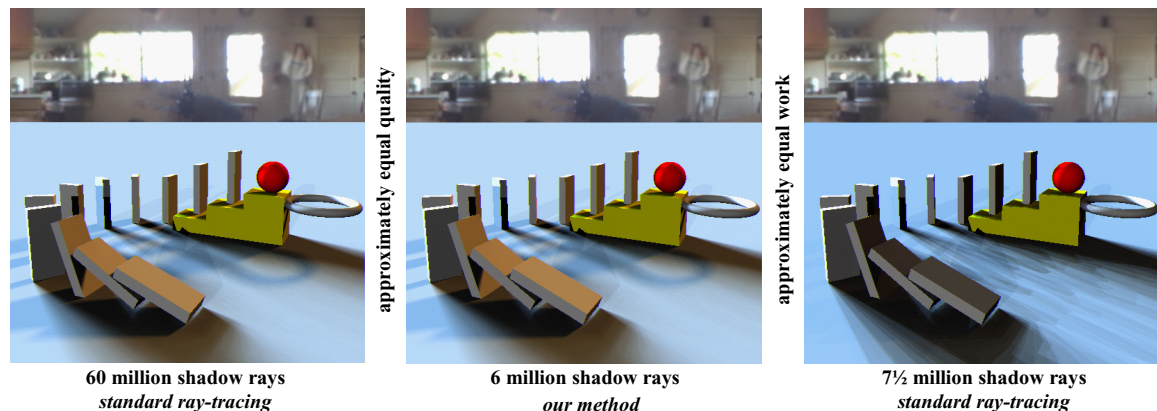


Fig. 8.2: **Domino Scene Comparing Visibility Calculations:** A scene illuminated by a sampled environment map. The left image is rendered in POV-Ray using shadow-ray tracing to determine light-source visibility for the 400 lights in the scene, as sampled from the environment according to (Agarwal *et al.*, 2003). The center image uses our Coherence-Based Sampling to render the same scene with a 90% reduction in shadow-rays traced. The right image is again traced in POV Ray, but with a reduced sampling of the environment map (50 lights, again using (Agarwal *et al.*, 2003)) to approximate the number of shadow-rays traced using our method. Note that the lower sampling of the environment map in the right image does not faithfully reproduce the soft shadows.

8.3 Introduction

Complex illumination adds realism to computer-generated scenes. Real-world objects hardly ever exist in environments that are well approximated by a small number of lights. We therefore focus on realistic shading and shadowing from environment maps, as first described by (Blinn & Newell, 1976), and later by (Miller & Hoffman, 1984), (Greene, 1986), and (Debevec, 1998). The subtle shadowing effects created by complex illumination are one of the most important visual cues for realistic renderings. Correct shadowing requires solving for the visibility of each light direction with respect to each pixel. This is extremely expensive, since the illumination can come from any direction in the environment map. We leverage recent sampling methods like (Agarwal *et al.*, 2003), (Kollig & Keller, 2003), and (Ostromoukhov *et al.*, 2004), that reduce the environment to a few hundred non-uniform directional lights. However, shadow

testing for all sources at each pixel is still the bottleneck in a conventional raytracer or other renderer.

In this chapter, we show how to calculate the visibility function efficiently by exploiting coherence in both the angular and spatial dimensions. We recognize that the expensive operation when lighting with complex illumination is the tracing of secondary shadow-rays. Thus, in our method, we trace all primary eye-rays and cull only shadow-rays, whenever possible. Our method is closest to that of (Agrawala *et al.*, 2000), who exploited coherence in the visibility of an area light source to reduce the number of shadow-rays cast. We adapt their method to environment maps, and show that modifications to both the spatial sharing and angular sharing of visibility information are required for accurate and efficient visibility sampling in the context of environment lighting. The salient differences from area lighting are the lack of a well-defined boundary, lack of locality, and lack of efficient uniform sampling for environment lighting. By addressing all of these points, we demonstrate, for the first time, how coherence-based visibility sampling can be used for full-sphere illumination.

We identify two important components of an algorithm estimating visibility by reusing previously calculated visibility, and explicitly tracing new shadow-rays to correct errors in the estimate. By correctly sharing information among pixels, as well as light directions, we show that accurate visibility predictions can be achieved by tracing only about 10% of the shadow-rays that would normally be required by a standard raytracer without loss in accuracy, as shown in Figure 8.2. Working source code, data, animations, and scripts to recreate all images are available online at the address listed at the end of this chapter.

8.4 Background

Previous work describes coherence-based methods for rendering scenes illuminated by point or area lights. Most of these techniques find image-space discontinuities and share visibility or shading information within regions bounded by discontinuities, thus reducing the number of primary eye rays (and recursively, the number of shadow-rays). We discuss some of these methods below, and discover that image-space radiance interpolation is not well-suited for complex illumination. Furthermore, an efficient method cannot rely on the presence of a light boundary, the locality of the light(s), nor the ability to uniformly sample the lighting. When these assumptions are broken, existing methods often degenerate to an exhaustive raytracing for both eye- and shadow-rays.

(Guo, 1998) was able to efficiently render a scene lit by area light sources by finding radiance discontinuities in image-space. A sparse grid of primary rays was fully evaluated, and radiance discontinuities were discovered by comparing the results of these calculations. The rest of the image pixels are bilinearly interpolated, thus culling the actual primary and secondary rays for those pixels. Unlike area light sources, environment map illumination produces continuous gradients instead of discoverable discontinuities. The method used by (Guo, 1998) would incorrectly interpolate large regions of the scene that in fact have smooth, yet complex variations in radiance.

(Bala *et al.*, 2003) also interpolate radiance in image-space, but determine the regions of interpolation by examining the visibility of lights. Each pixel is classified as empty, simple, or complex, depending on whether it contains 0, 1, or more discontinuities of light visibility. Only complex pixels are fully evaluated. When rendering images illuminated by a sampled environment map, nearly every pixel becomes complex. This is due to the presence of hundreds of light sources, several of which exhibit

a change in visibility when moving from one pixel to the next.

(Hart *et al.*, 1999) demonstrated an analytic integration method for calculating irradiance due to area light sources. They find all light-blocker pairs, exploiting coherence of these pairings between nearby pixels. The lights are then clipped against the blocker(s) to produce an emissive polygon corresponding to the visible section of the light source. The light-blocker pairs are discovered using uniform sampling of the solid angle subtended by the area light source, taking advantage of the light’s locality. For an environment map which subtends the entire sphere of directions, this method degenerates to brute-force visibility sampling.

(Agrawala *et al.*, 2000) achieve their speedup by assuming that visibility of the light source is similar for adjacent pixels. They determine visibility of regularly spaced samples of the area light source, and then share that visibility information with the subsequent pixel of the scanline order. Errors are corrected by explicitly tracing new shadow-rays to light samples that lie along visibility boundaries (and recursively flooding the shadow traces when errors are detected). Locality of the area light source and a well-defined edge ensured that the number of samples was small, and boundaries were well-defined.

Our work also relates to much recent effort on real-time rendering of static scenes illuminated by environment maps. Precomputed radiance transfer methods such as (Ng *et al.*, 2003) and (Sloan *et al.*, 2002) have a different goal of compactly representing and reusing the results of exhaustive calculations. They perform a lengthy precomputation step during which standard raytracing is used to compute the radiance transferred to all parts of a static scene. In contrast, we focus on computing the visibility of an arbitrary scene more efficiently. PRT methods may benefit from using our algorithm in the raytracer they employ for the precomputation stage.

8.5 Preliminaries

In this section, we briefly describe how our method relates to rendering engines that solve the reflection equation. The exit radiance at a point, x , in the direction ω_0 , is given by

$$L(\mathbf{x}, \omega_0) = \int_{\Omega} f_r(\omega, \omega_0) L(\omega) V(\mathbf{x}, \omega) (\omega \cdot n) d\omega,$$

where \mathbf{x} is a location in the scene with a BRDF of f_r , n is the normal, and ω is a direction in the upper hemisphere, $\Omega_{2\pi}$. $V(\mathbf{x}, \omega)$ is the 4D, binary visibility function of the scene. The *light-space* of each location, \mathbf{x} , is the set of directions in the upper hemisphere, Ω , centered at \mathbf{x} . $V(\mathbf{x}, \omega)$ assigns to each direction in the light-space of \mathbf{x} , a binary value indicating whether occluding geometry exists in that direction, or not. (Agarwal *et al.*, 2003) demonstrated that the integral above can be approximated with an explicit sum over a set of well-chosen directions $\{\omega_i\}$, with associated regions or cells, as shown in Figure 8.3 *right*. An efficient approximation of several hundred directional lights can be found for most environment maps, and we assume that such a technique has been used. Often, the hardest part of solving for $L(\mathbf{x}, \omega_0)$ is computing the visibility term, $V(\mathbf{x}, \omega)$. Our work concentrates on making this visibility calculation more efficient. Any system which can substitute its computation of $V(\mathbf{x}, \omega)$ in the above equation can make use of the methods described in this chapter. We demonstrate that a sparse set of samples in $\{\mathbf{x}\} \times \{\omega\}$ can be used to reconstruct $V(\mathbf{x}, \omega)$ with a high degree of accuracy.

Given a set of directional lights, we partition the set of all directions into regions that are identified with each light, as seen in Figure 8.3 *right*. This is done by uniformly sampling a unit disk and projecting each sample onto the unit sphere to obtain a direction. The sample is then assigned the ID (visualized as a color) of the

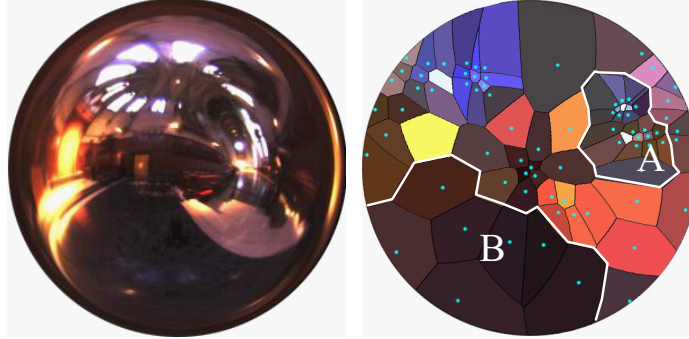


Fig. 8.3: Voronoi Regions: The environment map for Grace Cathedral (left) is sampled into 82 directional lights. Each light is surrounded by the directions nearest to it, as defined by the Voronoi diagram (right). Note that the sampling is irregular, with region A being densely sampled and region B containing sparse samples.

light that is closest to the obtained direction, thus constructing the Voronoi diagram. To improve the resolution of regions near the edge of the disk, a transformation such as parabolic mapping ((Heidrich & Seidel, 1998)) could be used. In our experience, even for environment maps sampled at 400 lights, with the smallest cells near the horizon, a 513×513 pixel Voronoi diagram suffices. (As presented here, only directions where $y > 0$ appear in the diagram. The extension to the full sphere with a second map is straightforward.) We will see that a notion of adjacency in the light-space is needed. Since the lights are not evenly spaced, we identify all the neighbors of a light region by searching the Voronoi diagram. Since the connectivity of Voronoi regions is the dual Delaunay triangulation, the average connectivity of lights in our sampled environments is 6. Because we are using environment maps that represent distant light sources, the Voronoi diagram, and the list of neighbors for each light cell is precomputed once for a particular environment.

8.6 Algorithm Components

Leveraging coherence to estimate visibility occurs in two phases. First, we construct an approximate notion of visibility based on previous calculations (Section 8.6.1).

Then we must clean up this estimate by identifying and correcting regions that may have been predicted incorrectly (Section 8.6.2). In this section, we present several choices for components that provide these functions. The next section (Section 8.7) will analyze the possible combinations to determine the best algorithm for both correctness and efficiency. For the rest of the discussion we will refer to a pixel, and the scene location intersected by an eye-ray through that pixel, interchangeably.

8.6.1 Estimating Visibility

Our goal is to reuse the points of geometry that represent intersections between shadow-rays and objects in the scene. To predict the visibility at one spatial location, we consider the occluding geometry discovered while calculating visibility at nearby locations, thus taking advantage of spatial coherence. We must choose which previously evaluated pixels will be most helpful when attempting to reuse visibility information. We explore the options of scanline and grid-based pixel evaluations.

Scanline Ordering (S)

When evaluating the pixels in scanline order, we use the previous pixel on the scanline, as well as the two pixels directly above them, on the previous scanline. The reasoning behind this is that visibility changes very little from pixel to pixel, and thus using the immediate neighbors gives a good first approximation to the visibility at the current pixel.

Grid-based Ordering (G)

Scanline ordering has the notable limitation that all of the previously evaluated pixels come from the same general direction (up, and to the left). Therefore, we present an alternative which always allows us to reuse visibility from 4 previously evaluated pixels

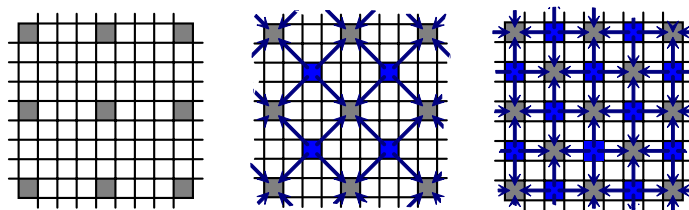


Fig. 8.4: Grid-Based Evaluation: First, the image is fully evaluated at regular intervals. This produces a coarse grid (left). Next, the center (blue) of every 4 pixels is evaluated based on its surrounding 4 neighbors (gray). All 4 are equidistant from their respective finer-grid-level pixel (center). Again, a pixel is evaluated at the center of 4 previously evaluated pixels (right). At the end of this step, the image has been regularly sampled at a finer spacing. This becomes the new coarse level, and the process repeats.

each one from a different direction. In this case, the pixel location for which visibility is being evaluated is the average of the pixels that are informing the estimation process. To a first approximation, the visibility can also be taken to be the average of the visibility at the 4 nearby pixels. This is similar to the coarse-to-fine subdivision first introduced by (Whitted, 1980) for anti-aliasing. The precise pattern (detailed in Figure 8.4) is chosen here to maximize reuse of information from already-evaluated pixels. We have found that an initial grid spacing between 32 and 8 pixels generally works well for minimal overhead.

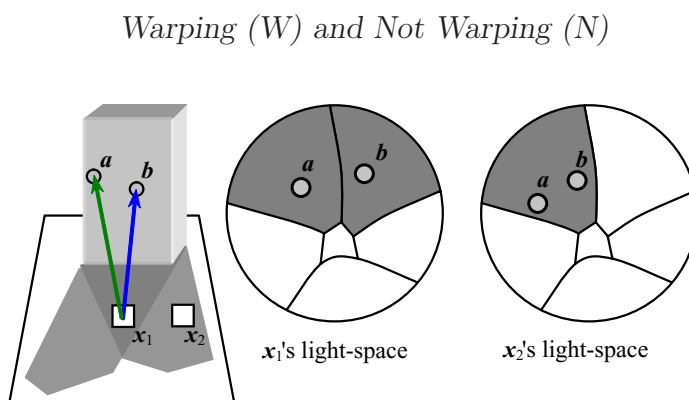


Fig. 8.5: Warping Blockers: Visibility for x_1 is computed by tracing shadow-rays and finding a and b . The visibility for x_2 is predicted by warping blocker points a and b to their positions in the light-space relative to x_2

When blockers are distant, the visibility of a light source at a given pixel can be assumed to be the same at a nearby pixel. However, when blockers are close to the surface being shaded, it may be advantageous to warp the direction of a blocker in the nearby pixel's light-space, to a new location in the light-space of the pixel being shaded, as in Figure 8.5.

Though mathematically correct, warping does not produce a completely accurate reconstruction of blocked regions. As in any image-based rendering (IBR) scheme, warping results in holes, and other geometry reconstruction errors. Estimation errors must be corrected, as described in the next section. We have discovered through empirical testing that this correction method can work well both when warping, and when not warping the blockers from nearby pixels. In both cases, the initial visibility estimation contains enough accurate information to allow for good error correction.

Constructing the Visibility Estimate

The visibility estimate for a pixel is constructed as follows: Each pixel is informed of blockers that were discovered in its neighbors (scanline or grid-based). The direction to these blockers is known, and may be warped to adjust for disparity between the pixel and its informers. At this point the pixel is aware of as many blockers as existed in all of its neighbors, combined. Each blocker is assigned to one of the cells in the light-space of that pixel. (By using the precomputed Voronoi diagram discussed in Section 8.5, finding the cell corresponding to any direction is a simple table lookup.) Some cells receive many blockers, some few, while others may receive none, as seen in Figure 8.6.1. The following subsection discusses how to interpret and correct this data.

8.6.2 Correcting Visibility Estimates

We have shown that spatial visibility coherence among nearby pixels can be used to make educated estimates of visibility. In this section we discuss how angular coherence among nearby directions can be used to correct errors produced by the estimation phase. Once some estimate of visibility exists for the light-space of a pixel, we would like to identify regions of the light-space that may have been estimated incorrectly, and fix them.

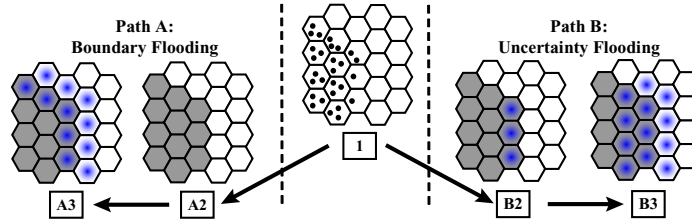


Fig. 8.6: Light-space Flooding: 1: The blockers (black dots) from 3 neighboring pixels (not shown) are placed in the light-space of the current pixel. Each hexagonal cell represents a cell in the Voronoi diagram that partitions the light-space. **Path A.** A2: Light cells which contain more blockers than the threshold (1) are predicted as blocked (grey), while others are predicted as visible (white). A3: Light cells whose neighbors' visibility differs are shadow-traced (blue center). **Path B.** B2: Light cells that contain many (in this case 3 since we are using 3 previous pixels) blockers are predicted as blocked. Those that contain no blockers are predicted as visible. Light cells containing few (1 or 2) blockers are marked as uncertain and shadow-traced (blue center). B3: The neighboring light cells of all uncertain cells are shadow-traced.

Boundary Flooding (B)

Any cell that contains more blockers than a given threshold is assumed to be blocked (Figure 8.6.A2). Our experience shows that the exact value of this threshold has little effect on the final algorithm, and we set it to 0.99. This estimation of visibility is prone to error near the boundary of cells which are visible, and cells which are blocked. This is because such a boundary is likely to exist, but may not lie exactly where the estimate has placed it. We attempt to correct these errors by explicitly casting

shadow-rays in the directions that lie along a boundary (Figure 8.6.A3). Wherever the shadow-ray indicates that an error had been made in the estimation stage, we recursively shadow trace all the adjacent regions. This process continues until all shadow-rays report back in agreement with the estimate.

Uncertainty Flooding (U)

Instead of guessing that boundaries are prone to error, we can use a more explicit measure of when we are uncertain of the visibility constructed by the estimation phase. Cells that contain many (4 in grid-based, 3 in scanline) blockers are considered blocked because all of the informing pixels agree that it is blocked. Similarly, cells without any blockers are considered visible. Whenever the neighbor pixels disagree on the visibility for any cell of the light-space (indicated by a small number of blockers), we mark it as uncertain (Figure 8.6.B2). We explicitly trace shadow-rays for these regions, and flood outwards, recursively tracing more shadow-rays when the estimation disagrees with the shadow-ray (Figure 8.6.B3).

8.7 Analyzing Possible Component Combinations

So far, we have developed four variations of the first phase: estimating visibility. We can use scanline evaluation (S) or grid-based evaluation (G), and for each of these, we may choose to warp (W) or not warp (N) the blockers from nearby pixels. We have also developed two variations for the second phase: correcting the estimate. We can trace shadow-rays at the visibility boundaries (B) or at the explicitly marked uncertainty regions (U). We performed comprehensive evaluations of all eight possibilities, and will now present the findings of the components' usefulness, both on their own, and as they interact with the other variations.

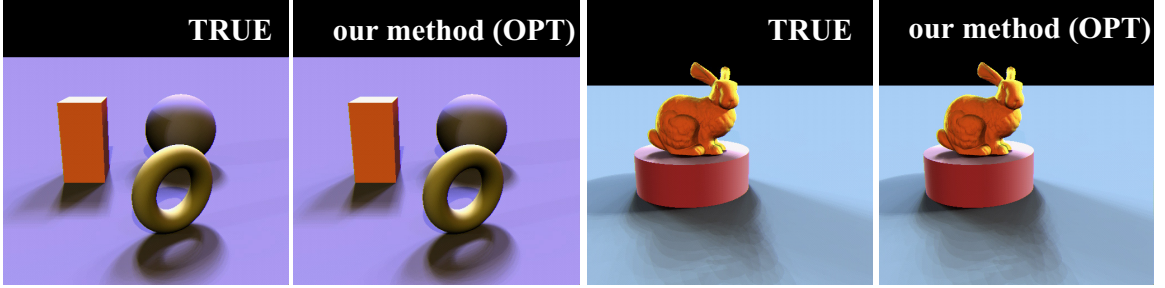


Fig. 8.7: **Shapes and Bunny Ground Truth Comparisons:** Shapes and bunny, lit by 200 lights. OPT is indistinguishable from TRUE.

8.7.1 The Scenes

We analyze the results of different combinations of the framework components for 4 scenes shown in Figures 8.2, 8.7, and 8.9. All tests were run on 513×513 images. The scenes contain only diffuse, non-textured objects to maximize the visibility of cast shadows. The lighting environments are high-dynamic range light probes from www.debevec.org, sampled into point lights according to (Agarwal *et al.*, 2003). We enhance the visualization of shadows by increasing the intensity of lights in small regions. Each scene uses a different environment map. The *shapes* scene shows simple objects with both smoothly varying and discontinuous curvature, with curved cast shadows. Its environment contains several tight clusters of bright lights. The *bunny* represents a scene with a common combination of simple parametric objects and complex curved geometry with self-shadowing. The *dominos* show a scene with many separate objects that interact, resulting in fairly detailed shadows, as might be found in a typical animation. The bunny and dominos environments contain small lights near the horizon, as well as non-uniform lights from above. The *plant* scene includes detailed geometry that casts very intricate shadows with low coherence. Its environment is relatively uniform. A representative example of the data we collected for these scenes is shown in Figure 8.8 — a close-up of the shapes scene.

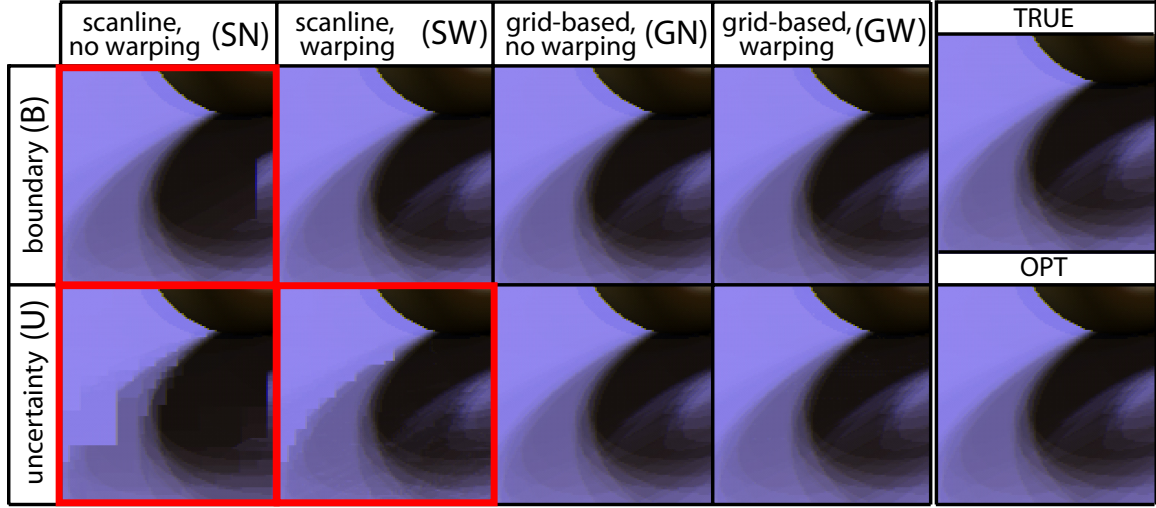


Fig. 8.8: Error Analysis of Visibility Coherence Components: The above table shows possible combinations of the two components of our framework as described in Section 8.6. The *TRUE* image is provided for comparison. This is a close-up of the torus shadow in the shapes scene, as illuminated by a 100 light approximation. The *OPT* result, as described in Section 8.8, is shown in the bottom-right. No perceptible errors exist. Significant errors are visible in those combinations outlined in red, whereas the other renderings contain very minor errors, only noticeable under high magnification.

8.7.2 Correctness Analysis (Scanline vs. Grid)

By examining Figure 8.8, we notice that only one of the combinations which uses scanline evaluation order was able to produce reasonable results: SWB. For boundary flooding, the change in visibility may occur away from any existing boundaries. This is exactly the case that the torus reveals, and the reason its shadow is filled in Figure 8.8:SNB. For uncertainty flooding, scanline’s informing pixels are too close together to have very different notions of the scene geometry, and no correction is initiated in the form of uncertainty flooding. Hence, there can be staircasing artifacts as seen in Figure 8.8:SNU. Even with warping, uncertainty flooding from the three pixels immediately to the left and above does not suffice, although it performs somewhat better, as seen in Figure 8.8:SWU. Thus, we must use warping and boundary flooding in scanline algorithms. It is interesting to note that this corresponds closely

to the approach of (Agrawala *et al.*, 2000).

All four grid-based possibilities (Figure 8.8:G--) give almost error-free results. The coarse-to-fine nature of grid-based evaluation is robust in the presence of different scales of visibility events. In addition, by using the grid-based approach, boundary and uncertainty regions are marked conservatively enough to allow the correction stage to fix nearly all the errors of the estimate. The details of each case are discussed in the next subsection.

8.7.3 Performance Analysis

	shapes				bunny				plant				dominos			
lights:	50	100	200	400	50	100	200	400	50	100	200	400	50	100	200	400
TRUE	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
SWB	40.5	34.3	26.3	22.4	52.4	42.6	33.6	25.9	63.2	51.6	44.3	39.7	54.5	43.6	36.0	28.8
GWB	22.0	17.9	15.1	12.4	21.5	16.6	13.5	11.6	37.8	34.5	32.6	30.5	31.6	28.2	24.9	21.8
GNB	21.7	17.1	13.8	10.6	21.2	15.8	12.0	9.3	37.7	34.6	32.7	30.6	30.3	25.7	21.9	18.0
GWU	10.4	10.7	11.0	11.3	9.1	9.4	10.2	11.5	33.3	32.4	32.3	31.1	20.4	22.3	22.2	22.6
GNU	9.4	8.8	8.3	7.5	8.1	7.7	7.2	6.7	35.3	33.8	33.4	31.9	17.3	16.9	16.3	15.3
OPT	4.8	4.6	4.5	4.3	4.6	4.4	4.1	3.9	21.4	20.9	20.8	20.2	9.4	9.2	9.1	8.8

Tab. 8.1: Percentage of Shadow-Rays Traced: The entries for the table indicate the percentage of shadow-rays where $N \cdot L > 0$ that were traced by each method. For each scene an environment map was chosen, and then sampled as approximately 50, 100, 200, and 400 lights.

The previous subsection concluded that three of the scanline algorithms are error-prone (outlined in red in Figure 8.8), and we do not consider them further. We now examine the efficiency of each of the remaining combinations (SWB, GWB, GWU, GNB, GNU). Tables 8.1 and 8.2 compare the performance for each of the four scenes. Table 8.1 reports the percentage of shadow-rays traced for each method, at different light sampling resolutions of the scenes' environments (50, 100, 200, 400). Table 8.2 shows the actual running times for rendering each scene with 200 lights, as timed on an Intel Xeon 3.06GHz computer, running Windows XP. The times reported in

environments sampled at 200 lights	shapes		bunny		plant		domino	
	sec	speedup	sec	speedup	sec	speedup	sec	speedup
primary rays only (EYE)	4.0	N/A	4.0	N/A	4.4	N/A	4.2	N/A
ground truth (TRUE)	75.3	1.0	80.8	1.0	152.3	1.0	124.9	1.0
scanline, warping, boundary (SWB)	34.9	2.2	54.8	1.5	120.3	1.3	67.1	1.9
grid-based, warping, boundary (GWB)	25.2	3.0	36.7	2.2	113.2	1.3	55.7	2.2
grid-based, no warping, boundary (GNB)	16.8	4.5	23.2	3.5	95.1	1.6	35.4	3.5
grid-based, warping, uncertainty (GWU)	19.6	3.8	30.6	2.6	109.0	1.3	47.1	2.6
grid-based, no warping, uncertainty (GNU)	11.2	6.7	16.7	4.8	94.7	1.6	23.8	5.2
optimized algorithm (OPT)	7.1	10.5	10.7	7.6	58.3	2.6	14.7	8.5

Tab. 8.2: **Visibility Timings:** The wall-clock timings for the full rendering with 200 lights, shown here, are related to, but not equivalent to the reduction in work as measured in shadow-rays traced (see Table 8.1).

Table 8.2 include the (relatively small) time for the initial setup of the raytracer and tracing just the EYE rays for each pixel, as well as performing the diffuse shading calculation. The errors in the five remaining algorithms are generally small ($<0.1\%$) either when measured as absolute mispredictions, or when viewed as the resulting color shift in the image. All of these algorithms enable significant reductions in the number of shadow-rays traced, with negligible loss in quality.

To Warp or Not to Warp

The motivation for warping blockers into the local frame of a pixel is that it may produce a better approximation of the occluding scene geometry. This turns out to be counterproductive for non-uniformly distributed samples of the light-space. Consider Figure 8.3 *right*. When the blockers in an area of densely sampled lights (region A), are warped onto an area of sparse lights (region B), large cells in region B that are mostly visible may receive a large number of blockers, clustered in a corner of the cell. These would be incorrectly estimated as blocked. Symmetrically, if warping occurs from a sparse region to a dense one, cells that are actually blocked may not receive any blockers and be erroneously estimated as visible. When the spacing of

the light samples is not uniform, warping can introduce more errors in the visibility estimation, requiring more work to be done in the correction phase.

When comparing **GWU** with **GNU** and **GWB** with **GNB**, Table 8.1 shows that the non-warping algorithms perform slightly better for all but the plant scene. When comparing the warping algorithms (**GWB**, **GWU**) to the equivalent non-warping algorithms (**GNB**, **GNU**) in Table 8.2, we can see a significant speedup in the wall-clock timings. Not warping requires fewer instructions since a simple count of the blockers in each cell suffices, whereas warping must perform the 3D transformation of the blockers and project them into the new cells. One additional advantage of not warping is that we eliminate the dependence on accurate depth information for the blockers. This can be useful when rendering engines optimize the intersection test of shadow-rays by stopping as soon as an intersection is found, without discovering the true depth of the closest blocker. (To obtain controlled comparisons, we did not turn on this optimization.)

Boundary Flooding vs. Uncertainty Flooding

Since the illumination from an environment map spans the entire sphere, many visibility boundaries can exist, resulting in flooding to most of the lights when boundary flooding is used. Uncertainty flooding has the advantage of being restricted to those areas which are visible at some neighboring pixels, and blocked at others. It is particularly well suited to combat the sampling discrepancies described in the previous subsection (see Section 8.7.3 and Figure 8.3). When under- or over-sampling of the light-space occurs, some cells in that region receive few (1-3) blockers, and trigger explicit shadow traces for their neighbors. Marking cells as uncertain is equivalent to finding image-space discontinuities for individual shadows. If such a discontinuity passes through the area surrounded by the informing pixels, that light is marked

as uncertain in the current pixel. In essence, uncertainty flooding concentrates on regions with image-space discontinuities, while boundary flooding concentrates on light-space discontinuities.

Looking at Table 8.1, we see that in all cases, uncertainty flooding performs about twice as well as the corresponding boundary flooding algorithm, at the lower resolutions. In particular, the first row (shapes50), shows **GWB** doing 22% of the work and **GWU** doing only 10%. Similarly, **GNB** does 22% of the work, while **GNU** does only 9%. This indicates that marking boundaries as regions that require shadow-tracing is an overly-conservative estimate. In our experience, not all visibility boundaries are error-prone.

Since the work done by boundary flooding is proportional to the area of the light-space occupied by boundaries, its performance is strongly dependent on the sampling rate of the environment. Notice that for shapes, bunny, and dominos, boundary flooding algorithms degrade by a factor of 2 as the light resolution is decreased from 400 to 50. On the other hand, the performance of algorithms based on uncertainty flooding is relatively independent of the number of lights.

8.7.4 Discussion of Analysis

The examples shown in Figure 8.8 indicate that scanline evaluation order is much more constrained than grid-based. This is primarily because the small increments as the scanline progresses are sensitive to the scale of changes that may occur in visibility. We see in the results outlined in red in Figure 8.8, that a missed visibility event propagates along the scanline until some other event triggers its correction via flooding. It is instructive to compare **GWB** and **SWB** in Table 8.1, and notice that **GWB** is more efficient by about a factor of 2 for most cases. One reason for this is that scanline must consider the lights near the horizon as part of a boundary.

New blockers that rise over the horizon as the scanline progresses are missed without flooding on the lights near the horizon of the pixel’s local frame. On the other hand, grid-based evaluation incorporates information from all four directions. This enables it to mark new visibility events as regions of uncertainty when they occur between the 4 informing pixels (i.e. near the pixel of interest).

Correctness is achieved when the correction phase is given an accurate guess of where errors might exist in the initial estimate. Efficiency is achieved when the original estimate is good, and indicators of possible error are not overly-conservative. For the algorithms that correctly combine the estimation and correction phases, accuracy is always obtained. In difficult scenes that exhibit low coherence (such as the plant), the algorithms degrade in performance, but still produce accurate results.

8.8 *Optimizations and Implementation*

Based on the analysis of the previous section, we determine that GNU is the best combination of the framework components for both accuracy and performance. We also rendered animations for all of our test scenes by rotating the environment to see if any popping artifacts would occur. All of the animations using GNU were indistinguishable from the TRUE animation. We therefore base our optimized algorithm (OPT) on GNU, making it grid-based, without warping, and using uncertainty flooding as the means of correcting errors (Section 8.8.1). We describe the implementation details for adding a GNU-based algorithm to any standard ray-tracing system in Section 8.8.2. Finally, we take an in-depth look at the time and memory usage implications of these methods (Section 8.8.3).

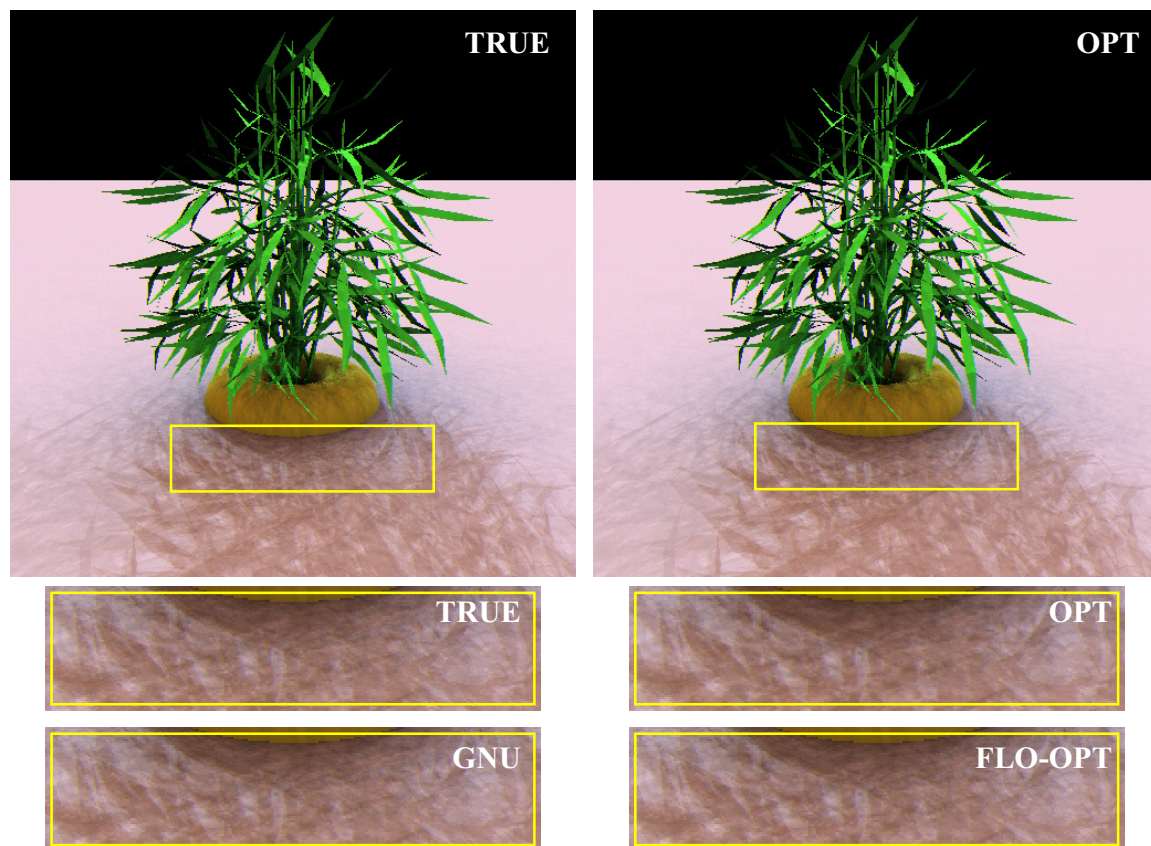


Fig. 8.9: Plant Comparisons of Visibility Algorithms: The plant image, seen here as illuminated by a 100 light approximation of the Kitchen environment, has very detailed shadows with low coherence. Nevertheless, our algorithms produce a very accurate image, as seen in the direct comparisons and closeups.

8.8.1 Optimized Algorithm

More than half of the shadow-rays traced by our algorithms are triggered by flooding. We have seen that both boundary and uncertainty flooding are effective methods for correcting prediction errors. A close inspection of the shadow-rays traced due to flooding reveals that many of them agree with the original visibility estimate, and therefore only serve to halt further flooding, without contributing anything to the visibility calculation. We can avoid many such shadow-traces by combining the ideas of uncertainty and boundary flooding. We still shadow-trace all regions of uncertainty as indicated by the original visibility estimate (Figure 8.6.A2). However, instead of

flooding to all neighbors (typically 6) of a cell whose shadow-trace disagrees with the estimate, we only flood to those neighboring cells that also disagree with the shadow-trace result. This means that for uncertainty regions that lie on a border, we only flood onto one side of the border; trying to find where it truly lies. This optimization reduces the number of flood-triggered traces by about 40%. For complex shadows, small artifacts are introduced by this method, as may be visible in the Figure 8.9:OPT close-up, but we feel that the performance gain is worth the slight difference in image quality. Since animations are more sensitive to small differences, we recommend that this optimization only be used for stills, and omitted in animations. (A comparison of these animations for the shapes scene is available online at the address listed at the end of this chapter.)

In an effort to further reduce the number of shadow-rays traced due to flooding, we note that half of the image pixels are contained in the finest grid — when the initial estimate is informed by a pixel’s immediate neighbors. If all 4 neighbors agree on the visibility of a particular light, the center pixel can only disagree if there is a shadow discontinuity smaller than a single pixel passing through it. Since the image cannot resolve elements smaller than a single pixel, we ignore these sub-pixel shadow discontinuities and turn off flooding altogether at the grid’s finest level only (FLO). We still rely on error correction due to explicit uncertainty marking. This reduces the total number of shadow-rays required to calculate the image by about 25%, without adding any noticeable artifacts to the final image, as seen in the Figure 8.9:FLO-OPT close-up. This optimization also is safe for animations, as its effects are only noticeable at a sub-pixel level. Sub-pixel sampling for antialiasing is a popular method employed by many rendering engines. When such a feature is used in conjunction with our method, the highest resolution of sampling should be considered to be the native image resolution for the purposes of defining the grid.

Our optimized algorithm (OPT) uses the above two methods of restricting flooding; flooding only to neighbors that disagree with the shadow-ray, and not flooding at the finest grid level. The high quality of OPT can be seen in Figures 8.2, 8.8, 8.7, and 8.9. The close-ups in Figure 8.8 and Figure 8.9 reveal that OPT produces images nearly indistinguishable from the ground-truth (TRUE). Even on the stress-test provided by the plant scene (Figure 8.9), OPT has only a 0.4% error rate, as measured in actual shadow-ray prediction errors. The error rate for OPT on the bunny is 0.0026%, meaning only 1 in 38,000 shadow-rays disagree from ground-truth. In terms of visual quality of the images OPT produces accurate results for all the scenes. As visualized in the final images, these errors produce rendering that are nearly identical to the true images when viewed as standard 24 bit RGB images. Adding the above two optimizations typically leads to a 50% reduction in the number of shadow-rays traces, when compared to GNU. As reported in Table 8.1, this can result in a factor of 20 improvement in efficiency over standard visibility testing in ray-tracers. In Table 8.2, the timings show that as OPT's performance improves, shadow-tracing is no longer the bottleneck for rendering. In the shapes scene, tracing the primary eye rays (4.0 seconds) takes longer than adding complex shadows to the scene (3.1 seconds).

8.8.2 Implementation

```

1  procedure Calculate-All-Visibilities()
2      for (gridsize = 16; gridsize > 1; gridsize /= 2) do
3          foreach pixel,  $p$ , in current grid do
4               $lights-to-trace \leftarrow \emptyset$ ;
5               $n[1..4] \leftarrow \text{Find-Evaluated-Neighbors}(p)$ ;
6               $object(p) \leftarrow \text{Trace-Primary-Ray-At-Pixel}(p)$ ;
7              if  $object(n[1]) == object(n[2]) == object(n[3]) ==$ 
8                   $object(n[4]) == object(p)$  then
9                  foreach light,  $L$ , in sampled-environment-map do
10                     if  $vis(n[1], L) == vis(n[2], L) ==$ 
11                          $vis(n[3], L) == vis(n[4], L)$  then
12                          $vis(p, L) \leftarrow vis[n[1], L]$ ;
13                     else
14                          $vis(p, L) \leftarrow \text{'uncertain'}$ ;
15                          $lights-to-trace \leftarrow lights-to-trace \cup L$ ;
16                     endif
17                 endforeach
18                 Flood-Uncertainty( $p$ );
19             else
20                 foreach light,  $L$ , in sampled-environment-map do
21                      $vis(p, L) \leftarrow \text{Trace-Shadow-Ray}(p, L)$ ;
22                 endforeach
23             endif
24         endforeach
25     endfor

```

Code 2: **Compute Visibility:** Code for computing visibility for each pixel as in GNU.

Our algorithms are all very simple to add to a conventional ray-tracer or other rendering system. We have done so in the context of a widely available ray tracer, POV-Ray. Adding coherence-based sampling requires few changes to the core rendering engine, and uses only a few additional simple data structures for support. Furthermore, in our experience, care need not be taken to ensure that the implementation is engineered precisely. Even if it does not adhere strictly to our method, it will still provide a vast improvement over full ray-tracing. In particular, GNU and

OPT are very simple to implement. For each pixel, once the 4 nearby pixels are identified in the grid-based evaluation, a simple check of agreement for each light is performed. If all 4 pixels agree that a light is blocked or visible, it is predicted as blocked or visible, respectively. When the 4 pixels do not agree on a particular light, it is shadow-traced, with possible flooding to neighboring lights. If the optimizations are used, only neighbors whose prediction disagrees with the shadow-trace are traced, and no flooding to neighbors is performed at the final grid level. Code for GNU is shown in [procedure Calculate-All-Visibilities](#) (Code 2).

Lines 1–4 describe the use of grid-based evaluation as shown in Figure 8.4. To prime this process, the coarsest level of the grid is evaluated at a scale of 16 pixels, which we found to be suitable for all of the scenes we tested. For each of these pixels (<0.5% of the image), all shadow-rays are traced. This is comparable to the priming for scanline which requires the top row and left column to be fully evaluated (0.4% for 513x513 images). In lines 5 and 6, we perform a sanity check to make sure that only visibility calculations relating to the same scene object are used for estimation. We have found that there is a significant loss in coherence when the informing pixels are not on the same object as the current pixel. We therefore fully shadow-trace around these silhouette edges to determine visibility. For very complex scenes such as the plant in Figure 8.9, explicitly tracing in this situation traces 7% of the shadow-rays. For average scenes like those in Figure 8.2 and Figure 8.7, this results in about 1–3% of the shadow-rays.

In lines 7–11, the initial rough estimate of visibility is formed as shown in Figure 8.6 steps 1 and A2. In line 12, the error correction phase is called, as described in Section 8.6.2, and diagrammed in Figure 8.6.A3. The pseudocode for this is shown in [Flood-Uncertainty](#)(Code 3).

When [Flood-Uncertainty](#) is called, the set *lights-to-trace* contains all those lights

	procedure Flood-Uncertainty(pixel p)
1	while ($lights\text{-}to\text{-}trace \cap \text{set-of-untraced-lights}(p) \neq \emptyset$,
	with an L from the above intersection do
2	$tempVis \leftarrow \text{Trace-Shadow-Ray}(p, L);$
3	if $tempVis \neq vis(p, L)$ then
4	$vis(p, L) \leftarrow tempVis;$
5	foreach of L 's neighbors, N do
6	$lights\text{-}to\text{-}trace \leftarrow lights\text{-}to\text{-}trace \cup N$
	endforeach
	endif
	endwhile

Code 3: **Uncertainty Flooding:** Code for flooding uncertainty until all guesses are verified by ray tracing.

which were deemed ‘uncertain’. None of these have been traced yet. All other lights have had their visibility predicted, but also were not explicitly traced. Line 1 checks to see if there are any lights in *lights-to-trace* which have not yet been shadow-traced. If any such light, L , is found, a shadow-ray is traced from the object intersection for p in the direction of L , at Line 2. At Line 5, the neighbors of L are added to the list of lights to trace. If OPT were being implemented, line 5 would only add those neighbors whose prediction disagreed with $tempVis$. It may seem reasonable to remove a light from *lights-to-trace* once it has been traced, rather than checking against all the untraced lights in Line 1. However, doing so would lead to an infinite loop since Line 6 would add a neighbor, and when that neighbor got traced, it would add the original light.

8.8.3 Time and Memory Considerations

In this section, we evaluate some of the more subtle (yet interesting) aspects of coherence-based visibility sampling.

400 light sampling		shapes	bunny	plant	domino
1	sec to trace TRUE	245.50	232.70	378.10	337.50
2	sec to trace OPT	9.40	14.80	118.90	26.20
3	time speedup	26.12X	15.72X	3.18X	12.88X
4	theoretical speedup	23.20X	24.33X	4.94X	11.38X
5	ratio of speedups	1.13	0.65	0.64	1.13
6	usec/ray in TRUE	3.24	3.52	5.62	5.39
7	usec/ray in OPT	3.05	5.82	8.68	4.93
8	ratio of usec/ray	1.06	0.60	0.65	1.09

Tab. 8.3: Shadow-Ray Timings: Row 5 shows the ratio of real speedup based on wall-clock timings (Rows 1 and 2), relative to theoretical speedup based on the number of shadow-rays traced as reported in Table 8.1. Row 8 shows the ratio of average times to trace a shadow-ray in the unmodified portion of the ray-tracer, with and without our method. A comparison reveals that the perceived loss in speedup is due to the increased average difficulty of tracing a shadow-ray, due to culling of ‘easy’ shadow-rays.

Timings

While the improvements in running time reported in Table 8.2 are significant, in some cases, they are somewhat less than that predicted from the reduction of shadow-rays alone, shown in Table 8.1. We investigate this further in Table 8.3, which focuses only on the time spent tracing shadow-rays. Rows 1 and 2 show the wall-clock time for a full rendering of the scenes using both TRUE (trace all shadow-rays) and OPT. We compare actual and theoretical speedups in rows 3 and 4. Row 5 compares these two speedups. Notice that for bunny and plant, the actual timed speedups are not as good as the theoretically predicted speedup. We measured the average time for just the operation of tracing a shadow-ray, and report these in Rows 6 and 7. We do this by generating a list of shadow-rays traced during the rendering with each method. Those shadow-rays are then retraced in the absence of any other operations. Since both methods use the same unmodified POV-Ray ray-

tracing engine, this is a very controlled experiment. We notice that not all shadow-rays are equally expensive. In particular, shadow-rays that are not occluded generally need to perform only the bounding box intersection test. When a ray intersects an object, or grazes the silhouette of an object, an exact point of intersection must be discovered (or proven non-existent). In our algorithm, we optimize away many of the ‘easy’ shadow tests, tracing primarily the ‘difficult’ shadow-rays. Hence, the average time to trace a shadow-ray in OPT can exceed that in TRUE. Row 8 of Table 8.3 shows the ratio of the average time per ray when all rays are traced (TRUE), versus the average time to trace only rays needed by OPT. A comparison to Row 5 reveals that the perceived loss in speedup is due to the increased average difficulty of tracing a shadow-ray. We also see that the actual computational overhead of our algorithm is negligible. For the shapes and domino scenes, we achieve the theoretical speedups as predicted in Table 8.1.

We performed a similar analysis of timings for algorithms that include warping. In those algorithms, some extra time (20%) is expended on operations such as blocker retrieval and warping, and light-cell identification through lookup in the Voronoi diagram. The added overhead is partially due to cache misses in the memory. The net performance of the grid-based warping algorithms is still a significant gain over standard ray tracing in most cases, as seen in Table 8.2.

Memory

The grid-based algorithms that use warping store all blocker points for reuse. The memory usage for 200 lights, for a 513x513 image, corresponds to approximately 160MB, assuming 20 bytes per blocker. When memory usage is important, an additional 3% of the pixels can be traced to split the image into independent blocks of 64x64. The peak memory usage will then be approximately 2.5MB. This compares

favorably with the size of the geometry for complex scenes, such as the bunny model.

For algorithms that do not warp, such as OPT, the memory consumption can be significantly reduced. In this case, we do not need to store information about the blocker, but simply need one bit for storing past visibility calculations. In these cases, for 200 lights in a 513×513 image, memory usage can be reduced to 5MB. If we were to use independent blocks of 64×64 , as discussed above, the memory consumption can be reduced to a very insignificant 100KB.

8.9 Other Considerations

When strong coherence exists in the visibility function, as in most scenes, efficient evaluation can be achieved using very simple coherence-based methods. It is important to couple predictions based on coherence with an appropriate measure for discovering areas of possible error. IBR uses the techniques of warping and splatting to recreate geometry. After experimenting with splatting (not discussed here), we determined that too much information must be gathered to attempt a highly detailed reconstruction. We were encouraged to start with a rough approximation, and work to create appropriate error-correcting functionality. This approach is aided by the fact that in a ray-tracing environment we can benefit from the added ability to introduce new samples wherever we believe they are needed. A good measure of uncertainty guarantees that even scenes with weak coherence, such as the plant, will be rendered accurately, though at a lower prediction rate. It is important to remember that our method is only as accurate as the sampling of the environment map. If tiny objects exist in the scene such that all shadow-rays miss them, the lighting must be sampled more densely to produce the cast shadows from such objects. It is also possible to create a scene that fools our method into believing that all predictions are correct,

when they are not. This would require an unnatural collusion between the geometry, lighting, camera position, and image resolution. If there is concern that such a grid of objects exists in the scene, boundary *and* uncertainty flooding can be used. While we tested our method with many types of objects, we did not experiment with particle objects that cast shadows, such as hail. We expect real-world scenes with gravity and solid objects to be rendered accurately with our method.

The algorithm we presented considers the prediction of each shadow-ray to be equally important. In real scenes, some lights contain more energy than others, and the cosine fall-off term gives further importance to particular lights. It is straightforward to consider a scheme that catalogs the lights with the most energy for each possible normal direction, and treats those lights specially. Specifically, such lights tend to be small, and thus are more likely to be mispredicted. Adding boundary flooding for only these high-energy lights ensures higher fidelity in the final image. Besides the differing energy of shadow-rays, we saw in Section 6.3.1 that some shadow-rays are more expensive to trace than others. Unfortunately, the work required to trace a ray seems to be inversely proportional to the predictability of its path. This is understandable, and may be used in the future to set a theoretical lower bound for coherence-based ray-tracing.

Our algorithm also considered accuracy to be much more important than efficiency. We presented methods of combining and optimizing the components in a way that often sacrifices efficiency to maintain accuracy. It may be possible to achieve higher speedups if some tolerance for errors exists. We saw in Section 8.8.1 that restricting the flooding can improve performance at the cost of some error. An adjustable level of such flood-restriction may provide more control for accuracy/efficiency tradeoff. However, one must be careful of a naive implementation that would result in unwanted popping artifacts, or other large fluctuations in the image.

9. USER INTERFACE FOR 1D FUNCTION EDITING

Section 3 introduced 1D functions as factors of BRDFs, per equation 1. Sections 3.3 and 3.4 introduced the ability to edit these functions directly, without relying on UCV-driven functions. In this chapter, we explain our system for function editing used throughout this work.

One-dimensional functions are distinct from plane curves in several important ways. First, a function can have only one value in the range for every value in the domain. Second, a function's smoothness and other intrinsic characteristics are independent of its representation. Third, the values of a function are of primary importance and these inform the shape, while with splines, the shape is often primary, defining the values as a consequence. Finally, the types of edits on a function form a different paradigm than splines. Whereas splines use controls such as 'set the location' or 'set the tangent' of a point on the curve, functions use such notions as 'amplify' and 'translate'. For these reasons, we choose to avoid the temptation of fitting splines to our functions, and instead develop a set of higher-order operations designed to work well with 1D functions. Extensions of these to 2D remains for future research.

The two main tools we use for editing functions are Amplify and Translate. Instead of operating on control points, we operate on regions of the function (i.e. intervals of the domain). Each operator defines a primary region in which the operation is applied uniformly, and secondary regions in which the operator's strength diminishes until its effect is non-existent. Outside of any of the regions of influence for an operation,

the user is guaranteed that the function's values will not change. Furthermore, we guarantee that no discontinuities will be introduced at the boundaries of any of these regions. Each operator can act in the X or Y direction. Each operator is a 1D function that takes as input four values in the domain that denote the extent of the primary and secondary regions of influence: a , b , c , and d . Each operator also takes a scalar value for the magnitude of its action. The simplest is the Y -translate operator: $Trans_{Y,mag}^{a,b,c,d}(x)$. The Y -translate is applied by adding it to the function, as shown in Figure 9.1:

$$Trans_{Y,mag}^{a,b,c,d}[f(x)] = f(x) + Trans_{Y,mag}^{a,b,c,d}(x) \quad (9.1)$$

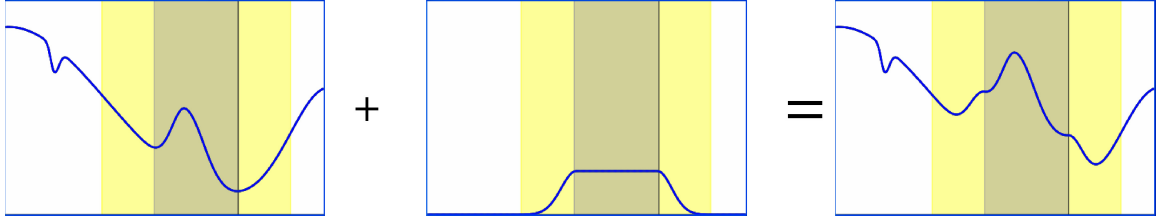


Fig. 9.1: **Y-Translation:** Translation in the Y direction

The helper function, $Trans_{Y,mag}^{a,b,c,d}(x)$ seen in the middle of Figure 9.1, is created from the following parts:

$$Trans_{Y,mag}^{a,b,c,d}(x) \equiv \begin{cases} 0 & \text{if } x < a \\ s_{mag}^{a,b}(x) & \text{if } a \leq x < b \\ mag & \text{if } b \leq x \leq c \\ s_{mag}^{c,d}(x) & \text{if } c < x \leq d \\ 0 & \text{if } x > d \end{cases} \quad (9.2)$$

where $s_{mag}^{a,b}(x)$ is a smooth function with $s(a) = 0$, $s(b) = mag$, and all derivatives of $s(x)$ at a and b are zero. Such smooth functions exist, and are constructed based

on $e^{frac{-1x^2}$. For implementations that only require one or two levels of smooth continuity, other functions can be constructed based on the sine and cosine functions. Details will not be presented in this document. Note that the helper function is the result of applying the translate operator to the line $y = 0$.

The X -translate operator is a composition that changes the rate of x from identity ($x = y$) to a shifted version, as shown in Figure 9.2:

$$Trans_{X,mag}^{a,b,c,d}[f(x)] = f(Trans_{X,mag}^{a,b,c,d}(x)) \quad (9.3)$$

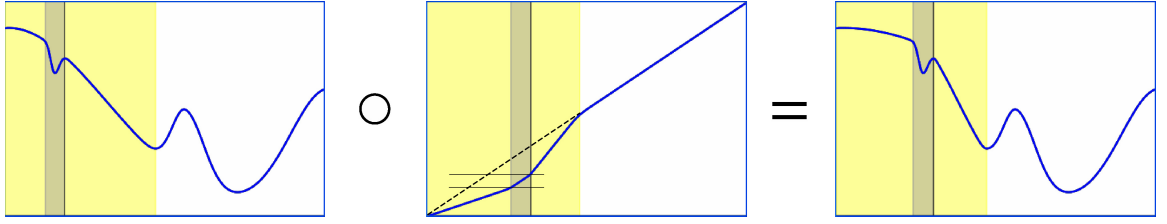


Fig. 9.2: **X-Translation:** Translation in the X direction

Like the Y -translate operator, the X -translate operator is defined in sections, similar to 34. The differences are the values at the boundaries, and the derivative of the first derivative of the smooth filler function must be 1. Like the X -translate, the Y -translate's helper function is an application of the operator itself, but this time to the line $y = x$.

The Y -amplify operator is similar to the Y -translate, but uses a multiplication, and base function $y = 1$, as shown in Figure 9.3:

$$Amp_{Y,mag}^{a,b,c,d,base}[f(x)] = f(x)Amp_{Y,mag}^{a,b,c,d,base}(x) \quad (9.4)$$

The Amplify operators take an additional argument, *base* , that specifies the baseline relative to which the amplification occurs. Again, the helper function is the

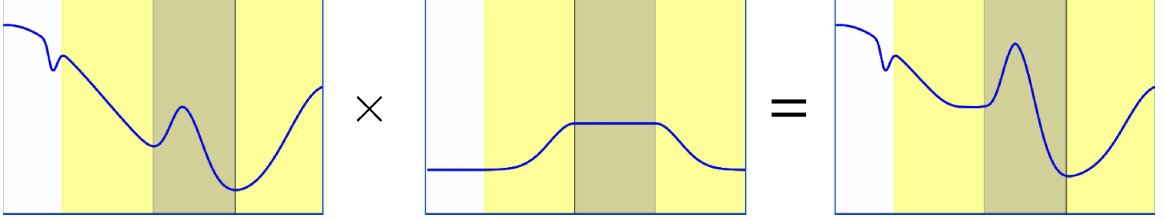


Fig. 9.3: **Y-Amplification:** Amplification in the Y direction

result of applying the Amplify-Y operator to the line $y = 1$.

The final operator is Amplify-X, and takes the form of a composition:

$$Amp_{X,mag}^{a,b,c,d,base}[f(x)] = f(Amp_{X,mag}^{a,b,c,d,base}(x)) \quad (9.5)$$

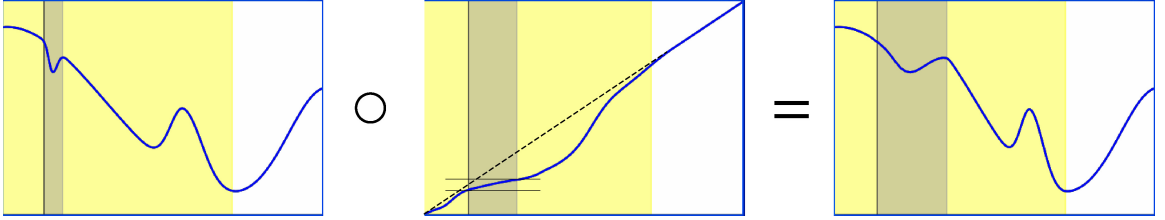


Fig. 9.4: **X-Amplification:** Amplification in the X direction

Like all of the other operators, the helper function for Amplify-X is the result of applying the amplify operation to the basic line ($y = x$).

9.1 Empirical Definition of Helper Functions

Since all of the operations are based on generating a helper function (the second item in the figures), these 1D function edits can be defined empirically by defining what the operation would do to the base curve. For the X-operations, the base curve is $y = x$. For the Y-operations, the base curve is either $y = 0$ or $y = 1$, as shown above. Defining the helper function is sufficient for then applying it to an arbitrary

1D function through addition, multiplication, or compositing, as described in the previous section.

9.2 *Smoothness and Layered operators*

The number of times the result of an operation is differentiable is the min of the original function's continuity and the helper function's continuity. It is possible to build helper functions which are infinitely differentiable (smooth), and therefore applying the operation does not introduce any discontinuities to the original function. This has the obvious exception when the operation has a degenerate condition, such as zero-width regions of influence. The invariance of continuity can be proven using the chain-rule of differentiation for the X operations (function compositions), and by the linearity of the derivative operator for the Y operators (adds and multiplies).

Often, both Amplify or both Translate operators are applied 'simultaneously'. For example, when moving a portion of the function along a diagonal, there is a component of Translate- X and Translate- Y . This can be represented as a layering of operators. It is important to apply the Y operator first. Otherwise, it will not be applied to the correct portion of the function since the X operators change the regions of influence.

While it is theoretically possible to continue layering operations, periodically, it may be beneficial to collapse the edited function into a new function by evaluating it explicitly.

9.3 *Use in BRDF Editing and Other Graphics Applications*

The operations described in Section 9 are natural to most processes that make use of 1D functions. Most previous edits of 1D functions have been performed with splines,

but the unpredictable nature of a spline's shape as control point get close requires a lot of 'counter-editing' to correct for undesirable fluctuations. Image editing software uses 1D functions to define color correction and gamma curves. Animation designers use 1D functions to describe the speed of objects along their 3D path. All of the edits we have shown in Part I operate on 1D functions, and use the operators described above. Any scalar value that varies with time is a 1D function.

Part IV

CONCLUSION

10. CONCLUSION

This work has presented a new capability to render scenes interactively with complex materials, geometry, and lighting (including global illumination) while the user adjusts the material properties of objects in the scene. This *final-placement BRDF editing* is imperative anywhere choices for BRDFs must be made based on how such decisions affect the appearance of objects and their surroundings.

10.1 *Final-Placement*

Traditional workflows for choosing and editing BRDFs have used simplified visualizations, such as spheres lit by a point light. After a material is created in this way, it is stored to be passed into an offline renderer at final frame-generation. The image(s) is then examined, and different choices can be made for the materials, based on how they affect the appearance of the final scene. In most cases, the final-placement appearance of a material is quite different from the simplified visualization. Special effects houses tend to adjust for this difference by placing disembodied lights, or making local tweaks to the material – ultimately leading to the desired shot-by-shot look, but sacrificing consistency and reproducibility. The ideal situation is to be able to choose the materials based on visualizations of the final scene(s). This is the capability provided by the current work, with a tradeoff for flexibility of other scene parameters (such as lighting and viewpoint). This means that the design process must still be iterative - alternating between lighting and material design, but is no longer based on

an iterative cycle (with a long offline rendering step) while in the mode of designing the scene’s materials.

10.2 Theoretical Framework with Few Assumptions

Part II presents a precomputation-based framework for interactive BRDF editing. The precomputation step is founded on a standard path-tracer, and therefore inherits the generality of Monte Carlo path-tracing. Generating the precomputation data is very similar to rendering a single frame. The main difference is not the operations performed, but how the results are accumulated. In traditional rendering, the results of many samples at a pixel are accumulated into a single color. With the BRDF editing framework, each sample’s result is accumulated into the appropriate bin of a high-dimensional tensor at each pixel. The rendering engine’s task is to weight these bins appropriately and collapse their stored values into a single color at runtime. The implication is that the final quality of the rendered images are as good as the offline path-tracer can produce for a single frame. While more samples may be needed to achieve the same signal-to-noise ratio, there is no inherent limitation in the quality achievable by the renderers presented in Part II. Two important qualifications exist. First, the precomputed data takes up much more storage than a single frame, and the resolution of the image and BRDF factors is limited in practice to fit the resources available. Second, intra-pixel integrations such as anti-aliasing and motion blur cannot be directly applied without further theoretical scaffolding. It is important to note that the practical limitations of storage are not limitations of the framework, and the same theoretical mechanisms will be useful for more accurate implementations as computing resources improve.

10.3 *Complement to Relighting*

A similar ‘final-placement’ approach has already been developed for lighting design. Precomputation-based relighting has enabled artists and engineers to interactively manipulate the complex lighting of a scene, while receiving feedback on how these edits affect the final scene with shadows, complex geometry, and complex materials. Just as relighting requires fixing the geometry and materials of the scene, BRDF Editing is the complementary solution that requires fixing the geometry and lighting, while leaving BRDFs unspecified until render-time. Just as relighting systems originally made restrictive assumptions on the nature of the source illumination (requiring a 2D environment map), so must BRDF Editing restrict the editing operation from acting on the full 4D space in which BRDFs live. This work has recommended editing a set of 1D factors that span the space of “editable” behavior for each material. Relighting has advanced to allow near-field illumination changes, using aggressive compression. Unlike any of the relighting works that rely on an approximating basis (such as spherical harmonics) or aggressive compression, the work presented here is able to faithfully render the full nature of a BRDF, up to the degree of discretization of the 1D factors. In Chapter 7, we discussed how the temporal coherence developed for BRDF editing can be used in relighting to recapture the full 2D representation without resorting to compression that may lose many important characteristics of the lighting. Chapter 3 introduced the *quotient BRDF*, which is vital to maintaining a full representation of the BRDF while editing some 1D aspect of its behavior.

10.4 Continuous Editing and Discrete Choices

BRDF Editing is unlike geometric or lighting design in that there is no good real-world analogue to the process. While geometric tools try to mimic the act of clay modeling or manufacturing processes, it is rare that one can change the material of an object by directly interacting with it. Some exceptions to this are sanding and painting, but these operations are often long and difficult to reverse. In addition, even situations that present alternate choices for materials (such as choosing a tile for a new floor), offer discrete choices. In the virtual setting, BRDF editing is a first-order operation (not a byproduct of some other action), and editing can be accomplished along a continuous spectrum. Even the seemingly continuous realm of paint colors does not afford one the option to continuously vary the color of a wall or car (since applying a new paint must be done to the whole surface uniformly.) Therefore, BRDF Editing opens up a new mode of appearance design that has not been previously accessible in real-world settings. This possibility is accompanied by the challenge of finding appropriate cognitive actions for achieving the desired edits. Chapter 9 presented an initial system for editing 1D functions that represent material behavior along some parameterized direction. This has been shown to allow for a wide variety of edits, but does not purport to be a final solution.

11. FUTURE DIRECTIONS

This thesis (and the related published works) has introduced the first general BRDF editing framework for final-placement material design. The BRDF is just one of many material representations — the most complex of which is the 8D BSSRDF. There are two major directions for extending the current work in the future. First, more complex representations such as BTFs and BSSRDFs may be analyzed to produce similar material editing applications. Second, refinements and practical extensions to the current BRDF framework can be added to make the work applicable in a broader range of applications.

11.1 Higher-Complexity Functions

The BRDF is a local approximation to a more complex set of interactions between light and scene objects. More complex behaviors such as those arising from normal/bump/displacement -maps and finally fully 6D BTFs cannot be edited in the current framework. It should be noted that many of these effects can be included into the precomputations as described in Part II, but will not be editable at runtime. One hurdle to creating an interactive editing system that admits these higher-complexity material attributes is that even offline editing of such effects has not been well studied. Just as BRDF editing does not have a real-world analog, BTFs present an even tougher challenge when it comes to developing useful editing paradigms. The most likely manifestation of these editing operations will require finding a representation

built up from low-dimensional factors, each of which can be easily edited by a human user. This is similar to the editable BRDF representation introduced in Chapter 3. Just as that representation includes a complex but non-editable component (the quotient BRDF), so will higher-complexity functions likely include some fixed component and some editable terms.

Subsurface scattering and transparencies with editable index-of-refraction are other examples of departures from the local, opaque assumption of BRDFs. A more immediate extension of the BRDFs used in this thesis is editable anisotropy shape. While Chapter 3 describes how to edit parametric anisotropy, recent work ((Lawrence *et al.*, 2006)) suggests that anisotropic highlights of real-world materials are not oval. Controlling this shape is a task that has not been explored offline nor interactively. Future work in BRDF editing may incorporate such highlight-shape adjustments, once more data is captured to determine the variety of this phenomena in the real world.

11.2 *Practical Extensions*

The work presented in this thesis demonstrates practical implementations of the theoretical framework developed in Chapter 5. Extending the implementation to account for production workflows would add to the practicality of the method. Very little compression is recommended in this thesis. Preliminary experiments with standard compression methods (e.g. non-linear wavelet) for both inter- and intra-pixel data has shown that artifacts quickly become noticeable in the regime of BRDF editing. An examination of Table 6.1 shows that the last bounce of global illumination precomputations requires the most storage. However, that bounce also contributes the least to the quality of the image, and is generally smooth due to the smoothing effect of multiple reflections. Therefore, a compression scheme that allows for constant

importance-to-storage ratios across bounces would seem a fruitful avenue of future research.

One advantage of the absence of compression is that the precomputed data can be passed directly to the render. This suggests a progressive precomputation that allows users to begin working with coarse/noisy data very quickly, while a background process continues to add precomputation data in order to refine the renderings throughout an editing session.

Another extension that would allow the techniques proposed in this thesis to better integrate with current production workflows is the ability to precompute on a short animated sequence. Digital artists today work often with a turntable animation where either the lighting or camera make a 360° rotation around a scene, while attributes are adjusted. The capability to precompute on a similar preset animation would no-doubt require the use of some compression, since storing many frames with the footprints described in previous sections would not be practical. In addition, it is more natural to perform such precomputation and compression on vertices instead of pixels, as is currently the case.

Extending the pixel-based precomputation methods described in this thesis to vertex-based would provide several advantages. (The precomputations in Sections 4.2, 6.1.1, and Chapter 8 are all pixel-based) While these methods assume a fixed viewpoint, and therefore would not allow scene walkthroughs even with a vertex-based extension, other operations can be enabled by computing directly on the scene's geometry. Besides the aforementioned per-vertex compression, changes in image resolution, camera zoom and field-of-view, as well as camera rotation (without change in position) would be possible with vertex-bound precomputed data.

11.3 Advantages of Precomputation-based Rendering

The ‘holy grail’ of real-time rendering is to be able to edit any aspect of a scene and rerender the image in less than 30ms. GPUs and programmable shaders approach this problem by increasing the speed and versatility of the scanline-shade pipeline for rendering. However complex lighting and global illumination are not part of this pipeline. In addition hardware-based rendering speed is proportional to the complexity of the input data. Modern scenes with millions of polygons, hundreds of lights, and hundreds of shaders continue to require more computation power than the industry can supply. In contrast, precomputation-based rendering (that fixes some aspect of the scene) can be decoupled from the complexity of many scene components. For example, the work presented in this thesis renders at the same speed regardless of geometric and lighting complexity of the scene. Similarly, relighting is decoupled from the material complexity, and interactive camera motion techniques (radiosity and (surface) light-fields) are decoupled from the material and lighting complexity.

A more restricted problem is the ‘holy-grail’ of interactive design. This problem recognizes that scenes do not change arbitrarily, but must be able to change according to the will of a human designer. With the introduction of BRDF editing in this work, there is an opportunity now to combine the three design scenarios (relighting, camera motion, BRDF editing) into an iterative system that fixes two, while allowing interactive design of the third. The ‘holy grail’ will not be achieved with a brute-force approach, but by partitioning the problem into manageable subproblems. A worthwhile direction of future work is to be able to quickly convert between the representations needed for each type of editing operation, to unite them into an interactive rendering experience that gives the user control over any of: lighting, materials, or camera, but not necessarily simultaneously.

REFERENCES

- Agarwal, Sameer, Ramamoorthi, Ravi, Belongie, Serge, & Jensen, Henrik Wann. 2003. Structured importance sampling of environment maps. *Acm transactions on graphics*, **22**(3), 605–612.
- Agrawala, Maneesh, Ramamoorthi, Ravi, Heirich, Alan, & Moll, Laurent. 2000 (July). Efficient image-based methods for rendering soft shadows. *Pages 375–384 of: Proceedings of acm siggraph 2000*. Computer Graphics Proceedings, Annual Conference Series.
- Arvo, James, Torrance, Kenneth, & Smits, Brian. 1994. A framework for the analysis of error in global illumination algorithms. *Computer graphics*, **28**(Annual Conference Series), 75–84.
- Ashikhmin, Michael, & Shirley, Peter. 2000. An anisotropic phong brdf model. *J. graph. tools*, **5**(2), 25–32.
- Ashikhmin, Michael, Premoze, Simon, & Shirley, Peter. 2000. A microfacet-based BRDF generator. *Pages 65–74 of: Siggraph 00*.
- Bala, Kavita, Walter, Bruce J., & Greenberg, Donald P. 2003. Combining edges and points for interactive high-quality rendering. *Acm transactions on graphics*, **22**(3), 631–640.
- Barbic, James, & James, Doug. 2005. Real-time subspace integration of St. Venant-Kirchoff deformable models. *Acm transactions on graphics (siggraph 2005)*, **24**(3), 982–990.
- Ben-Artzi, Aner, Ramamoorthi, Ravi, & Agrawala, Maneesh. 2006a. Efficient shadows from sampled environment maps. *journal of graphics tools*, **11**(1), 13–36.
- Ben-Artzi, Aner, Overbeck, Ryan, & Ramamoorthi, Ravi. 2006b. Real-time BRDF editing in complex lighting. *Acm transactions on graphics (siggraph 06)*, **25**(3), 945–954.
- Blinn, James F. 1977. Models of light reflection for computer synthesized pictures. *Pages 192–198 of: Siggraph '77: Proceedings of the 4th annual conference on computer graphics and interactive techniques*. New York, NY, USA: ACM Press.

- Blinn, James F., & Newell, Martin E. 1976. Texture and reflection in computer generated images. *Communications of the acm*, **19**(10), 542–547.
- Chen, Wei-Chao, Bouguet, Jean-Yves, Chu, Michael H., & Grzeszczuk, Radek. 2002. Light field mapping: efficient representation and hardware rendering of surface light fields. *Acm trans. graph.*, **21**(3), 447–456.
- Cohen, Michael F., & Wallace, John R. 1993. *Radiosity and realistic image synthesis*. Academic Press.
- Colbert, Mark, Pattanaik, Sumanta, & Krivanek, Jaroslav. 2006. Brdf-shop: Creating physically correct bidirectional reflectance distribution functions. *Ieee comput. graph. appl.*, **26**(1), 30–36.
- Cook, Robert, & Torrance, Kenneth. 1982. A reflectance model for computer graphics. *Acm transactions on graphics*, **1**(1), 7–24.
- Dana, Kristin J., van Ginneken, Bram, Nayar, Shree K., & Koenderink, Jan J. 1999. Reflectance and texture of real-world surfaces. *Acm transactions on graphics*, **18**(1), 1–34.
- Daubechies, Ingrid. 1988. Orthonormal bases of compactly cupported wavelets. *Communications of pure and applied math*, **41**(0).
- Debevec, Paul. 1998 (July). Rendering synthetic objects into real scenes. *Pages 189–198 of: Proceedings of siggraph 98*. Computer Graphics Proceedings, Annual Conference Series.
- Dobashi, Yoshinori, Kaneda, Kazufumi, Nakatani, Hideki, & Yamashita, Hideo. 1995. A quick rendering method using basis functions for interactive lighting design. *Computer graphics forum*, **14**(3), 229–240.
- Dorsey, Julie, Arvo, James, & Greenberg, Donald. 1995. Interactive design of complex time dependent lighting. *Ieee computer graphics and applications*, **15**(2), 26–36.
- Durand, Frédo, Holzschuch, Nicolas, Soler, Cyril, Chan, Eric, & Sillion, François X. 2005. A frequency analysis of light transport. *Acm transactions on graphics (siggraph 05)*, **24**(3), 1115–1126.
- Ershov, Sergey, Kolchin, Konstantin, & Myszkowski, Karol. 2001. Rendering pearlescent appearance based on paint-composition modelling. *Pages 227–238 of: Chalmers, A., & Rhyne, T.-M. (eds), EG 2001 proceedings*, vol. 20(3). Blackwell Publishing.
- Greene, Ned. 1986. Environment mapping and other applications of world projections. *Ieee computer graphics & applications*, **6**(11), 21–29.

- Guo, Baining. 1998 (July). Progressive radiance evaluation using directional coherence maps. *Pages 255–266 of: Proceedings of siggraph 98*. Computer Graphics Proceedings, Annual Conference Series.
- Hapke, Bruce W. 1963. A theoretical photometric function of the lunar surface. *Journal of geophysical research* 68, **15**, 4571–4586.
- Hart, David, Dutré, Philip, & Greenberg, Donald P. 1999 (Aug.). Direct illumination with lazy visibility evaluation. *Pages 147–154 of: Proceedings of siggraph 99*. Computer Graphics Proceedings, Annual Conference Series.
- Hasan, Milos;, Pellacini, Fabio, & Bala, Kavita. 2006. Direct-to-indirect transfer for cinematic relighting. 1089–1097.
- Heidrich, Wolfgang, & Seidel, Hans-Peter. 1998 (Aug.). View-independent environment maps. *Pages 39–46 of: 1998 siggraph / eurographics workshop on graphics hardware*.
- Kajiya, James T. 1986. The rendering equation. *Pages 143–150 of: Siggraph '86: Proceedings of the 13th annual conference on computer graphics and interactive techniques*. New York, NY, USA: ACM Press.
- Kautz, Jan, & McCool, Michael. 1999. Interactive rendering with arbitrary BRDFs using separable approximations. *Pages 247–260 of: Eurographics rendering workshop 99*.
- Kautz, Jan, Vázquez, Pere-Pau, Heidrich, Wolfgang, & Seidel, Hans-Peter. 2000. Unified approach to prefiltered environment maps. *Pages 185–196 of: Proceedings of the eurographics workshop on rendering techniques 2000*. London, UK: Springer-Verlag.
- Kollig, Thomas, & Keller, Alexander. 2003 (June). Efficient illumination by high dynamic range images. *Pages 45–51 of: Eurographics symposium on rendering: 14th eurographics workshop on rendering*.
- Kontkanen, Janne, Turquin, Emmanuel, Holzschuch, Nicolas, & Sillion, Francois. 2006. Wavelet radiance transport for interactive indirect lighting. *In: Eurographics symposium on rendering*.
- Lafortune, Eric P. F., Foo, Sing-Choong, Torrance, Kenneth E., & Greenberg, Donald P. 1997. Non-linear approximation of reflectance functions. *Pages 117–126 of: Siggraph '97: Proceedings of the 24th annual conference on computer graphics and interactive techniques*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co.

- Lawrence, Jason, Rusinkiewicz, Szymon, & Ramamoorthi, Ravi. 2004. Efficient BRDF importance sampling using a factored representation. *Acm transactions on graphics (siggraph 2004)*, **23**(3).
- Lawrence, Jason, Ben-Artzi, Aner, DeCoro, Christopher, Matusik, Wojciech, Pfister, Hanspeter, Ramamoorthi, Ravi, & Rusinkiewicz, Szymon. 2006. Inverse shade trees for non-parametric material representation and editing. 735–745.
- Lensch, Hendrik P. A., Kautz, Jan, Goesele, Michael, Heidrich, Wolfgang, & Seidel, Hans-Peter. 2001 (June). Image-based reconstruction of spatially varying materials. *Pages 103–114 of: Rendering techniques 2001: 12th eurographics workshop on rendering*.
- Leung, Thomas, & Malik, Jitendra. 2001. Representing and recognizing the visual appearance of materials using three-dimensional textons. *Int. j. comput. vision*, **43**(1), 29–44.
- Liu, Xinguo, Sloan, Peter-Pike, Shum, Heung-Yeung, & Snyder, John. 2004. All-frequency precomputed radiance transfer for glossy objects. *Pages 337–344 of: Eurographics symposium on rendering*.
- Matusik, Wojciech, Pfister, Hanspeter, Brand, Matt, & McMillan, Leonard. 2003. A data-driven reflectance model. *Acm trans. graph.*, **22**(3), 759–769.
- McCool, Michael D., Ang, Jason, & Ahmad, Anis. 2001. Homomorphic factorization of brdfs for high-performance rendering. *Pages 171–178 of: Siggraph '01: Proceedings of the 28th annual conference on computer graphics and interactive techniques*. New York, NY, USA: ACM Press.
- Miller, Gene S., & Hoffman, C. Robert. 1984 (July). Illumination and reflection maps: Simulated objects in simulated and real environments. *In: Course notes for advanced computer graphics animation in siggraph 84*.
- Minnaert, Marcel. 1941. The reciprocity principle in lunar photometry. *Astrophysical journal*, **93**, 403–401.
- Nayar, Shree K., Krishnan, Gurunandan, Grossberg, Michael D., & Raskar, Ramesh. 2006. Fast separation of direct and global components of a scene using high frequency illumination. *Acm trans. graph.*, **25**(3), 935–944.
- Ng, Ren, Ramamoorthi, Ravi, & Hanrahan, Pat. 2003. All-frequency shadows using non-linear wavelet lighting approximation. *Acm transactions on graphics*, **22**(3), 376–381.
- Ng, Ren, Ramamoorthi, Ravi, & Hanrahan, Pat. 2004. Triple product wavelet integrals for all-frequency relighting. *Acm transactions on graphics (siggraph 2004)*, **23**(3), 475–485.

- Nicodemus, F. E., Richmond, J. C., Hsia, J. J., Ginsberg, I. W., & Limperis, T. 1977. *Geometric considerations and nomenclature for reflectance*. National Bureau of Standards (US).
- Nimeroff, Jeffry S., Simoncelli, Eero, & Dorsey, Julie. 1994. Efficient Re-rendering of Naturally Illuminated Environments. *Pages 359–373 of: Fifth eurographics workshop on rendering*. Darmstadt, Germany: Springer-Verlag.
- Oren, Michael, & Nayar, Shree. 1994. Generalization of lambert’s reflectance model. *Pages 239–246 of: Siggraph 94*.
- Ostromoukhov, Victor, Donohue, Charles, & Jodoin, Pierre-Marc. 2004. Fast hierarchical importance sampling with blue noise properties. *Acm transactions on graphics*, **23**(3), 488–495.
- Overbeck, Ryan, Ben-Artzi, Aner, Ramamoorthi, Ravi, & Grinspun, Eitan. 2006. Exploiting Temporal Coherence for Incremental All-Frequency Relighting. *In: Eurographics symposium on rendering*.
- Pharr, Matt, & Humphreys, Greg. 2004. *Physically based rendering: From theory to implementation*. Morgan Kaufmann.
- Phong, Bui Tuong. 1975. Illumination for computer generated pictures. *Commun. acm*, **18**(6), 311–317.
- Ramamoorthi, Ravi, & Hanrahan, Pat. 2002. Frequency space environment map rendering. *Acm transactions on graphics (siggraph 02 proceedings)*, **21**(3), 517–526.
- Rusinkiewicz, Szymon. 1998a. bv: brdf viewer. *In: graphics.stanford.edu/smr/brdf/bv*.
- Rusinkiewicz, Szymon. 1998b. A new change of variables for efficient BRDF representation. *Pages 11–22 of: Eurographics rendering workshop 98*.
- Schlick, Christophe. 1994. An inexpensive BRDF model for physically-based rendering. *Computer graphics forum*, **13**(3), 233–246.
- Séquin, Carlo H., & Smyrl, Eliot K. 1989. Parameterized ray-tracing. *Pages 307–314 of: Siggraph ’89: Proceedings of the 16th annual conference on computer graphics and interactive techniques*. New York, NY, USA: ACM Press.
- Sloan, Peter-Pike, Kautz, Jan, & Snyder, John. 2002. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. *Pages 527–536 of: Siggraph ’02: Proceedings of the 29th annual conference on computer graphics and interactive techniques*. New York, NY, USA: ACM Press.

- Sloan, Peter-Pike, Hall, Jesse, Hart, John, & Snyder, John. 2003. Clustered principal components for precomputed radiance transfer. *Acm trans. graph.*, **22**(3), 382–391.
- Sloan, Peter-Pike, Luna, Ben, & Snyder, John. 2005. Local, deformable precomputed radiance transfer. *Acm trans. graph.*, **24**(3), 1216–1224.
- Sun, Weifeng, & Mukherjee, Amar. 2006. Generalized wavelet product integral for rendering dynamic glossy objects. 955–966.
- Torrance, Kenneth, & Sparrow, Ephraim. 1967. Theory for off-specular reflection from roughened surfaces. *JOSA*, **57**(9), 1105–1114.
- Vasilescu, M. Alex O., & Terzopoulos, Demetri. 2004. *Tensortextures: multilinear image-based rendering*.
- Wan, Liang, Wong, Tien-Tsin, & Leung, Chi-Sing. 2005. Spherical q2-tree for sampling dynamic environment sequences. *Pages 21–30 of: Rendering techniques*.
- Wang, Rui, Tran, John, & Luebke, David. 2004. All-frequency relighting of non-diffuse objects using separable BRDF approximation. *Pages 345–354 of: Eurographics symposium on rendering*.
- Ward, Greg. 1992. Measuring and modeling anisotropic reflection. *Pages 265–272 of: Siggraph 1992*.
- Ward, Gregory J., Rubinstein, Francis M., & Clear, Robert D. 1988. A ray tracing solution for diffuse interreflection. *Pages 85–92 of: Siggraph '88: Proceedings of the 15th annual conference on computer graphics and interactive techniques*. New York, NY, USA: ACM Press.
- Whitted, Turner. 1980. An improved illumination model for shaded display. *Communications of the acm*, **23**(6), 343–349.
- Zhou, Kun, Hu, Yaohua, Lin, Stephen, Guo, Baining, & Shum, Heung-Yeung. 2005. Precomputed shadow fields for dynamic scenes. 1196–1201.
