ARTISTIC PATH SPACE EDITING OF PHYSICALLY BASED LIGHT TRANSPORT

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

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

genehmigte

Dissertation

von

Thorsten-Walther Schmidt

aus Bad Friedrichshall

Tag der mündlichen Prüfung:	26. Juli 2017
Erster Gutachter:	Prof. Dr.-Ing. Carsten Dachsbacher
Zweiter Gutachter:	Jun. Prof. Dr.-Ing. Boris Neubert

Abstract

Creating realistic images is an important goal of computer graphics, with applications, among others, in the feature film, architecture, and medical industries. Physically based rendering, which has recently seen wide adoption across fields, refers to the accurate numerical simulation of light transport along the paths prescribed by the model of geometric optics, which is sufficient to achieve photorealism for typical scenes in the aforementioned cases.

Overall, the computer-based authoring of images and animations with well-designed and theoretically sound shading is vastly simplified today. However, taking into account details such as the structure of the output device is important for practical implementations, and, for example, the subdomain of scalable physically based rendering of participating media is far from being a solved problem.

Moreover, image synthesis is only one part of a larger process: the effective communication of ideas and information between people. Be it the shape and function of a building, the medical visualization of a computed tomography scan, or the mood of a movie sequence: messages in the form of reconstructed images are ubiquitous in today's world. Unfortunately, adoption of the simulation-centered methodology of physically based rendering has also led to a loss of intuitive, fine-grained, and local artistic control over the final image that was present in earlier, less rigorous paradigms.

The contributions of this dissertation cover several aspects of image synthesis, spanning the range from high-quality rendering of finely detailed geometry at the subpixel level, to efficient physically based rendering algorithms for participating media. The main focus of this work, however, are approaches that enable effective visual understanding and artistic manipulation of light transport, while maintaining globally consistent, plausible results. The central idea is that visualization, selection, and editing operations should be performed directly in the space encompassing all possible light paths, as opposed to state-of-the-art methods which either work in image space or are tailored to specific, isolated lighting effects, such as mirror reflections, shadows, or caustics. Testing of the proposed methods has shown them to be effective in solving real-world rendering problems.

Kurzfassung

Die Erzeugung realistischer Bilder ist ein wichtiges Ziel der Computergrafik, mit Anwendungen u.a. in der Spielfilmindustrie, Architektur und Medizin. Die physikalisch basierte Bildsynthese, welche in letzter Zeit anwendungsübergreifend weiten Anklang findet, bedient sich der numerischen Simulation des Lichttransports entlang durch die geometrische Optik vorgegebener Ausbreitungspfade; ein Modell, welches für übliche Szenen ausreicht, Photorealismus zu erzielen.

Insgesamt gesehen ist heute das computergestützte Verfassen von Bildern und Animationen mit wohlgestalteter und theoretisch fundierter Schattierung stark vereinfacht. Allerdings ist bei der praktischen Umsetzung auch die Rücksichtnahme auf Details wie die Struktur des Ausgabegeräts wichtig und z.B. das Teilproblem der effizienten physikalisch basierten Bildsynthese in partizipierenden Medien ist noch weit davon entfernt, als gelöst zu gelten.

Weiterhin ist die Bildsynthese als Teil eines weiteren Kontextes zu sehen: der effektiven Kommunikation von Ideen und Informationen. Seien es nun Form und Funktion eines Gebäudes, die medizinische Visualisierung einer Computertomografie oder aber die Stimmung einer Filmsequenz – Botschaften in Form digitaler Bilder sind heutzutage omnipräsent. Leider hat die Verbreitung der – auf Simulation ausgelegten – Methodik der physikalisch basierten Bildsynthese generell zu einem Verlust intuitiver, feingestalteter und lokaler künstlerischer Kontrolle des finalen Bildinhalts geführt, welche in vorherigen, weniger strikten Paradigmen vorhanden war.

Die Beiträge dieser Dissertation decken unterschiedliche Aspekte der Bildsynthese ab. Dies sind zunächst einmal die grundlegende Subpixel-Bildsynthese sowie effiziente Bildsyntheseverfahren für partizipierende Medien. Im Mittelpunkt der Arbeit stehen jedoch Ansätze zum effektiven visuellen Verständnis der Lichtausbreitung, die eine lokale künstlerische Einflussnahme ermöglichen und gleichzeitig auf globaler Ebene konsistente und glaubwürdige Ergebnisse erzielen. Hierbei ist die Kernidee, Visualisierung und Bearbeitung des Lichts direkt im alle möglichen Lichtpfade einschließenden „Pfadraum“ durchzuführen. Dies steht im Gegensatz zu Verfahren nach Stand der Forschung, die entweder im Bildraum arbeiten oder auf bestimmte, isolierte Beleuchtungseffekte wie perfekte Spiegelungen, Schatten oder Kaustiken zugeschnitten sind. Die Erprobung der vorgestellten Verfahren hat gezeigt, dass mit ihnen real existierende Probleme der Bilderzeugung für Filmproduktionen gelöst werden können.

Übersicht

Diese Arbeit setzt sich mit Beiträgen im Bereich der Bildsynthese und -darstellung auseinander, welche die Bandbreite von grundlegender Subpixel-Bildsynthese, über effiziente Bilderzeugungsverfahren für Medien, zu Ansätzen zum effektiven visuellen Verständnis der Lichtausbreitung umspannen, dabei eine lokale gestalterische Einflussnahme ermöglichen und gleichzeitig auf globaler Ebene konsistente und glaubwürdige Ergebnisse erzielen. Das Absicht ist dabei, den Bereich zwischen der „korrekten“ physikalisch basierten Bildsynthese und völliger künstlerischer Freiheit zu ergründen. Erstere erschwert, den Bildentstehungsprozess freier zu gestalten, währenddessen letztere es mühsam macht, glaubwürdige, überzeugende Bilder zu generieren. Unser Beitrag soll einen Mittelweg aufzeigen, der erlaubt, computergeneriertes Bildmaterial zu erschaffen, das sowohl glaubwürdig als auch ausdrucksstark ist; die Begriffe „ausdrucksstark“ und „künstlerisch“ sollten hierbei recht weit aufgefasst werden, da auch im Bereich der Visualisierung, bei der die Vermittlung von Informationen im Vordergrund steht, eine gestalterische Einflussnahme explizit erwünscht ist.

Im Folgenden fassen wir die wichtigsten Aspekte der Arbeit zusammen. Auf oberster Ebene gliedert sich diese in drei Themenbereiche, welche in der Praxis eng verzahnt sind: (I) Bildsynthese, (II) Lichttransport, (III) Visualisierung und künstlerische Bearbeitung. Die Struktur folgt dabei in jedem Themenbereich dem Schema, zunächst vorhergehende Arbeiten zusammenzufassen, um danach eigene Beiträge vorzustellen. Auf der nächsten Ebene ist die Dissertation wie folgt in Kapitel unterteilt:

In Kapitel 1 setzten wir uns kurz mit dem Spannungsfeld zwischen der künstlerischen Gestaltung von Bildern und deren mechanistischer Erzeugung am Computer auseinander. Obgleich die künstlerische Gestaltung eine lange Tradition hat, sind neuartige Methoden zur rechnergestützten Bilderzeugung hauptsächlich auf die akkurate Simulation des Lichttransports ausgelegt.[1]

In Kapitel 2 befassen wir uns zunächst mit mathematischen Grundlagen, die in der Bildsynthese häufig Anwendung finden. Im einzelnen besprechen wir kurz Maße (im Sinne der Maßtheorie), Dirac-Verteilungen, einige Grundlagen der Wahrscheinlichkeitsrechnung und danach etwas ausführlicher die Monte-Carlo-Integration, welche ein wichtiger Bestandteil der heute verbreiteten physikalisch basierten Bildsyntheseverfahren ist. Schließlich behandeln wir kurz die Fourieranalyse von Signalen und die Integration von Funktionen, welche auf der Einheitskugel $\mathcal{S}^2$ oder Halbkugel $\mathcal{H}^2$ definiert sind.

In Kapitel 3 thematisieren wir die Grundlagen der Bildsynthese, angefangen damit, was überhaupt im Sinne der Computergrafik ein Bild ist und wie Farben

[1] Diese Kapitel enthält ebenfalls eine Liste aller Publikationen, zu denen ich beigetragen habe.

repräsentiert werden. Danach gehen wir auf die Abtastung und Rekonstruktion von Bildern ein, besprechen, welche Fehler durch Verletzung des Abtasttheorems auftreten können, und wie man diese vermeiden oder minimieren kann. Danach beschäftigen wir uns kurz mit Ausgabegeräten, und ordnen schließlich die Möglichkeiten der realistischen Echtzeitbildsynthese ein.

Um zur Darstellungsverbesserung den Eindruck einer höheren Auflösung zu erreichen, als das Ausgabegerät eigentlich aufweist, kann man seine Subpixel-Struktur ausnutzen. In Kapitel 4 stellen wir eine generelle Technik zur optimalen Subpixel-Bildsynthese vor.[2] Nach einer kurzen Einführung in das Thema gehen wir auf den Ansatz von Platt und Kollegen ein, und zeigen, wie dieser für beliebige ein- und zweidimensionale Subpixel-Muster erweitert werden kann, um optimale Bildfilter zu erhalten. Basierend auf einer Fourier-Analyse der Filter führen wir analytische Filter ein, um sie dann schließlich für die Echtzeit-Kantenglättung und die Texturfilterung einzusetzen. Zu guter Letzt gehen wir auf eine Benutzerstudie ein, in welcher wir die verbesserte Bildqualität unserer Filter verifizierten.

Kapitel 5 eröffnet mit der physikalisch basierten Bildsynthese den zweiten großen Themenblock dieser Arbeit. Neben Grundlagen der Radiometrie, der Modellierung von Lichtquellen, Oberflächengeometrie und Reflexionsverhalten der Szene, besprechen wir mit der Lichttransport- und Messgleichung die Grundbausteine eines physikalisch basierten Bildsyntheseprogrammes. Die Einführung einer weiteren Transportgröße neben dem Licht führt schließlich zur Pfadintegralformulierung des Lichttransports nach Veach, welche ein äußerst nützliches theoretisches Werkzeug zur Beschreibung von Bildsyntheseverfahren ist, und uns das Konzept des *Pfadraums* an die Hand gibt. Im Anschluss führt uns die Strahlungstransportgleichung zu einem generalisierten Modell des Lichttransports, welches auch die Lichtstreuung in partizipierenden Medien, wie etwa der menschlichen Haut oder Wasserdampf, berücksichtigt, aber aufgrund ihres Simulationsaufwands in der Regel vereinfacht werden muss. Schließlich kommen wir zu einer Kategorisierung physikalisch basierter Bildsyntheseverfahren, deren gemeinsamer Nenner die zufällige Abtastung des Pfadraums ist.

Kapitel 6 geht auf eigene Beiträge im Bereich der interaktiven Bildsynthese in heterogenen, partizipierenden Medien ein.[3] Hier zeigen wir, wie unter den richtigen Annahmen ein skalierbares Verfahren zur Bilderzeugung hergeleitet werden kann, welches sowohl eine interaktive Vorschau als auch qualitativ hochwertige Endresultate liefert, und dabei systematische Fehler vorhergehender Verfahren mit vergleichbaren Zielen minimiert. Interaktive physikalisch basierte Bildsyntheseverfahren bilden die Grundlage für die spätere Visualisierung und Manipulation des Lichttransports.

Der dritte und letzte Themenblock beginnt in Kapitel 7 mit einer ausführlichen Klassifizierung zum Stand der Forschung bezüglich künstlerischer Bearbeitung von Materialmodellen, virtueller Beleuchtung und der *allgemeinen Erscheinung* von Bildern.[4] Nachdem wir feststellen, dass in der physikalisch basierten Bildsynthese eine klare Trennung zwischen Material und Licht nicht immer möglich ist – da die Erscheinung des Bildes letztlich von beiden abhängt –, vertreten wir die Perspektive, dass die Erscheinung implizit im Lichtfeld verborgen ist, wovon das Bild nur einen Ausschnitt zeigt. Mit diesem Hintergedanken besprechen wir dann vorhergehende

[2]Diese Kapitel beruht auf einer vorhergehenden Publikation [47].
[3]Basierend auf der Publikation [46].
[4]Dieses Kapitel basiert auf den Veröffentlichungen [201, 202].

Arbeiten im Bereich der Beleuchtungs- und Materialgestaltung (im Sinne der Computergrafik), und gehen auf die verbreitete Methodik zur Evaluierung verschiedener Benutzerschnittstellen ein.

In Kapitel 8 gehen wir auf die Lichttransportvisualisierung ein. Aus Anwendersicht ist bei der physikalisch basierten Bildsynthese ein kritischer Aspekt, den komplexen simulierten Lichttransport zu Verstehen, denn nur aufgrund eines tieferen Verständnisses können effektiv Änderungen durchgeführt werden. Nach einer Besprechung vorhergehender Ansätze – bei einem Verfahren im Detail – erörtern wir zunächst Datenstrukturen zur Pfadrepräsentation, bevor wir schließlich unseren eigenen Beitrag [200] vorstellen: Ein direktes Visualisierungsverfahren, welches Lichtpfade analysiert, gruppiert, und, inspiriert durch eine Technik aus der Informationsvisualiserung, als Bündel darstellt.

Dieses Visualisierungsverfahren ist eng verzahnt mit unserem Hauptbeitrag in Kapitel 9: Ein interaktives System, mit dem die Beleuchtung einer Szene visualisiert, ausgewählt, und bearbeitet werden kann.[5] Die Kernidee ist, Visualisierung und Bearbeitung des Lichts direkt im alle möglichen Lichtpfade einschließenden Pfadraum durchzuführen. Dies steht im Gegensatz zu Verfahren nach Stand der Forschung, die entweder im Bildraum arbeiten oder auf spezielle, isolierte Beleuchtungseffekte wie perfekte Spiegelungen, Schatten oder Kaustiken zugeschnitten sind. Unsere gewählte Repräsentation des Lichtes über Pfade ist sehr allgemein, und dabei prinzipiell unabhängig vom eingesetzten (physikalisch basierten) Bildsyntheseverfahren.

Neben der Visualisierung des Lichttransports führen wir eine Analyse des Lichtfeldes durch, welche dem Benutzer, basierend auf groben Skizzen oder der direkten Platzierung von Auswahlbereichen in der Szene, gepaart mit einem Clustering und einer Sortierung von Lichtpfaden, semiautomatische Mechanismen zum Auffinden interessanter Lichttransportphänomene an die Hand gibt.

Schließlich ist die *künstlerische Beeinflussung des physikalisch basierten Lichttransports im Pfadraum* ein entscheidender Beitrag unserer Arbeit. Als konkrete Umsetzung dieser Idee stellen wir zwei Verfahren namens *Path Retargeting* und *Path–Proxy Linking* vor, welche erlauben, Teile des Lichttransports in einer Szene zu verändern, dabei aber glaubwürdige – da global konsistente – Endergebnisse erzielen. Unsere beiden Verfahren schließen vorhergehende Arbeiten nach Stand der Forschung, wie zum Beispiel die Manipulation von Spiegelreflexionen oder die nicht geradlinigen Ausbreitung direkten Lichts, als Sonderfälle mit ein. Unser System integriert sich in eine in der Industrie weit verbreitete Software, und zuletzt gehen wir auf eine praktische Erprobung in Zusammenarbeit mit der Filmakademie Baden-Württemberg ein.[6]

Kapitel 10 bildet den Abschluss dieser Dissertation, ordnet die Arbeit noch einmal im Gesamtkontext ein, bespricht dabei Beschränkungen unseres Ansatzes und zeigt Richtungen für weiterführende Arbeiten auf.

[5] Dieses Kapitel basiert ebenfalls auf [200].

[6] Im weiteren Verlauf seit der ursprünglichen Publikation ist das Verfahren heute nun auch Gegenstand einer KIT-Ausgründung in der Filmbranche.

Erklärung

Ich versichere wahrheitsgemäß, diese Dissertation bis auf die angegebenen Hilfen selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben sowie kenntlich gemacht zu haben, was aus Arbeiten anderer und eigenen Veröffentlichungen unverändert oder mit Änderungen entnommen wurde.

Karlsruhe, den 3. April 2018

(Thorsten-Walther Schmidt)

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Figure 1.1: Supposed image of *crocuta crocuta spelaea*, an extinct prehistoric subspecies of spotted hyena, found in Lascaux Cave, Dordogne, France. Drawing *after drawing after* rock painting [215].

While mimicking the appearance of the real world is a longstanding goal of computer graphics, the idea to artistically interpret and depict our perception of reality is much older (see, e.g., Fig. 1.1). Even when taking into account the uncertainty of exact dating, the occupation of humans with creating images clearly predates written history.[1]

The development of arts and crafts over the millennia has lead to our current understanding of "art," a term used with two different meanings: there is art for art's sake—what may be called high art, fine art, academic art, true art, etc.—and commercial art. Arguably, computer graphics is mostly used for the latter, if only for its potential to more efficiently serve the seemingly ever-increasing demand for high-quality pictures in entertainment, advertising, virtual engineering, visualization, and elsewhere.

[1]Incidentally, see McCloud [141] for an interesting discussion on the tension between images and written words—ironically itself expressed in the comics medium.

Regarding the development of the field, if art was only about depicting objective reality, then it should have disappeared with the invention and widespread availability of photography. Instead, around the *fin de siècle* art split into a multitude of different movements [240]. This development already started in the 19th century, at around the same time that photography became practical.

Sidestepping all philosophical aspects of image creation,[2] from a technical perspective, artworks need a medium in which they are expressed. The conception of computers—and display devices—in the 20th century introduced yet another viable medium for creating imagery.

1.1 Computer Graphics

Computer graphics (CG), in essence, is using computing machinery to assist in the process of creating images.[3] Though firmly rooted in applied computer science, it is an interdisciplinary subject, overlapping fields such as mathematics, physics, the psychology of perception, design and art, and others.

Substantial research in the past 40-plus years has developed methods to reproduce reality—or rather, artistic idealizations thereof—accurately using numerical methods and computers. One could argue that this has had a similarly disruptive effect on visual art as the invention of photography at the turn of the 20th century.

Sketchpad, Ivan Sutherland's thesis work from 1963 [224], is often considered to be the starting point of modern computer graphics (and graphical user interfaces). Early methods focused on vector displays, deflecting an electron beam hitting a phosphorescent screen that—by means of the afterglow of said screen and the persistence of vision of the human visual system—leaves behind a trail which is interpreted as a line.

The introduction of *raster graphics* led to the conception of images as (usually rectangular) grids of picture elements—whence the now-familiar term "pixel." The principle of moving an electron beam still applied, but instead of moving it across the screen in arbitrary fashion, it cycled across fixed patterns on the screen, giving rise to concepts such as refresh rate, vertical sync, etc. Color displays introduced separately controlled, color-filtered *subpixels*, which at high enough resolutions the human visual system fuses into a continuous picture [59]. Modern display technology still follows the same idea.

Early on, computer graphics has been used commercially, and the process of *rendering* images with a computer is now a commonplace phenomenon. Apart from solving the visibility problem, i.e., determining what object can be seen "through" a single pixel, an important issue for graphics research on rendering has been that of *shading*, whereby the color value of visible pixels is computed.

1.2 Physically Based Rendering

One popular approach to shading is *realistic* rendering. Thanks to extensive research in the past, the availability of efficient methods for realistic image synthesis has

[2]Starting with the basic question, "What is an image of something?" we note that the animal in Fig. 1.1 has been imperfectly reproduced and re-reproduced *at least five times* as you are reading this, thousands of years after its original lifespan. Given that it was not the product of somebody's imagination.

[3]As opposed to using "non-computing machinery," say, a printing press, a pen, or some other simple device.

become the standard across industries. The entertainment industry has been a large driving factor behind both rendering research and the development of specialized graphics hardware, a trend from which other areas such as lighting design, product prototyping, architectural visualization, and mechanical engineering have benefited as well.

Indeed, even scientific visualization, which has traditionally been using so-called non-photorealistic rendering, is importing ideas from this domain, e.g., ambient occlusion for molecular visualization [226] or full global illumination for medical visualization of scalar fields, such as CT scans [127].

In movie and games production, realistic image synthesis is now usually marketed under the terms *physically based rendering* (PBR) and *physically based shading* (PBS), respectively. Noting that "physically based" almost always only means "according to geometric optics," we will use these terms nonetheless. On the topic of PBR, and specifically, path tracing in movies, see a recent survey by Christensen and Jarosz [22].

If *Sketchpad* has been a cornerstone for CG, then the dissertation of Eric Veach [229] has been similarly influential in the narrower field of PBR, laying much of the groundwork for what is current rendering research.

1.3 Artistic Freedom

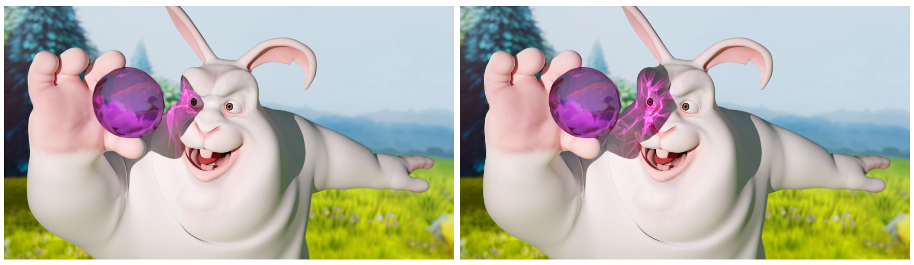

Figure 1.2: Artistic Editing of Physically Based Rendering, as applied to a cartoon character from an animated short movie. See Fig. 9.9 for detail. Image taken from [200].[4]

With the introduction of PBR came the question: how do you create the inputs that give rise to the final image? At first glance, the physically based *production pipeline* is quite simple: You model a scene according to its desired physical properties—geometry, materials, lighting and camera—and then you run a simulation which, within the constraints of its physical and mathematical model, is true to reality. While *modeling* a scene is a large problem domain in itself, *computing images* this way is in a sense very simple and accurate.

It leaves much to be desired from an artistic perspective, though. Indeed, long before the advent of computer graphics, exercising active artistic control over the image formation process has already been commonplace. Take (portrait) photography as an example, where using off-frame reflectors to create indirect, bounced lighting is standard. Even much earlier, realist fine art painters commonly used other "tricks" like increasing the contrast of edges to separate figure from ground.

[4]*Big Buck Bunny* by the Blender Foundation is licensed under CC BY 3.0.

Along those lines, we note that, e.g., reflections do not have to be perfectly accurate for an image to be still perceived as realistic. The famous *Arnolfini Portrait* (1434) by Flemish Renaissance painter Jan van Eyck, showing intricate realistic caustics and curved mirror reflections in its details, was by later analysis confirmed to have physically inaccurate reflections [31]. One step further, the *Rokeby Venus* (ca. 1650) by Diego Velázquez features a mirror reflection *deliberately altered* to depict the subject's face (cf. [193]).

Similarly, since the early beginnings of computer graphics in image production, many tricks and techniques have been developed to guide lighting and shading in order to realize an artistic vision. This, if asking a movie director, is after all the purpose of image creation.[5]

Unfortunately, the shift to physically based rendering has made many of these tricks and techniques largely obsolete.

1.4 Problem Statement

The goal of this dissertation is to explore the area between rigorous physically based rendering and completely creative artistic freedom. The former makes it hard to *art-direct* the image formation process, whereas the latter makes it very tedious to create *plausible, convincing* images.[6] The hope is to arrive at a fertile middle ground, giving artists the ability to author computer-generated imagery that is both believable and expressive at the same time.

1.5 Organization of this Thesis

On a high level, this dissertation is divided into three topical parts which are intimately coupled in practice: (I) image synthesis, (II) light transport, and (III) visualization and artistic editing. The treatment of each part follows the structure of first reciting previous work and then introducing original contributions.

The following list gives a chapter-by-chapter overview of the contents of this thesis:

- Ch. 1 gave the context and problem setting for our work, hinting at the need for effective methods for artistic control of physically based rendering.
- Ch. 2 outlines the mathematical notation and concepts used throughout the rest of this work.
- Ch. 3 covers some of the basics of image synthesis in computer graphics, which is a large sub-field in itself.
- Ch. 4 discusses (optimal) *subpixel* rendering, which takes into account the pixel structure of display devices to attain higher perceived resolution. This chapter is based on [47].[7]
- Ch. 5 focuses on physically based rendering (PBR), the means by which one generates photorealistic images.

[5]Art-directability is important in other sub-fields of CG, e.g., dynamics simulation for animation [68].

[6]As a final nod to philosophy before getting into rather technical matters: this may be viewed as yet another instance of the Apollonian vs. Dionysian dichotomy.

[7]With this publication, my involvement has been mostly on the evaluation—including user study—and presentation aspects of the work.

- Ch. 6 summarizes a novel scalable many-lights rendering method for heterogeneous participating media, which is one area where PBR is (still) difficult. This chapter is based on [46].[8]
- Ch. 7 gives an extensive overview of previous approaches to the artistic editing of appearance, material and lighting. This chapter is based on [201, 202].
- Ch. 8 discusses existing and novel approaches to the visualization of light transport, as one can only effectively manipulate what is (visually) understood. This chapter is partially based on [189, 200].[9]
- Ch. 9 details our novel framework and system for artistic path space editing of physically based light transport, which we deem a viable, very general solution to the problem statement above. This chapter is based on [200].
- Ch. 10 concludes this thesis, discussing limitations of our approach and suggesting possible directions for future work.

1.6 List of Publications

Please note that many of the results presented in this thesis have been previously published as follows:

- [46] Thomas Engelhardt, Jan Novák, Thorsten-Walther Schmidt, and Carsten Dachsbacher. Approximate bias compensation for rendering scenes with heterogeneous participating media. *Computer Graphics Forum (Proceedings of Pacific Graphics)*, 31(7):2145–2154, September 2012.
- [47] Thomas Engelhardt, Thorsten-Walther Schmidt, Jan Kautz, and Carsten Dachsbacher. Low-cost subpixel rendering for diverse displays. *Computer Graphics Forum*, 33(1):199–209, February 2013.
- [200] Thorsten-Walther Schmidt, Jan Novák, Johannes Meng, Anton S. Kaplanyan, Tim Reiner, Derek Nowrouzezahrai, and Carsten Dachsbacher. Path-space manipulation of physically-based light transport. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 32(4):129:1–129:11, August 2013.
- [201] Thorsten-Walther Schmidt, Fabio Pellacini, Derek Nowrouzezahrai, Wojciech Jarosz, and Carsten Dachsbacher. State of the art in artistic editing of appearance, lighting, and material. In *Eurographics 2014 - State of the Art Reports*. Eurographics Association, April 2014.
- [202] Thorsten-Walther Schmidt, Fabio Pellacini, Derek Nowrouzezahrai, Wojciech Jarosz, and Carsten Dachsbacher. State of the art in artistic editing of appearance, lighting, and material – extended version. *Computer Graphics Forum*, 35(1):216–233, February 2016.

[8]With this publication, my involvement has been mostly on the evaluation and presentation side.
[9]I have not been directly involved in the publication of Reiner et al. [189].

Part I

Image Synthesis

Chapter 2

Mathematical Foundations

This chapter briefly reviews some of the mathematical notation and concepts useful for image synthesis in general (Ch. 3), and physically based rendering in particular (Ch. 5).

We will enumerate important notation and results, but keep detailed derivations to the (secondary) literature such as [1, 59, 177, 184, 229]. Where applicable, and as is now common in the advanced rendering literature, our notation in this and following chapters mostly follows Veach [229]. Regarding content, we will assume a general understanding of linear algebra and analysis, and only highlight concepts we deem non-obvious.[1]

2.1 Measures

Broadly speaking, measures make the concept of assigning a numerical quantity to differentials of some integration variable more explicit. As an example, the standard notation for the one-dimensional, real-valued[2] integral,

$$I = \int_a^b f(x)\,\mathrm{d}x, \tag{2.1}$$

is just a shortcut for omitting the "obvious" one-dimensional (Lebesgue) measure of length, i.e., the width of the infinitesimal interval around x. As such, it could have been written explicitly as:

$$I = \int_a^b f(x)\,\mathrm{d}l(x). \tag{2.2}$$

More generally, we will write integrals over an arbitrary domain $\Omega \subseteq \mathbb{R}^N$ as:

$$I = \int_\Omega f(x)\,\mathrm{d}\mu(x), \tag{2.3}$$

[1]That is, from the perspective of a computer scientist, not a mathematician.
[2]Except for the Fourier transform, we will exclusively deal with real-valued integrals.

where $\mu : \Omega \to \mathbb{R}$ is the *measure function*. This notation is important for higher-dimensional integrals, where other measures beyond the standard Lebesgue measure (i.e., length, area, volume, etc.) may be equally useful. It also helps avoid making errors with the application of Dirac distributions.

2.2 Dirac Distributions

Dirac distributions often show up in the mathematical models employed in computer graphics; common examples include pinhole cameras, point and directional lights, and glass and mirror materials.

Dirac distributions are defined such that, when being part of an integrand expressed as a product, they yield the remaining factors evaluated at a single, fixed abscissa x_0, e.g.:

$$\int_\Omega f(x)\,\delta_\mu(x-x_0)\,\mathrm{d}\mu(x) := f(x_0). \tag{2.4}$$

Note the formal dependence of δ_μ on the measure function μ, which reminds us that simply dropping the integral and evaluating $f(x_0)$ *only* works if the differential's measure matches. When switching between, e.g., solid angle and projected solid angle measures in the reflection integrals, it is important to consider this [229].

As they are not proper functions themselves—noting the ":=" in Eq. 2.4—, Dirac distributions may also be thought of as the *limit* of a series of unit area triangle functions of progressively decreasing support, centered around x_0 (cf. Fig. 2.1). Oftentimes, Dirac distributions are visualized as vertical arrows at x_0, with the height denoting the other factors of the integrand at x_0; we will meet Dirac distributions again when discussing sampling and reconstruction (Sec. 3.3).

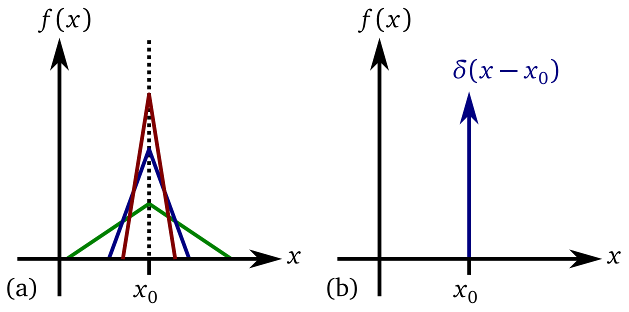

Figure 2.1: Dirac distributions often show up when integrating incident light in physically based rendering. (a) Unit-area triangle functions centered on x_0, whose height approaches infinity as their width approaches zero. (b) Typical graphical notation as an upward-pointing arrow.

2.3 Probability Theory

This section gives a rundown of key concepts from probability theory, to the extent useful for Monte Carlo integration and its application to rendering. As with the other topics in this chapter, see the literature already given for more details.

For a discrete random variable X with K possible outcomes, we can assign probabilities $p_k \in [0,1]$ that a certain outcome x_k may happen. From this, we may compute the *expected value* (also known as expectation or mean):

$$E[X] = \sum_{k=1}^{K} p_k x_k, \tag{2.5}$$

i.e., the average of outcomes x_k over a large set of observations (where we assume that summation over x_k is defined, e.g. $x_k \in \mathbb{R}$).

The same concepts can be mapped to *continuous* random variables, were we have a *probability density function* (PDF) $p(x)$. As we are dealing with an infinite number of outcomes, we can only assign probabilities to (half-open) *intervals*, i.e.:

$$Pr\{X \leq x_1\} = P(x_1) := \int_{-\infty}^{x_1} p(x)\,\mathrm{d}x, \tag{2.6}$$

where $P(x)$ is the *cumulative density function* (CDF). This can be further generalized to arbitrary domains Ω and measures μ:

$$\begin{aligned} Pr\{X \in D \subset \Omega\} &= P(D), && (2.7)\\ p_\mu(x) &= \frac{\mathrm{d}P}{\mathrm{d}\mu}(x). && (2.8)\end{aligned}$$

Noting that functions of random variables are also random variables, the expected value of $Y = f(X)$ (for some random variable X) is computed as:

$$E[Y] = \int_{\Omega} f(x)p(x)\,\mathrm{d}\mu(x), \tag{2.9}$$

which, by the strong law of large numbers [229], conceptually gives us the average (arithmetic mean) value after *sampling* our random variable infinitely often. The *variance* of Y is defined as:

$$V[Y] = E[(Y - E[Y])^2], \tag{2.10}$$

and gives an indication how much Y deviates from the expectation. This becomes more clear when using the standard notation for *computing* these quantities for finite sample sizes in statistics.[3] Specifically, for N samples x_i of a random variable X, the (sample) mean, variance, and standard error are given by:[4]

[3] As used in, say, [51].

[4] In computer graphics, it is customary to refer to each of the x_i as *a sample* and the set of all x_i are simply *the samples*. In statistics, the set of all x_i is usually called *the sample*, whereas a single x_i is an *observation*. [229] The origin of this terminology mismatch is not clear, but may be related to the—slightly—related problem of image sampling and reconstruction.

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i, \tag{2.11}$$

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N} (x_i - \mu)^2, \tag{2.12}$$

$$\sigma = \sqrt{\sigma^2}. \tag{2.13}$$

As can be seen, the mean (Eq. 2.11) is the average of all samples, the variance (Eq. 2.12) is the average squared deviation from the mean, and the standard deviation (Eq. 2.13) is simply the square root of variance, expressed in the same units as X. The last fact proves useful to provide meaningful error estimates of, say, Monte Carlo integration of physical quantities.

2.4 Monte Carlo Integration

Monte Carlo (MC) integration is a method to numerically compute definite integrals using random numbers [59, 177, 184, 229]. As opposed to deterministic quadrature rules, Monte Carlo integration adapts well to higher-dimensional spaces, because its convergence behavior is independent of the integration domain's dimensionality. As we will see in Ch. 5, in physically based rendering, we will typically want to compute definite integrals over an infinite-dimensional space of light paths, so the Monte Carlo method is a good fit.

The basic one-dimensional MC method can be stated thus:

$$I = \int_a^b f(x)\,\mathrm{d}x \approx \frac{b-a}{N}\sum_{i=1}^{N} f(x_i), \tag{2.14}$$

where the samples x_i are taken from the standard uniform distribution $U(a,b)$.

Generalizing to arbitrary integration domains Ω, measures μ, and probability distributions p, we have:

$$I = \int_\Omega f(x)\,\mathrm{d}\mu(x) \approx \frac{1}{N}\sum_{i=1}^{N} \frac{f(x_i)}{p_\mu(x_i)} \tag{2.15}$$

The right-most sum is also called a Monte Carlo *estimator* for the integral I.

2.4.1 Importance Sampling

While the Monte Carlo method allows us to sample the integrand f from an arbitrary distribution p—provided that $f(x) \neq 0 \Rightarrow p(x) \neq 0$—it is beneficial to pick a distribution which is as similar as possible to the integrand. This is called *importance sampling*, as we would like to pick samples from the "important" parts of the integrand, i.e., where the integrand takes on high values, thus capturing the bulk of the sum in Eq. 2.15 with only a few, high-probability samples, such that the term f/p is approximately constant, and hence the variance of our MC estimator stays low.

Indeed, there is an optimal distribution to pick, namely the integrand f, normalized over the integration domain Ω.[5] Picking samples from this distribution is called *perfect importance sampling* and eliminates variance completely. However, this choice of distribution is not practical, as the derivation of the perfect distribution p implies knowing the integral of f over the integration domain, which is exactly what we want to compute using the Monte Carlo method in the first place!

Fortunately, it is often enough to pick a distribution p that is only similar enough to the integrand f. Furthermore, for practical purposes, the distribution p should be easy to sample from; thus one has to trade off the similarity to the integrand and the convenience of sampling. In some cases, there is no single good choice for p. As an example, the integrand of the reflection integral (Eq. 5.22) is a product of two terms, one of which is unknown. We will discuss this further in Ch. 5.

Also note that p must be a proper probability density and hence normalized over the integration domain.

2.4.2 Cumulative Distribution Inversion and Rejection Sampling

Given a choice of probability distribution p over the integration domain, we need a method to sample it, which is to say, generate a set of samples with distribution p.

The canonical approach is to analytically integrate the PDF to arrive at the cumulative distribution function (CDF) $P(x)$ and then (analytically) invert P to obtain a function $P^{-1}(x)$. Given uniformly, independent, and identically distributed variates $\xi_i \propto U(0,1)$, we can generate samples $x_i \propto p$ as:

$$x_i = P^{-1}(\xi_i),$$

essentially predetermining a certain sample probability and determining a corresponding input variable x_i. This is termed the *CDF inversion method*.

As can be seen from the description above, this method presupposes that we can integrate the PDF and then invert the CDF analytically. Of the two problems, CDF inversion is less problematic, if we can, e.g., afford to perform a binary search on P (noting that the CDF is monotonically increasing by definition). Alternative approaches are to pick another distribution p for which the analytic CDF inversion method works, or resort to *rejection sampling*: Given an arbitrary PDF p, we can generate samples in a higher-dimensional space and then only pick those that are “below” our original PDF. This obviously can be very inefficient, depending how well we can bound p in a higher-dimensional space.

2.4.3 Russian Roulette

Russian roulette (RR) is an approach to stochastically avoid the potentially expensive evaluation of an integrand, in an unbiased fashion, i.e., without any systematic error. RR is both beneficial in cases where we want to limit the evaluation depth of another recursive estimator, and also when the computaional cost of said estimator would be too prohibitive to evaluate fully. Given the estimator F, we can synthesize another estimator:

$$F' = \begin{cases} \frac{F}{\alpha} & \text{if } \xi < \alpha \\ 0 & \text{otherwise} \end{cases}, \tag{2.16}$$

[5] Assuming f is non-negative.

where $\alpha \in]0,1[\subset \mathbb{R}$ is some freely chosen constant, excluding the corner cases of 0 or 1, and $\xi \propto U(0,1)$. We note that the expected value is the same:

$$E\left[f'\right] = \alpha \frac{E\left[f\right]}{\alpha} + (1-\alpha)\,0 = E\left[f\right], \tag{2.17}$$

but variance is potentially increased [229].

2.4.4 Higher-dimensional Integrals

The Monte Carlo integration method can be generalized to multidimensional (real-valued) integrals easily. In fact, this is the scenario where they are known to excel in relation to other methods, not least by the curse of dimensionality, as the root-mean-square error of MC integration converges at $O(N^{-1/2})$, which only depends on the number of samples, N, but not the dimensionality of f [229]. In low-dimensional settings, deterministic quadrature rules are usually preferred for numerical integration [184]. Here, we will only briefly state the form of a multidimensional estimator:

$$I = \int_{\Omega} f(x_1,\dots,x_k)\,\mathrm{d}\mu(x_1,\dots,x_k) \approx \frac{1}{N}\sum_{i=1}^{N} \frac{f(x_{1,i},\dots,x_{k,i})}{p_\mu(x_{1,i},\dots,x_{k,i})}. \tag{2.18}$$

We note that the PDF potentially depends on all input variables of the integrand. If, however, the samples are taken in an independent fashion, the PDF simplifies to a product:

$$I \approx \frac{1}{N}\sum_{i=1}^{N} \frac{f(x_{1,i},\dots,x_{k,i})}{p_\mu(x_{1,i}).\cdots p_\mu(x_{k,i})}. \tag{2.19}$$

In practice, there is usually a mix of dependent and factored parts of the PDF, accounting for local and global sampling, respectively.

2.4.5 Efficiency

The variance, and hence error, of an MC estimate can be reduced by increasing the number of samples taken. However, this also incurs a computational cost. This trade-off is expressed in the goal to maximize the *efficiency* of estimator F, defined as [229, Eq. (2.19)]:

$$\varepsilon[F] = \frac{1}{V[F]T[F]}, \tag{2.20}$$

where $V[F]$ and $T[F]$ are the estimator's variance and evaluation time, respectively. Roughly speaking, this means that variance reduction techniques pay off in better results—in our application domain, higher-quality, more noise-free images. However, it also means that in some cases, at least for a finite number of samples, a simple, high-variance method may outperform a sophisticated, low-variance method by virtue of being faster to compute.

2.5 The Fourier Transform

The Fourier transform was originally described by Jean-Baptiste Joseph Fourier in 1822 [54]. While widely used across many fields of engineering, in our context we will find it useful to describe the process of image sampling and explain the sources of unwanted image artifacts which are collectively known as *aliasing*. See e.g. [59, 177, 184] (or nearly any textbook on mathematical physics) for a more detailed treatment. Note that there are multiple, essentially equivalent, but slightly different ways to write the Fourier transform; we will follow the notation and conventions of Press et al. [184] here.

The main idea behind the Fourier transform is that functions can be expressed in different spaces, which are given by a set of basis functions; specifically, the Fourier transform uses the complex exponential as a basis and expresses functions in the so-called *frequency space*. The original domain may be called *spatial domain* or *time domain*, depending on context.

A building block for the Fourier transform is the concept of *convolution*; formally, the convolution of two function f and g is given by:

$$(f \otimes g)(x) = \int_{-\infty}^{\infty} f(x')g(x-x')\,\mathrm{d}x'. \tag{2.21}$$

Given a one-dimensional function (or "signal") $h : \mathbb{R} \to \mathbb{C} : t \mapsto h(t)$, we can equivalently express it as another function $H : \mathbb{R} \to \mathbb{C} : \nu \mapsto H(\nu)$ in frequency space by convolution with the complex exponential:

$$H(\nu) = \int_{-\infty}^{\infty} h(t)\,\mathrm{e}^{2\pi i\nu t}\,\mathrm{d}t \tag{2.22}$$

$$h(t) = \int_{-\infty}^{\infty} H(\nu)\,\mathrm{e}^{-2\pi i\nu t}\,\mathrm{d}\nu \tag{2.23}$$

Note that in general, H is complex even if h is strictly real, i.e., the codomain of h is $\mathbb{R}$. For even functions $h(t) = h(-t)$, however, $H(\nu)$ is real.

Sometimes, operator notation is used to express the Fourier transform, as in $\mathcal{F}[h(t)] = H(\nu)$.[6] Because of their relation by $\mathcal{F}$, one says that $H = \mathcal{F}[h]$ is the dual of h, and vice versa. Additionally, $H(\nu)$ is also called the *spectrum* of $h(t)$.

In discussing sampling and reconstruction, we will need the Fourier transform of a few functions that will be given here. Due to the symmetry of the Fourier transform (except for a sign change), transforming from spatial domain to frequency domain or vice versa is equivalent for the following functions:

[6]Which is equivalent to saying that $\mathcal{F}$ is a higher-order function, i.e., operating on other functions.

$$h_1(t) = \begin{cases} \frac{1}{2T} & \text{if } |t| < T \\ 0 & \text{otherwise.} \end{cases} \quad (2.24)$$

$$H_1(\nu) = \left(\sin \frac{\pi \nu}{T}\right) / \left(\frac{\pi \nu}{T}\right) = \operatorname{sinc} \frac{\nu}{T} \quad (2.25)$$

$$h_2(t) = c \quad (2.26)$$

$$H_2(\nu) = c \cdot \delta(\nu) \quad (2.27)$$

$$h_3(t) = T \sum_{k \in \mathbb{Z}} \delta(t - kT) \quad (2.28)$$

$$H_3(\nu) = 1/T \sum_{n \in \mathbb{Z}} \delta(\nu - \frac{n}{T}) \quad (2.29)$$

h_1 (Eq. 2.24) is the unit-area box function of width $2T$, its dual is a (scaled) sinc function H_1 (Eq. 2.25). Eqs. 2.24 and 2.27 state that the dual of a constant function is the Dirac distribution (cf. Sec. 2.2). h_3, finally, is the impulse train or shah function, an infinite series of equidistant Dirac impulses with period T; its dual H_3 is another impulse train with period $1/T$ (Eqs. 2.28 and 2.29).

Another important fact to realize is that multiplication in the spatial domain is equivalent to convolution in frequency space. Again, due to duality, the relationship holds in the other direction as well:

$$\mathcal{F}[g(t) \cdot h(t)] = (\mathcal{F}[g] \otimes \mathcal{F}[h])(\nu) \quad (2.30)$$

$$\mathcal{F}[(g \otimes)(t)] = \mathcal{F}[g](\nu) \cdot \mathcal{F}[h](\nu) \quad (2.31)$$

Oftentimes, the given signal has been discretely sampled, and hence h (and H) are only given at discrete positions $k \in [0, N)$, in which case the *discrete Fourier transform* is given by:

$$H_n = \sum_{k=0}^{N-1} h_k e^{\frac{2\pi}{N} ikn} \quad (2.32)$$

$$h_k = \frac{1}{N} \sum_{n=0}^{N-1} H_n e^{-\frac{2\pi}{N} ikn} \quad (2.33)$$

We will use this when analyzing discrete filter kernels in Sec. 4.3.3.

2.6 (Hemi-)Spherical Integration

Many concepts and equations we will see later are defined in terms of integrals on the unit-radius sphere $\mathcal{S}^2$. Here, we will introduce two measures that are useful for these kinds of integrals, including common notational shortcuts and the conversion between measures. These topics are covered in various textbooks on advanced computer graphics [42, 59, 177] and also in Veach's dissertation [229].

An integral I over a directional-dependent function $f(\omega \in \mathcal{S}^2)$ may be written in *solid angle* measure $\sigma(\omega)$ as:

$$I = \int_{\mathcal{S}^2} f(\omega)\,\mathrm{d}\sigma(\omega), \tag{2.34}$$

which is equivalent to the more traditional:

$$I = \int_0^{2\pi}\int_0^{\pi} f(\theta,\phi)\sin\theta\,\mathrm{d}\theta\,\mathrm{d}\phi, \tag{2.35}$$

where f is now given in spherical coordinates (θ,ϕ). Following the (arbitrary) definition that z points "up," we can convert between unit directions and spherical coordinates by:

$$\omega_x = \sin\theta\cos\phi, \tag{2.36}$$
$$\omega_y = \sin\theta\sin\phi, \tag{2.37}$$
$$\omega_z = \cos\theta. \tag{2.38}$$

Solid angle $\sigma(\omega)$ measures the area of a set of points ω on the unit sphere $\mathcal{S}^2$, similarly to how the area measure $A(x)$ estimates the area of a set of positions x on a surface $\mathcal{M}$.

At a surface point x with normal vector aligned to our local coordinate system, integration may easily be limited to the upper hemisphere $\mathcal{H}^2_+$ or lower hemisphere $\mathcal{H}^2_-$:

$$I = \int_0^{2\pi}\int_0^{\pi/2} f(\theta,\phi)\sin\theta\,\mathrm{d}\theta\,\mathrm{d}\phi = \int_{\mathcal{H}^2_+(x)} f(\omega)\,\mathrm{d}\sigma(\omega), \tag{2.39}$$
$$I = \int_0^{2\pi}\int_{\pi/2}^{\pi} f(\theta,\phi)\sin\theta\,\mathrm{d}\theta\,\mathrm{d}\phi = \int_{\mathcal{H}^2_-(x)} f(\omega)\,\mathrm{d}\sigma(\omega), \tag{2.40}$$
$$\tag{2.41}$$

where we have $\mathcal{S}^2 = \mathcal{H}^2_+(x) \cup \mathcal{H}^2_-(x)$, i.e., at any surface point x, the surface normal N_x (or $N(x)$) divides the sphere into two disjunct hemispheres.

Another useful measure is *projected solid angle* $\sigma^\perp$, which estimates area on the unit sphere, projected orthogonally onto a surface. Using the same spherical coordinates as before:

$$\sigma^\perp(\omega) = |\cos\theta|\,\sigma(\omega). \tag{2.42}$$

With these definitions in mind, one can introduce a few notational shortcuts:

$$\mathrm{d}x := \mathrm{d}A_x := \mathrm{d}A(x), \tag{2.43}$$
$$\mathrm{d}\omega := \mathrm{d}\sigma_\omega := \mathrm{d}\sigma(\omega), \tag{2.44}$$
$$\mathrm{d}\omega^\perp := \mathrm{d}\sigma^\perp_\omega := \mathrm{d}\sigma^\perp(\omega). \tag{2.45}$$

Using differentials this time, we finally note that one may also convert between area, solid angle, and projected solid angle, given two surface points x and y:

$$d\omega^{\perp} = \left|\cos\theta_x\right| d\omega = \frac{\left|\cos\theta_x\right| \cdot \left|\cos\theta_y\right|}{\|x-y\|^2} dA_y = G(x,y)\, dA_y, \tag{2.46}$$

where $\omega = \frac{y-x}{\|y-x\|}$, θ_x and θ_y are the local polar angles at the points, and $G(x,y)$ is the *geometry term* (see Fig. 2.2).[7]

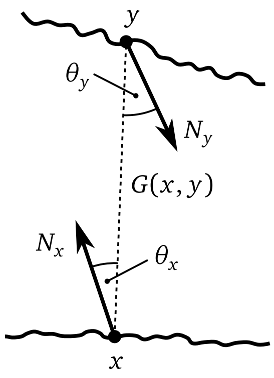

Figure 2.2: The geometry term $G(x,y)$ accounts for the relative distance and orientation of two surface points x and y.

When talking about participating media (cf. Sec. 5.9), the differential of a point, dx, may also have the meaning:

$$dx := dV_x := dV(x), \tag{2.47}$$

i.e., the differential volume around point x. The meaning will be inferred from context, seeing that by the definitions we adapt in Sec. 5.3, x can only be part of a surface $\mathcal{M}$ or a volume $\mathcal{V}$, but not both at the same time.

Even though more related to shortcut notation than to spherical integration, we will briefly state that line integrals between points x and z may be more concisely written with the following:

$$\int_x^y f(z)\, dz := \int_0^{\|x-y\|} f\left(x + s\frac{y-x}{\|x-y\|}\right) ds. \tag{2.48}$$

[7]Also known as the geometric coupling term.

Chapter 3

Basics of Image Synthesis

This chapter will discuss the basics of image synthesis—in the context of computer graphics (CG), more commonly called *rendering*. From a very high-level perspective, rendering is the process of creating images by means of an algorithm that processes an input description of a virtual scene, and outputs a single raster image (or an animated sequence of them).

Beyond this abstract description, there is a lot of variety in how these concepts are implemented, and rendering algorithms are implemented on a variety of computer platforms. For example, the development of dedicated graphics cards, together with their *graphics processing units* (GPUs), has lead to the wide availability of heterogeneous computer architectures in the PC sector. While first only accelerating simple triangle mesh processing and scanline conversion algorithms, GPUs are now massively parallel, programmable streaming processors with applications beyond just graphics, such as scientific computing or deep learning.

Though more focused on *real-time rendering*, Akenine-Möller, Haines, and Hoffman [1] give a good overview over the topic of image synthesis.

3.1 Creating Images

Though we have not yet defined what color actually is, an image is understood to be a two-dimensional distribution of color in some medium, essentially mapping each point on the—possibly unbounded—image plane to a single color.[1] Borrowing terminology from the signal processing field, this mapping from 2-D position to color may also be called *image signal* or *image function*, and we will use these terms interchangeably. Usually, the two-dimensional image plane is bounded by a rectangle.

There are several ways in which images can be represented. Yet, the vast majority of images are represented as *raster images* in the computer, i.e., a finite number of color samples on a regular rectangular grid (or raster). Every color sample is called a "picture element," a term more commonly contracted to *pixel*. This representation is very general, as by the Nyquist-Shannon sampling theorem (cf. Sec. 3.3), a finite grid of pixels is sufficient to fully reconstruct any continuous image, given that certain conditions are met by its frequency content (see e.g. [177]).

[1] This is a very technical conception of "image," but we note that this definition holds for images before the computer era, such as photographs and paintings, as well.

In practice, perfect reconstruction is not possible for several reasons. First, one has to reconstruct the image with an infinitely sized, ideal sinc filter. Second, the frequency content of images is oftentimes not limited. As an example, there are usually sharp, discontinuous edges along the silhouettes of objects.

But although images generally *do* contain arbitrarily high frequencies, the situation is not as dire as one might expect; in computer graphics, the problem is avoided by a combination of (a) prefiltering the signal where possible, (b) taking many samples when this is not possible and (c) using pixel filters that approximate the ideal sinc but have finite support.

Hence even images not meeting these band-limitation constraints can still be represented reasonably well.

To appreciate the problem we are trying to solve in rendering, note that from a data-oriented perspective, an image is simply a two-dimensional array of, say, 24-bit encoded color values. However, even when restricting ourselves to 8-bit grayscale images at a fixed resolution of, say, 100×100 pixels, the space of all images is already gigantic, e.g., there are $2^{8\times100\times100} \approx 10^{24082.398}$ different images. Granted, a human observer will perceive many of them as virtually identical, i.e., rotated, mirrored, darkened, etc. versions of the same image, and many will be just noise. But this hardly affects the (exponentially) vast size of our potential search space, so we need to restrict ourselves to models that generate a subset of this image space in a more coordinated manner.

Since for most images, there is no computational interdependence between pixels, rendering is commonly called *embarrassingly parallel*. In theory, we could compute every pixel on a separate processor, and at least in the Monte Carlo context of physically based rendering, the computation of a single pixel could be distributed further.

Given these assumption, we can conceptually think of "rendering" as the following simple procedure:

```
# render a rectangular raster image with resolution w * h
def render_image(w, h):
    for j in range(0, h): # optionally in parallel
        for i in range(0, w): # optionally in parallel
            set_pixel(i, j, pixel_measurement(i, j, w, h))
```

Beyond our array-of-colors description, and on a more concrete level, we want to render images of three-dimensional objects, which at the very least involves finding out which part of a three-dimensional scene is visible through each screen pixel. Popular algorithms are either *rasterization* or *ray tracing*.

Rasterization iterates over all geometrical representations of objects in the scene—typically triangles or polygons—, and projects each of them onto the screen, *shading* each projected fragment in turn. This shading procedure determines a fragment's color by using the local information at the fragment—such as geometrical attributes and material information–, as well as global information of the scene, to compute a single color value.

Ray tracing essentially follows the pseudocode procedure outlined above, casting a ray for each pixel, finding the closest intersection point in the virtual scene, and determining its color. As one iterates over image pixels and the other over objects, ray tracing is sometimes called an image-order algorithm and rasterization is called an object-order algorithm.

Closing our general discussion on the creation of images, we mention that there are other important image representations, most notably *vector images*, which represent images as a collection of two-dimensional geometric shapes such as polygons, circles, ellipses, parametric curves, etc. While vector images have advantages such as being resolution-independent, and are widely used for illustrations, diagrams, icons, etc., the simple yet flexible nature of raster images makes them much more amenable to represent just about any image signal, both real-world and synthetic, including photographs and simulations thereof. For the remainder of this thesis, we will focus on raster images, not the least because all current relevant display devices are raster-based.

Also, rendering what is seen through a pixel is not the only type of image that can be created with the computer as a medium; the entire field of *non-photorealistic rendering* (NPR) focuses on other styles and techniques for creating images, including hybrid 2-D/3-D methods and the simulation of traditional artistic techniques [60].[2]

3.2 Color

With a general idea of how to create images algorithmically, we still need to define what “color” is.

Color is the subjective reaction to a stimulus of electromagnetic energy in the visible range, distributed over a spectrum of wavelengths (more on this in Sec. 5.1). To arrive at an objective measure, we will simply say that color *is* this spectral power distribution (SPD) [1,59,177].[3]

Another way of seeing an image is thus as a three-dimensional function $f : \mathbb{R}^2 \times \mathbb{R}^+ \mapsto \mathbb{R}$, relating spatial position on the image plane and wavelength to “intensity” values. The job of rendering is then to evaluate this function at appropriate inputs.

Research into the human visual system (HVS) indicates that humans are trichromatic, i.e., arbitrary spectra are internally represented and processed using the inputs of only three primary color receptors. Therefore, different SPDs can lead to the same color perception; such spectra are called *metamers*.

Large-scale experiments of the *Commission internationale de l'éclairage* (CIE) in the first half of the 20th century lead to the development of a three-dimensional reference color space called XYZ. To convert from an SPD $P(\lambda)$ to the CIE XYZ color space, we evaluate the following integrals:

$$X = \int_{\Lambda} P(\lambda)\bar{x}(\lambda)\,\mathrm{d}\lambda, \tag{3.1}$$

$$Y = \int_{\Lambda} P(\lambda)\bar{y}(\lambda)\,\mathrm{d}\lambda, \tag{3.2}$$

$$Z = \int_{\Lambda} P(\lambda)\bar{z}(\lambda)\,\mathrm{d}\lambda, \tag{3.3}$$

where Λ is a convenient wavelength range, e.g. $[380, 780]$ for visible light, and the *spectral matching curves* $\bar{x}$, $\bar{y}$, $\bar{z}$ are normalized such that $\int_0^\infty \bar{y}(\lambda)\,\mathrm{d}\lambda = 1$. The

[2]“Artistic rendering” would be a lot more descriptive for this than “non-photorealistic.”

[3]Thereby ignoring (some) aspects on the psychology of perception, as is (still) common in CG.

matching curves map the average human observers' sensitivity to different spectra under daylight conditions to three hypothetical light sources, X, Y, and Z, such that Y represents the brightness, or *luminance* of a color.[4]

Displays usually have three red, green, and blue (RGB) subpixels (cf. Sec. 3.4), and can only reproduce a subset of possible perceptible colors, i.e., they have a limited *gamut*. To display XYZ colors, we thus linearly transform them to a different three-dimensional (RGB) space, which can be expressed as multiplication by a 3×3 matrix. The exact spectra of RGB subpixels vary from display to display, but there are standardized color spaces such as sRGB. To transform from XYZ to sRGB with a D65 white point, we do the following:

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 3.2404542 & -1.5371385 & -0.4985314 \\ -0.9692660 & 1.8760108 & 0.0415560 \\ 0.0556434 & -0.2040259 & 1.0572252 \end{pmatrix} \cdot \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{3.4}$$

Such matrices can be arrived at by convolution of the RGB primaries' spectra with the XYZ matching curves.

An important detail in displaying images is that, for historical and practical reasons, displays quantize color values and encode them in a non-linear space. This process is discussed in the context of *gamma conversion* or *gamma compression*, and has been introduced to balance quantization at low bit-depths, e.g., 8 bits per primary color, against the HVS's non-linear contrast perception, such that the perceived difference in brightness of adjacent pixel values is approximately constant [1]. The response curves roughly follow a power law, e.g.:

$$V = v^{\gamma} \Leftrightarrow v = V^{1/\gamma}, \tag{3.5}$$

where V is the linear display output, and v is the gamma-corrected input value. Conversely, for e.g., a display with $\gamma = 2.2$, a linear value $V = 0.7$ would have to be output as an 8-bit pixel value $v = 217$, such that $\left(\frac{217}{255}\right)^{\gamma} \approx 0.7011695 \approx V$. We note that the dynamic range of displays is also limited, as a pixel can only be so dark or bright; low-bit quantization is also inappropriate for larger dynamic ranges, and gamma correction is mainly applied to use 8-bit "true color" values more intelligently. Of note, while rendering should be done in a linear space, this has oftentimes been ignored in the past (or implemented wrongly, e.g., texture filtering in gamma space [1]).

As a final note, using the CIE spectral matching curves may be ill-advised when simulating imaging systems such as photographic cameras, as these may have different response curves, such that output RGB value may not be in a well-defined color space. As such, one may be better off directly using the camera response curves for rendering in order to more accurately reproduce the behavior of real imaging systems.

We will return to the topic of color when talking about subpixel rendering in Ch. 4.

[4]Later experiments also produced matching curves under low-light conditions.

3.3 Sampling and Reconstruction

Sampling and reconstruction is an integral part of computer graphics (CG) and image processing and has been widely studied for different tasks such as text rendering [101], polygon rasterization [41] or image filtering [147]. Pharr and Humphreys [177] give a good introductory to the subject, and our summary is loosely based on theirs.

Sampling and reconstruction of signals can be quite straightforwardly and elegantly expressed using the Fourier transform (Sec. 2.5) as a tool for analysis. In our case, we usually have three two-dimensional functions $r(x,y), g(x,y), b(x,y)$, representing the red, green, and blue color channels of an image to be rendered. Noting that Fourier analysis can be extended quite simply to N-dimensional domains, and to shorten our discussion, we will restrict ourselves to a single one-dimensional "image" $f(x)$

In reproducing images, the first step is *sampling*, whereby we evaluate our signal $f(x)$ at evenly spaced sample locations x_k, producing sample values $f_k = f(x_k)$. Effectively, this is synonymous to multiplying $f(x)$ with an impulse train, where the sample spacing in given by its period T. Applying what we mentioned earlier about the Fourier transform, this implies that sampling is convolution with an impulse train (of period $1/T$) in the frequency domain, which hence means we are creating an infinite number of copies of the spectrum of f, $F(\nu) = \mathcal{F}[f(x)]$.

In the *reconstruction* step, we (hope to) recreate our original function f from just the sample values f_k. Formally, we can perfectly recreate f by convolving our samples with an *ideal reconstruction filter*, namely the normalized sinc function. Again remembering the duality of convolution in space and multiplication in frequency, this amounts to multiplying our sampled signal with a box function in the frequency domain, effectively cutting off the spectrum outside a narrow frequency band around 0, the width of which is given by the sinc function's period. Choosing the width appropriately, we arrive at a single copy of F, which we already know to be the original function f in the spatial domain.

However, perfectly reconstructing the signal requires the infinite series of copies to not overlap each other in the frequency domain, so that an appropriate box function (in frequency space) can return exactly one copy of $F(\nu)$; otherwise, the copies are known to *alias* each other. This clearly depends on the period of the impulse train used to sample the signal, with shorter periods resulting in larger periods in the frequency domain, and hence a higher chance of not leading to aliasing. Furthermore, the frequency spectrum of f must also be finite, otherwise we cannot hope to ever isolate a single copy with a box function.

This property of sampling and reconstruction is formally stated by the *sampling theorem*: Any band-limited function (i.e., $F(\nu) = 0 \forall \nu < \nu_0$ for some ν_o) can be perfectly reconstructed when placing samples a maximum distance of $1/(2\nu_0)$ apart.

While a powerful result, in practice this may fail for three reasons: (1) we cannot afford to sample at such a high rate, (2) our image function f is not band-limited, and (3) we cannot use the ideal reconstruction filter, which has infinite support. Because of this, one instead uses *anti-aliasing* techniques to dampen aliasing, instead of avoiding it entirely.

One approach is to shift the locations of the sample points x_i randomly (within an interval of size T). While this does nothing to remove aliasing, aliasing artifacts then manifest as noise, and not as fixed structures. [5]. The human visual system perceives noise as less severe of an error. As a side note, this is what we will be doing when integrating the measurement equation (Eq. 5.32) using Monte Carlo later.[6]

A second approach to anti-aliasing is *adaptive sampling*, which tries to concentrate samples where frequencies are high or even infinite, such as at steps in the function $f(x)$. However, as one in general does not know where these steps happen in the domain of f, one can only use the samples already taken and apply heuristics where to place more of them.

A third anti-aliasing method is *prefiltering*, whereby one gives up the hope of reconstructing f perfectly, and instead creates a band-limited version $\hat{f}$, which is as close to f as possible, but known to have finite support in the frequency domain. Ideally, one would like to use the sinc function as a filter kernel, e.g. $\hat{f} = f \otimes \text{sinc}$, but we have already stated that its infinite support makes this impractical. Instead, one uses finite filters that mimic the shape of the (truncated) sinc kernel, such as the Mitchell filter [147]. Depending on our image function, we may also be able to "force" the frequency limitation externally, as in procedural texturing.

Of the three anti-aliasing techniques, prefiltering is arguably the most desirable in CG. In closing this section, we will briefly summarize work related to texture (pre)filtering, which is a crucial part in the rendering pipeline. In their seminal work, Greene and Heckbert [62] introduce the Elliptical Weighted Average (EWA) filter for high-quality texture filtering. McCormack et al. [142] describe *Feline*, a fast approximation of the EWA using MIP-mapping that lowers the filter cost. Shin et al. [207] improve this work by introducing an additional weighting of the samples. Recently, Mavridis and Papaioannou [138] describe a filter for modern GPUs that closely matches the EWA filter. Selecting an appropriate filter size for, e.g., ray tracing has been described by Igehy [81], and similar ideas apply to rasterization-based rendering.

3.4 Display Devices

In this section we will briefly outline some of the practicalities of physical displays; Glassner [59] discusses (CRT) display geometry and its implications for sampling and reconstruction, an aspect we will revisit (for newer display technologies) in Ch. 4.

In order to present the image we have computed to an observer, we must convert the encoded image to something that can be visually perceived. Common display technology involves additively blending together primaries designed to span a gamut of colors. Printing, on the other hand, is subtractive and passive, e.g. one needs a light source to illuminate a printed page; this is similar for e-ink displays.

Both methods are limited in the range and granularity of brightness they can reproduce. Furthermore, the resolution—typically measured in dots per inch (DPI)—is limited as well. We note that image resolutions are typically given as the number of pixels along the horizontal and vertical dimensions, but what matters just as much is the effective size of each pixel.

[5]"jaggies" or "crawlies," in the context of rasterization.

[6]Hoping the emitted importance $W_e(x, \omega)$ incorporates some pixel filter kernel as well, to further dampen aliasing. Foreshadowing later concepts, one could indeed claim that sampling path space, or rather the measurement contribution function, which is clearly not band-limited, with random samples is a form of anti-aliasing. The prime reason for using MC is the high dimensionality of path space, though.

A typical wide-screen display will, say, have a resolution of 1920×1080 RGB pixels, at a color depth of 24 bits per pixel, for a total uncompressed storage requirement of $\approx$ 6.22 MB. However, as of this writing, larger display sizes such as “4K” or “5K” are becoming standard, even among hand-held devices such as mobile phones.

Effective pixel sizes may also vary widely. By way of example,[7] a typical 27-inch desktop monitor at 2560×1440 image resolution will have a visible area of about $598 \times 336\,\mathrm{mm}^2$, for a pixel pitch of about 0.23 millimeters or, alternatively, a display resolution of about 108.8 dots per inch (DPI). By comparison, an example higher-end 4 inch mobile phone display with 1136×640 pixels has a visible area of about $49.9 \times 88.5\,\mathrm{mm}^2$, for a pixel pitch of 0.08 mm, or a—much larger—resolution of 326 DPI. Note that the effective resolution also depends on the distance of the observer to the display.

Recently, Didyk et al. [38, 227] describe an interesting approach to increasing the apparent display resolution for moving images and animations by accounting for the integration of the signal in the HVS.

3.5 A Note on Real-time Rendering

Given the right combination of hardware, software, and algorithms, it is possible to compute appealing images in a very short amount of time. Indeed, *real-time rendering* is an entire sub-field of computer graphics devoted to methods for computing images of reasonable quality for immediate display, fast enough that the human visual system (HVS) fuses them together, giving rise to the illusion of fluid animation.

The key difference beyond movies and animations, which *also* give the illusion of fluid motion, is that one can interact with the computer program creating the images, thus influencing what is generated next in (soft) real-time. Apart from video and computer games, this interactive experience of using a computer has also become a key component of graphical user interfaces (GUIs), to the point where the average user is shielded so much from the underlying system that they may think of the GUI as the actual operating system.

Despite impressive efforts towards it, achieving photorealism in real time is still an open problem in computer graphics—if not even the holy grail. As with many things, rendering images is a speed-vs.-complexity trade-off, and simplifying assumptions must be made depending on the application, often using tricks that capitalize on deficiencies of human perception.

We refer the reader to dedicated literature [1] for more information on real-time rendering, but do note that interactivity and fast rendering are also a key component both to optimal subpixel rendering (Ch. 4), scalable many-lights methods (Ch. 6), and the user interface side of our system of visualization (Ch. 8) and artistic editing of light transport (Ch. 9).

[7]These calculations assume pixels on a square grid, i.e., their horizontal and vertical pitches are supposed to be equal, such that pixel size can be estimated from the display resolution and diagonal alone.

Chapter 4

Optimal Subpixel Filtering

In this chapter,[1] we summarize the importance of subpixel rendering for the accurate reproduction of high-frequency geometry and show an approach to optimal subpixel filtering for diverse display geometries. We make the following contributions:

- A generalization of the approach of Platt [179] to obtain filters for displays with diverse 1-D and 2-D subpixel layouts, including displays that provide a fourth primary color to improve luminance efficiency [43].
- A derivation of analytic filters which are both easier to evaluate in practice and more suitable for image filtering.
- A combination of our filters together with GPU-based multi-sample anti-aliasing, which yields subpixel rendering at little additional cost compared to standard anti-aliasing.
- An adaptation of standard texture sampling to directly employ subpixel-aware rendering.
- Analysis of two user studies simulating different subpixel layouts and filters, confirming the optimality of our proposed filters.

4.1 Introduction

Besides dynamic range and gamut, display resolution is a physical limitation restricting the ability to reproduce real-world images. Recent work bypasses some of these limitations by exploiting the characteristics of the human visual system (HVS), e.g., by considering contrast sensitivity to enhance the perceived contrast, or accounting for retinal integration over time to increase the apparent resolution [38].

Beyond the idealized specifications given in Sec. 3.4, one has to realize that the pixels of real display devices are not capable of showing all colors at the same screen location. Rather, pixels are made out of monochromatic *subpixels*, which—given the right distance to the viewer—fuse into what appears to be multi-colored pixels. This is also true for older cathode-ray tube (CRT) monitors [59].

[1]This chapter is based on the previous publication [47].

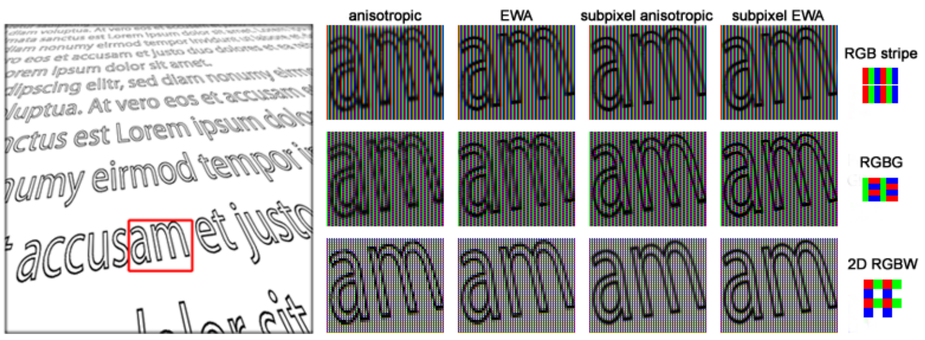

Figure 4.1: Subpixel rendering of a plane with a structured texture for different subpixel layouts, similar to those found in diverse displays nowadays. This image showcases high-quality texture filtering, one of the applications of our proposed optimal filtering framework, comparing standard filtering (anisotropic and EWA) to our subpixel-aware filtering (again, using anisotropic and EWA). Note how our method significantly reduces aliasing. Image taken from [47].

In this chapter, we focus on better utilizing the display resolution: normally, every pixel is assigned a color and the subpixels' brightness is adjusted such that their joint emission creates the respective stimulus.

By treating subpixels as individual entities and taking their relative position into account, there is an opportunity to increase the perceived resolution of the display. Microsoft's ClearType functionality [179, 180] is a well-known example of subpixel-aware rendering targeted at font rasterization. Other rendering scenarios with fine-scale geometry, such as grass blades, hair, wireframe or line drawings also benefit from the higher spatial resolution afforded by subpixel rendering.

Obviously, a naive sampling of the image signal for each subpixel, i.e. ignoring the subpixels' respective colors, yields distracting color fringing artifacts. The reason why subpixel rendering can work is that the contrast sensitivity of human vision is different for luminance and chrominance. This allows one to derive an image filter that provides higher perceived resolution, while suppressing color fringing to an imperceptible degree, as shown by Platt [179] and Klompenhouwer et al. [118].

We extend this work to find optimal filters for subpixel rendering for a diverse set of 1-D and 2-D subpixel layout patterns. We demonstrate that the optimal filters can be approximated well with analytic functions, and incorporate our filters into GPU-based multi-sample anti-aliasing to yield subpixel rendering at a very low cost (1–2 ms filtering time at HD resolution). We also show that texture filtering can be adapted to perform efficient subpixel rendering. Finally, we analyze the findings of a user study we conducted, underpinning the increased visual fidelity that can be achieved for diverse display layouts, by using our optimal filters.

4.2 Previous Work

As we have already general sampling and reconstruction (in a monochromatic setting) in Sec. 3.3, we will here only mention woks explicitly focused on subpixel rendering.

Platt [179] and Platt et al. [180] derive subpixel-aware filters for luminance and chrominance signals, assuming an RGB Stripe layout, by converting an RGB image into an opponent color space and using perceptual metrics. Messing and Daly's method [144] operates in a similar manner, but does not derive optimal filters.

Messing et al. [146] adapt Platt et al.'s work, proposing to use constrained optimization to solve for filters than can mask out defective subpixels; they demonstrate how their framework can be set up for one specific 2-D subpixel arrangement. Later work by the same authors [145] extends this to arbitrary subpixel patterns. However, as in Platt et al.'s work, the resulting filters are given in discretized form, i.e., for a specific display resolution, while our optimal filters are analytic and can thus be employed to efficiently downsample input signals of arbitrarily high resolution. and do not need explicit storage of filter kernels, which makes them more GPU-friendly. Furthermore, since our filters do not attempt to compensate for defective display hardware, we decided to directly build our method on the simpler approach by Platt et al.

The most widespread application of subpixel rendering is Microsoft's ClearType [179, 180], which is targeted at font rendering. It has even been evaluated perceptually using S-CIELAB [246]. Klompenhouwer et al. [118] describe a subpixel-aware scaling of images and conclude that considering subpixel arrangements is a crucial part of the signal processing chain. Fang et al. [49] address subpixel-aware image downscaling (fixed scale) using edge detection and empirically determined low-pass filters for suppressing chromaticity errors. Subpixel rendering is also closely related to image sensors: most cameras use a sensor overlaid with a so-called Bayer filter to selectively sample different wavelengths at interleaved locations, and the full RGB image is reconstructed by interpolating the missing data. Atcheson [5] describes a subpixel-aware reconstruction for directly displaying images from Bayer patterns.

4.3 Derivation of Filters

In this section, we first briefly describe the approach for deriving optimal filters for subpixel rendering, as pioneered by Platt [179]. We then analyze the resulting filters to arrive at analytic versions, and eventually generalize the optimal filter derivation framework to arbitrary subpixel patterns.

For displaying an image on matrix displays, such as LCD monitors, the image signal is normally sampled at the centers of the pixels, or an average pixel color is determined. This image sample then controls the brightness of the individual subpixels to create the perceived color. Subpixel rendering in contrast takes the underlying spatial subpixel structure within a pixel into account and retrieves an image sample for *each* subpixel. This approach increases the perceived resolution of the display, however, it is likely to cause obvious color artifacts if the display colors of the subpixels are ignored (see Klompenhouwer et al. [118] for a discussion). These artifacts are not surprising, as the chrominance signal is now under-sampled and therefore prone to aliasing.

Fortunately, color aliasing can be tolerated to a certain degree, which can be understood by converting a pixel's displayed color to an *opponent color space* describing color in a manner similar to the human visual system (HVS). Color is separated into a luminance channel, as well as an opponent red-green and yellow-blue chrominance channel. The HVS acts as a low-pass filter in this color space and diminishes the significance of the chrominance channels quickly for high frequency image details. This can be modeled with the (chrominance) contrast sensitivity function ((C-)CSF, Fig. 4.2), and in turn be exploited to reduce the amount of perceptible aliasing.

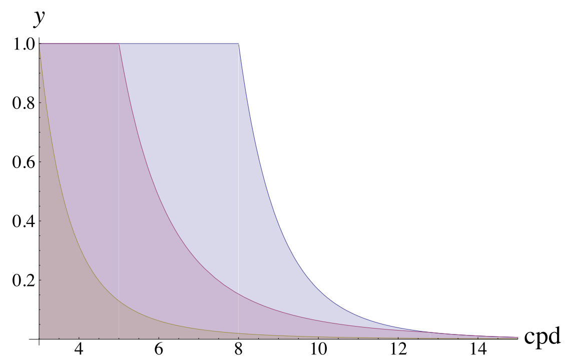

Figure 4.2: The simplified chromatic contrast sensitivity function is shown (right to left) for Y', C_1 and C_2 [179]. Image taken from [47].

4.3.1 The Contrast Sensitivity Function

The human eye's sensitivity to luminance differences depends on their spatial frequency [16]. This sensitivity is greatest at about 5–10 cycles per degree (CPD), and falls off for lower or higher frequencies. The idea of contrast sensitivity can also be applied to color differences, and is usually given for opponent colors, i.e. red-green and yellow-blue. The sensitivity for opponent color gratings (chrominance contrast sensitivity) is highest at small CPDs and then falls off quickly [149]. The actual shape of the function depends on the exact opponent color space. We follow Platt [179] and approximate the actual contrast sensitivity functions with simple functions, as depicted in Fig. 4.2.

4.3.2 Optimal Filtering

The goal of an optimal filter for subpixel rendering is to suppress color aliasing to be unnoticeable, while keeping a high spatial resolution of the luminance signal. We first review Platt's [179] approach to derive an optimal filter for RGB-stripe matrix displays. This assumes that a display pixel has $k = 3$ subpixels, i.e., a red, green, and blue subpixel. It also assumes that the RGB image signal is sampled at subpixels, denoted as α_k. At each subpixel position, an RGB color value γ_k is sampled from the image signal. The design of an optimal filter is inspired by a perceptual error metric, that seeks to minimize the error that is introduced when displaying an RGB color value γ_k of the image signal with only the single color intensity α_k of the k-th subpixel in an arbitrary scanline of the display.

Computing this error in a way that exploits the characteristics of the HVS for optimal display requires a conversion to an opponent color space, where the error at the position of the k-th subpixel can be expressed as:

$$\mathbf{E}(k) = \underbrace{3\mathbf{m}_k\alpha_k}_{\text{displayed color}} - \underbrace{\sum_{d=1}^{3}\mathbf{m}_d G_{k,d}}_{\text{image signal}} , \tag{4.1}$$

where d iterates over the RGB color channels and $\mathbf{m}_i$ is the (i mod 3)-th column vector of a 3×3 matrix that converts RGB to an opponent color space. We use the

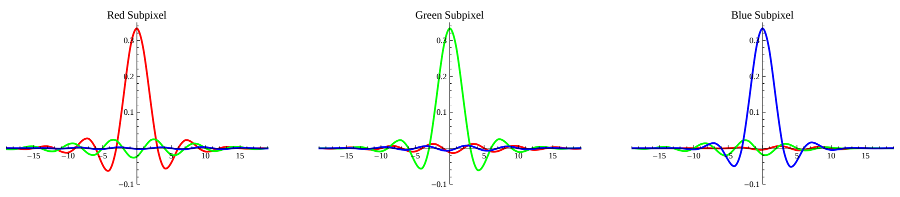

Figure 4.3: Optimal filters for the RGB stripe layout. The curves are colored according to the color component that influences the subpixel denoted. The x-axis is given in subpixel units, i.e. every pixel is three units wide. Image taken from [47].

linear $Y'C_1C_2$ opponent color space which was designed to work best with linear transformations [99] and for which the conversion from RGB is given by:

$$M_{Y'C_1C_2} = \begin{pmatrix} 0.23 & 0.73 & 0.06 \\ 0.20 & -0.31 & 0.11 \\ 0.23 & 0.66 & -0.89 \end{pmatrix}. \tag{4.2}$$

To measure the significance of the error on the perceived image it has to be weighted with the contrast sensitivity function (CSF). To this end, the error must be represented in frequency space:

$$\hat{\mathbf{e}}_f = \sum_{k=0}^{N-1} \mathbf{E}(k) \exp\left(-\frac{2i\pi k f}{N}\right), \tag{4.3}$$

where $\hat{\mathbf{e}}_f$ denotes one of the N (complex) Fourier coefficients which are computed as a weighted sum over the error at each of the N subpixels in a single scanline. The contrast sensitivity function acts as a low-pass filter on the error which quickly filters out errors that occur at high frequencies in the image signal. As mentioned before, we use the C-CSF suggested by Platt [179] (Fig. 4.2), which models it as a set of simple low-pass filters. We also examined other C-CSF models that fit experimental data (on a small sample of subjects) more accurately [149]; however, these functions tend to penalize luminance for low frequencies, which leads to band-pass filters. These have high support in the spatial domain and are also harder to model analytically. To support our original goal of efficient, low-cost subpixel rendering, we decided to utilize the simpler C-CSF by Platt.

The total squared error over all frequencies, luminance, and opponent color channels can then be computed as a sum over dot products:

$$\mathcal{E} = \sum_{c \in \{Y', C_1, C_2\}} \sum_{n=0}^{N-1} \mathbf{W}_c(f_n) \left\langle \hat{\mathbf{e}}_n, \hat{\mathbf{e}}_n^* \right\rangle, \tag{4.4}$$

where $\mathbf{W}_c(.)$ is the CSF for opponent channel c, and n iterates over all N Fourier coefficients $\hat{\mathbf{e}}_n$. f_n is the frequency (in cycles per degree) the n-th Fourier coefficient corresponds to. Note that the relation between Fourier coefficients and spatial frequency depends on the distance of the viewer to the display and the display's resolution. In the rest of the paper, we assume that 300 subpixels are viewed at roughly 16 cpd. With a pixel density of 100 dots per inch (DPI) this corresponds to a viewer distance of roughly 25 cm. These numbers can easily be adjusted for different viewing configurations.

To derive the subpixel intensity values that minimize the error, one has to compute the gradient

$$\nabla\mathcal{E} = \left(\frac{\partial E}{\partial \alpha_1}, \cdots, \frac{\partial E}{\partial \alpha_N}\right)$$

and solve for $\nabla\mathcal{E} = 0$. The i-th component of $\nabla\mathcal{E}$ can be derived as:

$$\frac{\partial E}{\partial \alpha_i} = \sum_{c,n,k} W_c(f_n) m_{c,i} \left(m_{c,k} A_k - \sum_{d=1}^{3} m_{c,d} \gamma_{k,d}\right) \cos\left(\frac{2n(k-i)}{\pi^{-1} N}\right), \tag{4.5}$$

where c runs over all three opponent color channels, and n and k run over N subpixels and N Fourier coefficients, respectively.

Using Eq. 4.5, one can restructure $\nabla\mathcal{E} = 0$ into a system of linear equations, whose solution yields a direct mapping of the RGB color values to subpixel intensity values:

$$\mathbf{A}\mathbf{x} = \sum_{d=1}^{3} \mathbf{B}_d \mathbf{y}_d \;\rightarrow\; \mathbf{x} = \mathbf{A}^{-1} \sum_{d=1}^{3} \mathbf{B}_d \mathbf{y}_d, \tag{4.6}$$

with $\mathbf{x} = \left(\alpha_1, \cdots, \alpha_N\right)^T$ and $\mathbf{y}_d = \left(\gamma_{1,d}, \cdots, \gamma_{N,d}\right)^T$.

The matrices $\mathbf{C}_d = \mathbf{A}^{-1}\mathbf{B}_d$ form a direct mapping of the d-th color channel of the RGB values γ_k to the subpixel intensities α_k of the k-th subpixel. Platt [179] observed that these mapping matrices are in block-Toeplitz form and thus, instead of solving the linear system for each scanline, nine discrete filter kernels can be extracted from the matrices. To control the intensity of each subpixel, one obtains three discrete filter kernels for every subpixel color, which combine all three RGB values into a single intensity α_k of the k-th subpixel. All 9 filter kernels are shown in Fig. 4.3. These filters can be stored and applied directly to obtain a subpixel-filtered image. However, due to their large spatial extent, the direct application of these filters is costly and we thus strive for simplifications without significant loss in filtering quality. Furthermore, we seek to model these filters analytically in order to simplify their description and evaluation. This is discussed in the following section.

4.3.3 Frequency Analysis and Analytic Formulation

To gain further insight into the optimal filters, we investigated the impulse response spectrum of each filter kernel illustrated in Fig. 4.4. We note that for each subpixel we obtain a low-pass filter and two band-pass filters that peak at about 16 cpd. We also see that the impact that band-pass filters have on the intensity of the subpixel is restricted to a narrow frequency band. Furthermore, due to their low amplitude, they hardly contribute to the intensity at all. Due to their limited support in frequency space and their marginal contribution, it is safe to omit those filters without introducing distracting artifacts. We have also verified this observation in our experiments (see Sec. 4.5.1).

The filter kernels based on the derivation from Sec. 4.3.2 are discrete and only defined at subpixel positions. This is highly impractical for image filtering applications, as the filter usually needs to be evaluated for multiple pixel samples that do not align with the subpixel positions. This implies that the optimal filter would have to be recomputed and stored for each sample layout used. Hence, we strive

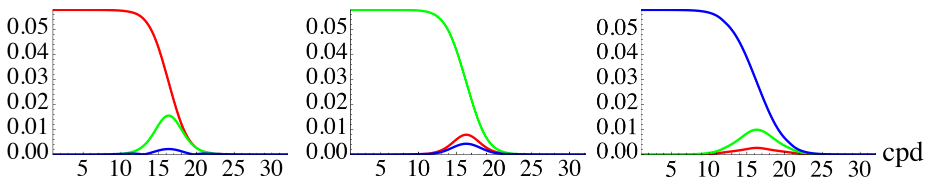

Figure 4.4: Filter response spectra for all three subpixels of an RGB display. Colored curves show the filter response of the corresponding color channels in the RGB signal that influence the red (left), green (middle) and blue (right) subpixels. Image taken from [47].

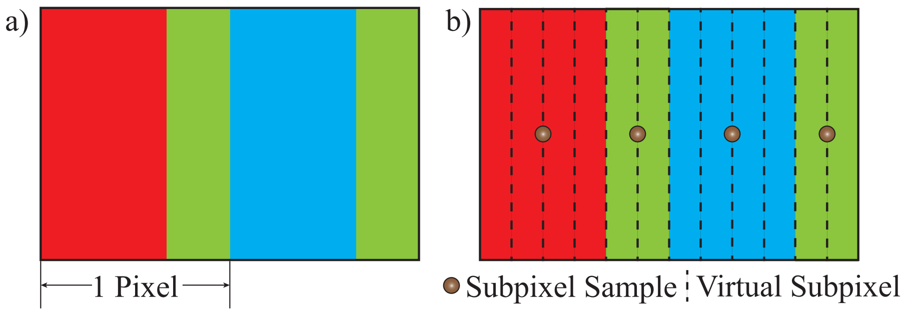

Figure 4.5: Using virtual subpixels to model nontrivial subpixel patterns. a) The *PenTile* RGBG subpixel layout reduces the number of red/blue subpixels, while increasing their size; a single pixel consists of only two subpixels. b) We use virtual subpixels to incorporate such patterns into our derivation. Image taken from [47].

for an analytic description of the filter kernels. Empirically, we have found that the low-pass filters can be modeled well with a Gaussian-windowed sinc function:

$$f(x) = \frac{1}{3}\frac{\sin ax}{ax}\exp\left(-\frac{x^2}{b}\right). \tag{4.7}$$

Using a non-linear least square fit, we determine the coefficients a and b as presented in Tab. 4.1. Our proposed analytic model fits the data exceptionally well, with a root mean squared error of less than 0.02 for each color channel. We also considered fitting well-known filters like the Mitchell filter [147], but deemed it unsuitable since they have only two negative lobes.

4.3.4 Arbitrary 1-D Subpixel Patterns

We now extend the basic analysis to arbitrary 1-D subpixel patterns. We exemplify this with the *PenTile* RGBG subpixel layout, which is commonly found on mobile devices such as phones. Exploiting the different sensitivity characteristics of the HVS, these displays offer less resolution for the red and blue color channels. Therefore, each pixel of the RGBG display is built of a green plus either a red or blue subpixel, where the latter have twice the spatial extent of the green subpixel (Fig. 4.5a). To apply optimal filtering, we have to take the layout of the subpixel pattern into

	RGB Stripe			
	R	G	B	–
a	$\pi/3$	$\pi/3$	$\pi/3$	–
b	92.65	76.58	46.62	–

	RGBG			
	R	G	B	G
a	$\pi/12$	$\pi/6$	$\pi/12$	$\pi/6$
b	457.37	476.62	457.37	479.06

	2-D RGBW			
	R	G	B	W
a	2.06	1.75	1.89	3.04
b	7.12	1.40	2.66	16.36

Table 4.1: Parameters for optimal filtering for displays with different subpixel patterns. For RGB stripe and 2-D RGBW parameters are given in subpixel units. RGBG parameters are shown in units of virtual subpixels (cf. Fig. 4.5).

account. For irregular sampling patterns we use zero padding by introducing virtual subpixels that are laid out in a regular pattern and then align the centers of each physical subpixel of the irregular layout to the virtual subpixels (Fig. 4.5b). This leaves virtual subpixels that cannot be assigned to physical subpixels and need to be ignored. Thus, we modify $\mathbf{m}_k$, which was used to select columns of the color conversion matrix (Eq. 4.2), so that if k does not align with a subpixel center, we set it to $\mathbf{0}$, effectively "deactivating" irrelevant virtual subpixels (similar to [145]). Fig. 4.6 shows the optimal filters we obtain for RGBG subpixels. As can be seen, the filters for red and blue subpixels adapt to the lower resolution and thus the filters have larger support than the filters for the two green subpixels.

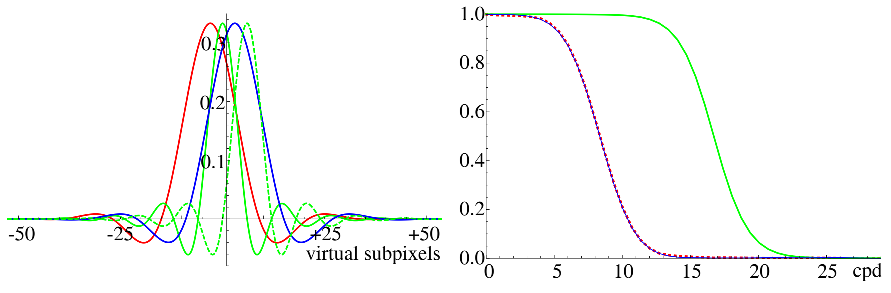

Figure 4.6: Optimal filters for the RGBG layout. Band-pass filters have been omitted, as their contribution is similarly low as for RGB displays. There are two green filters for the two different green subpixels involved in order to display an RGB color signal (cf. Fig. 4.5). The frequency plot shows the characteristics of the filters (note that the red and blue curves are very similar). Image taken from [47].

4.3.5 2-D Subpixel Patterns

Similar to Messing et al. [146], we extend the mathematical framework to two-dimensional subpixel patterns. However, we use the original formulation by Platt instead of constrained optimization, since most subpixel geometries exhibit rectangular structure, which is quite amenable to a straight-forward extension of the 1-D approach. In situations where subpixel shape is not negligible, we can introduce virtual subpixels, as in the previous subsection. In the 2-D case the error metric is a function of both the horizontal subpixel position s and the vertical subpixel position t:

$$\mathbf{E}(s,t) = 3\mathbf{m}_{s,t}\alpha_{s,t} - \sum_{d=1}^{3} \mathbf{m}_d \gamma_{s,t,d}. \tag{4.8}$$

In frequency space the error then becomes:

$$\hat{\mathbf{e}}_{n_x,n_y} = \sum_{s=1,t=1}^{N} \mathbf{E}(s,t) \exp\left(-\frac{2i\pi s n_x}{N}\right) \exp\left(-\frac{2i\pi t n_y}{N}\right). \tag{4.9}$$

As in Sec. 4.3.2, we can derive a gradient vector of the total error and set it to zero. In the 2-D case the gradient is given as:

$$\nabla\mathcal{E} = \left(\frac{\partial\mathcal{E}}{\partial\alpha_{1,1}}, \cdots, \frac{\partial\mathcal{E}}{\partial\alpha_{1,N}}, \cdots, \frac{\partial\mathcal{E}}{\partial\alpha_{N,N}}\right)^T, \tag{4.10}$$

and contains N^2 components. The analytic expression for the partial derivative of the error is:

$$\frac{\partial\mathcal{E}}{\partial\alpha_{s,t}} = \sum_{c,k,l,u,v} W_c(f_k,f_l) m_{c,s,t} \left[m_{c,u,v}\alpha_{u,v} - \sum m_{c,d}\gamma_{d,u,v}\right]\Phi, \tag{4.11}$$

with $\Phi = \cos\left(\frac{2\pi}{N}(k(u-s) + l(v-t))\right)$. As in the 1-D case, we can construct and solve a system of linear equations, where the solution of this system yields matrices which transform an entire 2-D block of RGB color values into subpixel intensities. Also in the same way, we can extract discrete filter kernels that transform color values into subpixel intensities. The system of linear equations grows quadratically with the number of subpixels and is inefficient to solve if many subpixels (i.e., hundreds) are taken into account. Fortunately, we only need to solve the equations once for each subpixel pattern.

2-D RGBW Pattern As an example, we have applied the analysis to the 2-D *PenTile* RGBW pattern, where 2×2 subpixels containing red, green, blue and white primaries form a single pixel. To be able to consider such a pattern for optimal filtering we need to treat white as a primary color and thus employ a matrix that can convert from RGBW into the $Y'C_1C_2$ opponent color space [191]:

$$M_{rgbw} = \begin{pmatrix} 0.15 & 0.47 & 0.04 & 0.37 \\ 0.13 & -0.20 & 0.07 & 0 \\ 0.14 & 0.43 & -0.57 & 0 \end{pmatrix}. \tag{4.12}$$

As most images and textures are given in RGB color space, we need to convert them first to RGBW. We do this by computing the minimum of all three RGB color

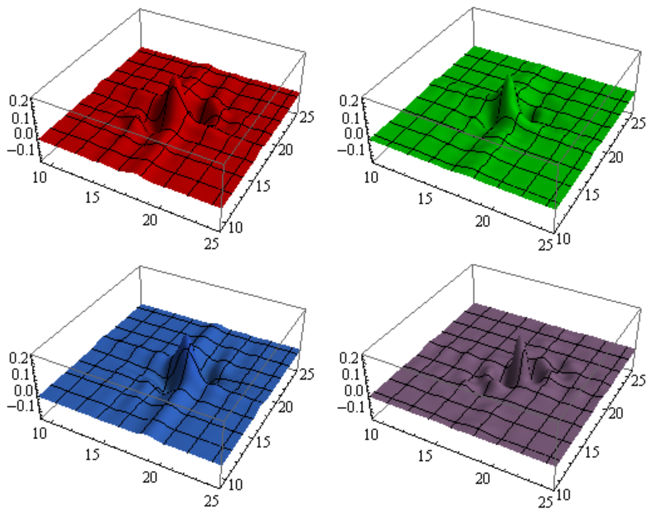

Figure 4.7: Four 2-D filters for red, green, blue, and white subpixels. The overall shape of the filters is similar to the ones in Fig. 4.3, but extended to the 2-D RGBW pattern. Image taken from [47].

channels, then subtracting that value from each component, treating it as pure white [43]. Using M_{rgbw} and the above derivation, we can compute the optimal 2-D filters which are shown in Fig. 4.7. Again, we ignore the band-pass filters and approximate the filters with low-pass characteristics using the following model:

$$f(x,y) = \frac{1}{4}\operatorname{sinc}\left(a\sqrt{x^2+y^2}\right)\exp\left(-\frac{(x^2+y^2)}{b}\right). \tag{4.13}$$

4.4 Applications

In the following we propose to include subpixel rendering in GPU-based image synthesis, which, as we will see, induces only very small overhead. In particular, we describe how multi-sampled anti-aliasing (MSAA) can incorporate subpixel rendering, as well as how texture filtering can take advantage of subpixel rendering. For brevity we restrict the discussion in the following section to the RGB stripe layout. Similar arguments can be made for other subpixel layouts.

4.4.1 MSAA Resolve

Multi-sampled anti-aliasing is a common technique used in interactive rendering to improve the quality of the displayed image. This technique computes images at a much higher resolution than required for display, using multiple color samples per pixel. Afterwards all visible color samples within a pixel are resolved into a single color value for display, applying an anti-aliasing filter that spans the entire pixel. (There also exist versions where color is computed only at a single location, but distributed over multiple visibility samples before being resolved.)

This technique can be easily extended to subpixel-awareness without increasing computation time of the resolve significantly. In order to apply subpixel filters during the MSAA resolve, we use the analytic formulation from Sec. 4.3.3 to compute the filter weights. We compute the distance d_x (in subpixel units) in the horizontal direction (for 1-D patterns) between the subpixel centers and the sample positions and compute the filter weights $w(d_x)$, exemplarily shown for the RGB stripe layout, as follows:

$$w(d_x) = \begin{cases} \frac{1}{3}\operatorname{sinc}\left(ad_x\right)\exp\left(-bd_x^2\right) & \leftrightarrow \quad |d_x| \leq 6 \\ 0 & \leftrightarrow \quad |d_x| > 6 \end{cases} \tag{4.14}$$

Since the optimal filter has infinite support, we truncate it after the first negative lobe, as the filter weights quickly diminish beyond this point. In case of the RGB stripe layout this is the case for distances d_x larger than 6 subpixels. For other layouts, this clamping distance has to be adjusted appropriately. This MSAA resolve can be executed very efficiently on the GPU, as will be shown in the results section. For 2-D patterns, we compute distances d_x and d_y and evaluate the appropriate (clamped) 2-D filter.

4.4.2 Texture Filtering

In rendering algorithms, texture filtering is an integral component. A single subpixel may cover a large texture region which must be filtered in order to avoid distracting artifacts due to undersampling of the texture. We propose subpixel texture filtering in order to increase the perceived display resolution, yielding textures that appear sharper.

Elliptical Weighted Average For high quality image texture filtering, the elliptical weighted average (EWA) [62] is often employed. This filter determines the elliptical footprint of a pixel in texture space and computes the pixel's color from all texels that fall within the elliptical region. The size of the ellipse is determined using the partial derivatives of the texture coordinates. To account for all texels that fall within the elliptical region in texture space, all texels in the bounding box of the ellipse are enumerated and tested for overlap with the ellipse.

For subpixel rendering we can directly fold subpixel filtering into the texture filtering algorithm. The optimal filter is a low-pass filter with a support of multiple subpixels on the screen. To account for this extended filter support in texture space, we scale the partial derivatives of the texture coordinates to span the filter's support and compute the ellipse from the scaled derivatives. For simplicity we assume that all subpixels share the same derivatives, hence we need to compute the ellipse only once per pixel. To compute the subpixel-filtered color value, we then need to displace the ellipse in texture space. Afterwards we traverse the common bounding box of all ellipses and compute a texel's contribution to all subpixels simultaneously. We use a Gaussian with hand-tuned standard deviation to obtain best filtering results, although any other filter kernel can be used for EWA filtering.

GPU-based Subpixel Texture Filtering While EWA filtering produces superior results, it is expensive to evaluate. It is also possible to modify GPU-based anisotropic texture filtering to become subpixel-aware. While it is not as accurate as EWA filtering, its performance is substantially higher. To this end, we apply the subpixel filter directly in screen space. We compute the filter taps by displacing a pixel's interpolated texture coordinates along the directions of its partial derivatives and then sample the texture using a trilinear or anisotropic texture look-up. For texture sampling, we require samples from multiple subpixels, which are then convolved with the subpixel filters. Consequently, we need to adjust the mip-map level from which we retrieve a texture sample. For this, we scale the partial derivatives to the extent of a subpixel and compute the appropriate mip-map level as described by Schilling et al. [199].

4.5 Results

All filters were implemented and executed on the GPU using Direct3D 11 shaders. Timings were taken on a PC with an Intel Core i7 860 processor with 2.80 GHz, 8 GB of main memory, and an AMD Radeon 5870 GPU. First we discuss our experimental evaluation of the band-pass filters derived for optimal filtering. Then we present our results for MSAA rendering for different subpixel layouts, which we simulated directly in the pixel shader, followed by a discussion of our subpixel texture filtering results. We also investigated the temporal properties of our filters, showing that our optimal filter is stable. Please see the supplementary material of the original publication [47] for an example animation. Finally, we present the findings of a user study we conducted in order to assess the visual quality of our proposed optimal subpixel filter, simulating different subpixel layouts. For all our results, we applied filtering in linear RGB color space, i.e. before gamma correction. This is the natural space for doing the MSAA resolve, but assumes linear input images for texture filtering (or requires inverse gamma correction beforehand).

4.5.1 Influence of Band-Pass Filters

We briefly discuss the influence of the omitted band-pass filters. As shown in Sec. 4.3.3 these filters only contribute to the filtered image for frequencies close to the cutoff frequency of the low-pass filters. We evaluate their influence on a high frequency pattern in Fig. 4.8, indicating that it is minimal. As can be seen in the difference image, the contributions are negligible, and become only visible when scaled 32-fold.

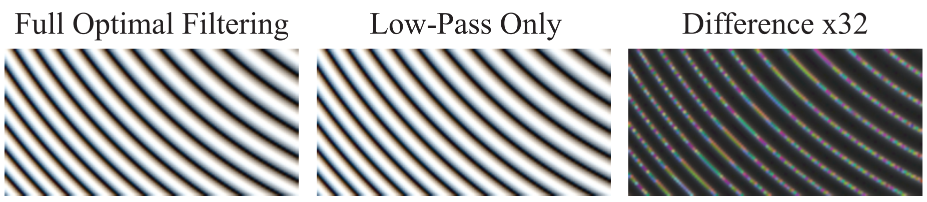

Figure 4.8: The influence of the band-pass filters obtained by optimal filtering is visually insignificant as demonstrated on this test pattern. Image taken from [47].

4.5.2 MSAA Resolve

To measure the performance of our custom MSAA resolve, we render an image with 8×MSAA in full HD resolution (1920×1080) using a render target with 8-bit precision per color channel. Fig. 4.9 shows several examples of this. (Images are best seen in a zoomed-in electronic version.[2] The original publication [47] has supplemental material demonstrating more examples.) We enabled execution of the pixel shader per image sample and thus we effectively perform super-sampling when rendering into an MSAA texture. To perform the subpixel resolve, the shader has to process 40 samples for the optimal filter, compared to 8 samples for a common per-pixel resolve pass (which corresponds to a non-subpixel-aware box filter). For efficiency we precompute the coefficients for the optimal filter, and then the performance of the resolve pass scales linearly with the number of samples in the MSAA texture and the color precision. For the standard per-pixel box filter the $8\times$MSAA resolve at 1920×1080 takes 0.53ms, compared to 2.29ms for our optimal filter.

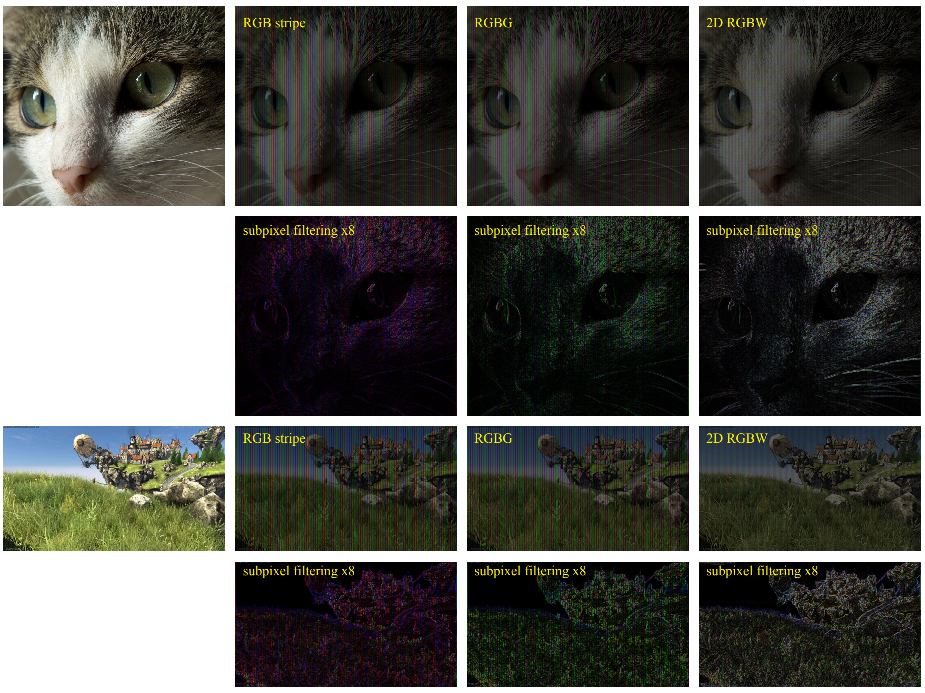

Figure 4.9: Subpixel renderings of a photograph and an image from the UNIGINE game engine benchmark; both images are high resolution and have been injected into an MSAA render target. We show different subpixel arrangements with filtering obtained from our subpixel-aware MSAA resolve. The second and fourth row show the absolute difference ($\times 8$) to images resolved with standard anti-aliasing. The colorful appearance is due to the relative positions of subpixels: the difference image for the RGB stripe pattern is mostly purple as the green subpixels reside on the RGB pixel centers, while red and blue subpixels are offset and thus most affected by the filtering. For the RGBG pattern the green subpixels are shifted, while all subpixels are roughly equally affected with the 2-D RGBW pattern. Image taken from [47]; benchmark image captured from UNIGINE Heaven DX11 Benchmark. UNIGINE Corp. 2012. All rights reserved.

[2]As of this writing, available at `http://cg.ivd.kit.edu/Subpixel.php`.

4.5.3 Texture Filtering

We evaluated our proposed subpixel texture filtering in terms of quality and speed. Fig. 4.1 shows a comparison of different texture filtering techniques. For this example a square texture with a resolution of 6144^2 pixels was used, while the final rendering was 1024^2 pixels in size. We have implemented all filtering methods in pixel shaders. Our implementation of the EWA algorithm is based on the original formulation by Greene [62]. Furthermore, we simulated the different subpixel layouts in a pixel shader for display on a generic liquid crystal display (LCD). As can be seen in the right part of Fig. 4.1, subpixel texture filtering reduces blurriness and increases the sharpness of the texture. In terms of performance, filtering cost mostly scales linearly with the number of required texture look-ups. Since the performance of the filters depends strongly on the resolution of the input texture, we only report relative performance values. For the RGBG layout, we have found that the filtering costs when using our optimal subpixel filtering algorithm increase by a factor of about 1.1 to 1.2. This only moderate gain in rendering time is due to the reduced pixel resolution that the *PenTile* RGBG display offers. For the other subpixel layouts, filtering is 1.8 to 2.2 times more costly than standard filtering.

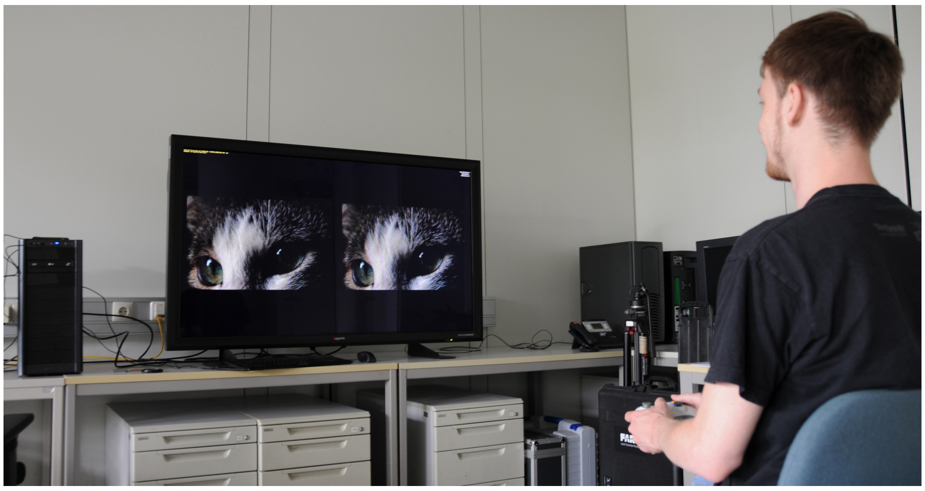

Figure 4.10: Test setup for the user study. *Lighting conditions in the photograph do not match actual testing conditions, where the environment illumination was completely darkened.* Image taken from [47].

4.5.4 User Studies

We conducted a user study to compare the quality of our proposed optimal filter with other filters. To this end, we employed a 56" (142.24 cm) Quad-HD monitor with a resolution of 3840×2160 pixels, i.e., a pixel pitch of 0.32mm. Participants sat 1.8m away from the display, which corresponds to a distance of about 50cm on a regular 24" Full-HD display (Fig. 4.10 shows our setup). The high-resolution display was used to show each of six different reference images (see Fig. 4.11 for an example; the supplemental material of the original publication [47] contains all images used) at the full resolution of the display. We then took grids of 3×3 monitor pixels to simulate a low-resolution display, e.g. for RGB stripe layout, the left column of the

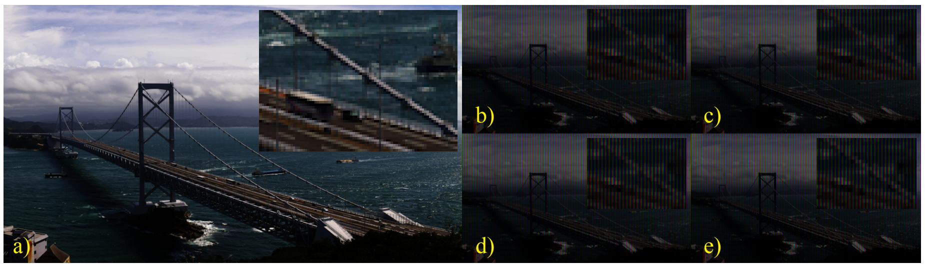

Figure 4.11: Different (subpixel) filters employed in the user study (RGB stripe layout). a) Reference image. b) Mitchell filter, no subpixel rendering. c) Box filter, subpixel rendering. d) Mitchell filter, subpixel rendering. e) Our optimal filter, subpixel rendering. Filtered images are dimmer due to simulating subpixels with real pixels. Although the differences seem negligible in the zoom-ins, they are noticeable when viewing the resulting images on the screen (as supported by our user study). Image taken from [47].

3×3 pixels shows red only, the middle column green, and the right column blue. We used our optimal subpixel filter, a box-shaped subpixel filter (roughly corresponding to ClearType), a subpixel Mitchell filter, and a standard Mitchell filter (no subpixel processing) to create four down-sampled versions (by 1/3 along each axis) of the original images. Both Mitchell filters use parameters $B = C = 1/3$, following the recommendation $B + 2C = 1$ [147].

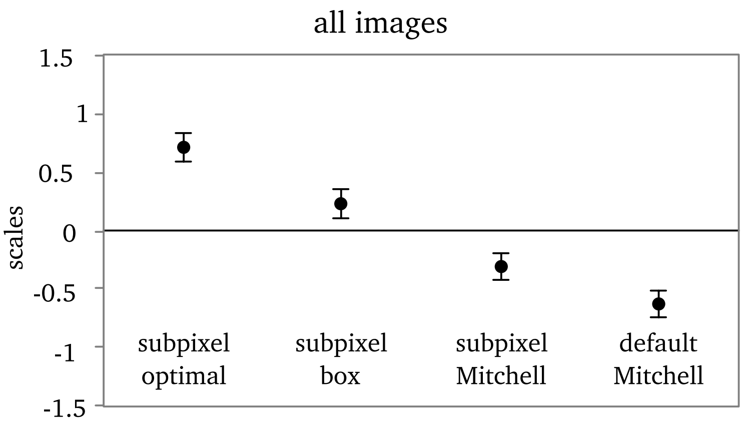

Figure 4.12: User study results for comparing different subpixel filtering methods for six different images (bridge, cat, fairy, hairball, text, and tree). Participants were asked to perform a pairwise comparison between the different methods and choose the best reproduction of a given image. Thurstonian scaling was applied and our optimal filter was significantly better than any other method ($p < 0.01$, t-test). Image redrawn from [47].

Participants of our study performed a pairwise comparison experiment on these images: each stimulus consisted of two pairs of images (enumerating all possible combinations of filters in random order) and participants chose the image that (subjectively) best replicated the high-resolution image. Subjects were allowed to look at the reference, high-resolution image in order to better assess the image quality of filtered results. To this end, they could selectively display the reference, which was displayed in place of the filtered images after a brief (1 sec) pause.

Twenty-two participants took part in the experiment. Fig. 4.12 summarizes the findings (more detail in supplemental material to the original article [47]). Overall, our optimal subpixel filter was significantly better than any other filtering method ($p < 0.01$, t-test). There is a clear benefit to using our optimal filter, even for displaying photographs or game-like content.

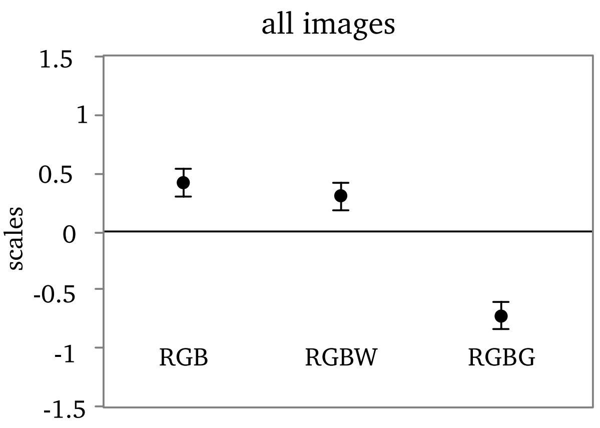

Figure 4.13: User study results for comparing different subpixel patterns for six different images (bridge, cat, fairy, hairball, text, and tree). Participants were asked to perform a pairwise comparison between the different pattern layouts and choose the best reproduction of a given image. There was no statistical significant difference between RGB and RGBW, but RGBG was significantly worse than RGB and RGBW ($p < 0.01$, t-test). Image redrawn from [47].

In a second experiment, we compared different subpixel patterns. The set-up was identical to the first experiment, i.e., using a Quad-HD monitor to simulate three different lower-resolution displays with an RGB, an RGBG, and an RGBW pattern, see Fig. 4.1 for the exact layouts. Please note that for the user study we used slightly different RGBG and RGBW layouts: to ensure a fair comparison of all sub-pixel patterns we simulated all of them with the same number of pixels (6×6) on the physical high-resolution screen. For all three cases, we used our optimal subpixel filter to drive the rendering. Participants again performed a pairwise comparison, where each stimulus compared an image rendered with two different subpixel patterns (enumerating all possible pairs of patterns). This was done for the same six images as above. The same 22 participants took part and the findings are summarized in Fig. 4.13. Overall, RGB and RGBW were not significantly different. However, both were significantly better ($p < 0.01$) than RGBG.

4.6 Conclusion

We have presented methods for subpixel-aware MSAA and texture filtering. To this end, we have analyzed different 1-D and 2-D subpixel patterns, and derived (perceptually) optimal, analytic filters. Our results show that the perceived display resolution increases without noticeable artifacts. The additional cost for subpixel-aware MSAA and texture filtering is small compared to the benefit of using them, so we generally recommend their application to rendering high-contrast, high-frequency geometry.

Returning to our brief discussion in Sec. 3.4, for high-end displays with high pixel densities (above 300 DPI), there is certainly less application for subpixel rendering, and if noticeable, it is mostly for high-contrast parts in images, such as black text on white background. However, display resolutions vary rather dramatically, and a variety of embedded or low-end systems may simply not have the computational capacity to drive such a large number of pixels at once. For typical viewing conditions at resolutions around 100 DPI, at least, our user study indicates a clear advantage to subpixel rendering, while incurring hardly any computational cost. It would be interesting to investigate at what resolutions the benefit of subpixel rendering becomes negligible.

Another direction for future work is combining our low-cost subpixel rendering with temporal integration in the spirit of Didyk et al. and Templin et al. [38, 227], which works best for static images, in order to enhance the display of both static and dynamic images.

Part II

Light Transport

Chapter 5

Physically Based Rendering

The goal of this chapter is to give a self-contained and consistent overview of the notation, concepts, and algorithms of physically based rendering (PBR).[1] The main focus is on those aspects of PBR that are most relevant to our end goal of *artistic editing*; we point the interested reader to the more comprehensive (secondary) literature on the subject, e.g. [1, 19, 25, 42, 59, 85, 95, 153, 154, 177, 185, 219].

Another widely used, largely synonymous term for PBR is "photorealistic rendering," which implies a standard test for image quality: When presenting the end result to a (human) observer, they should be unable to tell the computer-generated image from a real photograph.

Of course, we can also use PBR to create images that *could not have been photographed at all*, and still have observers judge them as realistic, or at least, plausible. It is this last point which explains the wide application of PBR in fields such as visual effects for entertainment, but also in predictive rendering of *things yet to be created in the real world*.

5.1 Radiometry

At the root of PBR, radiometry is an integral component; in the interest of consistent notation throughout this dissertation, we will summarize basic radiometric concepts in this section. Specifically, we will deal with the subject only on a level of detail that is sufficient to later explain the governing equations of physically based rendering.[2]

Radiometry deals with electromagnetic radiation, i.e., with a flow of photons which may be interpreted as either particles or waves carrying energy, depending on the context (see e.g. [1]). For the most part, the wave properties are ignored, together with effects such as polarization, interference and diffraction. That said, a photon's wavelength (and frequency) are of interest even in the geometric model of light, if only to be able to create color images (cf. Sec.3.2). Electromagnetic energy with wavelengths in the interval $[380\text{ nm}, 780\text{ nm}]$ is considered to be visible light.

[1]Parts of this chapter are based on our previous publications [46, 201, 202] and—as before–on the publications given throughout the text.

[2]The term physically *based* rendering already hints that the models simulated in computer graphics are somewhat of a simplification of the models developed in physics, which are, in turn, models of the real world.

We will start with introducing some basic quantities. A photon's wavelength λ [nm] and frequency ν [Hz $= \frac{1}{\mathrm{s}}$] are related by:

$$\lambda \nu = c \left[\mathrm{m} \cdot \mathrm{s}^{-1}\right], \tag{5.1}$$

where c is the photon's speed—the speed of light. A photon's energy is given by Planck's constant $\hbar$ and, either its frequency ν, or its wavelength λ:

$$Q = \hbar \nu = \frac{\hbar c}{\lambda} \, [\mathrm{J}] . \tag{5.2}$$

While we could directly use this for rendering, say, by counting the number of photons arriving at a sensor [229], the simulated particles called "photons" in CG may carry the aggregate energy (and behavior) of many real photons at once (as we will see later in Sec. **??** This also gives more flexibility in shaping arbitrary emission profiles than enumerating how many photons of which wavelength(s) are emitted.

Radiant power, also known as radiant *flux*, is energy per unit time:

$$\Phi(t) = \frac{\mathrm{d}Q(t)}{\mathrm{d}t} \left[\mathrm{J} \cdot \mathrm{s}^{-1} = \mathrm{W}\right] . \tag{5.3}$$

For the types of scenes typically rendered in CG, the assumption is that light travels instantaneously. This leads to usually dropping the time parameter and writing radiant power simply as Φ, with the understanding that there is an implicit time dependence to it. Time is important for animation though; just imagine simulating a television screen in a dark room or a flickering candle as a light source.[3]

Irradiance is incident flux per unit surface area:

$$E(x) = \frac{\mathrm{d}\Phi_i}{\mathrm{d}A(x)} \left[\mathrm{W} \cdot \mathrm{m}^{-2}\right], \tag{5.4}$$

where the surface may be hypothetical if we want to measure irradiance in free space. *Radiosity* $B(x)$ (and its synonym *radiant exitance* $M(x)$) are defined analogously, but for outgoing flux Φ_o, instead of its incident version Φ_i.

Intensity is the density of radiant power with respect to solid angle:

$$I(\omega) = \frac{\Phi}{\mathrm{d}\sigma(\omega)} \left[\mathrm{W} \cdot \mathrm{sr}^{-1}\right] . \tag{5.5}$$

Radiance is power per *projected surface area* per solid angle, or equivalently power per surface area per *projected solid angle*:

$$\begin{aligned} L(x, \omega) &= \frac{\mathrm{d}^2 \Phi}{\mathrm{d}A^{\perp}(x) \, \mathrm{d}\sigma(\omega)} \\ &= \frac{\mathrm{d}^2 \Phi}{\mathrm{d}A(x) \, |\cos\theta| \, \mathrm{d}\sigma(\omega)} \\ &= \frac{\mathrm{d}^2 \Phi}{\mathrm{d}A(x) \, \mathrm{d}\sigma^{\perp}(\omega)} \left[\mathrm{W} \cdot \mathrm{m}^{-2} \cdot \mathrm{sr}^{-1}\right], \end{aligned} \tag{5.6}$$

where we see that the projection manifests itself as (the absolute value of) the cosine of the angle of incidence θ, i.e., the inner product between ray direction ω

[3]Time is also important for more exotic frameworks such as transient rendering [88].

and surface normal N_x. The explanation for this factor is that more grazing incidence angles spread the same differential flux over a wider area of a surface.

As we will see later, even though we are measuring (weighted) radiant flux in PBR, radiance is the prime quantity in light transport "traveling" along rays. Also, when talking about light fields in CG, one usually means radiance as a function of position and direction, often regularly sampled in some subset of its domain.

Despite the definition of radiance (and other quantities given above) being independent of wavelength, in graphics literature it is implicitly assumed that radiance "has a color," usually given as RGB triplets in some color space. This works because for the most part the models used in CG ignore many spectral effects, such as fluorescence or phosphorescence, where light energy of one frequency is re-radiated at another frequency (at different time scales). At the very least, RGB values should be in a linear space.[4]

For "proper" spectral rendering, however, there exists the concept of *Spectral radiance*:

$$L_\lambda(x,\omega,\lambda) = \frac{\mathrm{d}^2\Phi_\lambda(\lambda)}{\mathrm{d}A^\perp(x)\,\mathrm{d}\sigma(\omega)}\left[\mathrm{W}\cdot\mathrm{m}^{-2}\cdot\mathrm{sr}^{-1}\cdot\mathrm{nm}^{-1}\right]. \tag{5.7}$$

Spectral variants of the other quantities are defined analogously. Understanding that different quantities have different units, we will drop units from now on, as is customary in graphics.

Beyond the basic quantities seen so far, there exist various others, such as *fluence*, which is the total incident radiance at a point in free space:[5]

$$\phi(x) = \int_{\mathcal{S}^2} L_i(x,\omega)\,\mathrm{d}\sigma(\omega). \tag{5.8}$$

However, the basic quantities noted above are usually sufficient for the purpose of PBR.

The relationships between the quantities deal with differentials, but they can equivalently be expressed in terms of integrals:

$$\Phi = \int_{\mathcal{M}} E(x)\,\mathrm{d}A(x), \tag{5.9}$$

$$E(x) = \int_{\mathcal{S}^2} L(x,\omega)\,\mathrm{d}\sigma^\perp(\omega), \tag{5.10}$$

$$L(x,\omega) = \int_{\Lambda} L_\lambda(x,\omega,\lambda)\,\mathrm{d}\lambda. \tag{5.11}$$

Indeed, we can arrive at different quantities through a variety of differentiation and integration steps, see Fig. 5.1.

In ending the topic of radiometry, we will briefly mention that there also exists the field of *photometry*, which deals with analogous quantities as radiometry. However,

[4]See Meng et al. [143] for other situations where RGB rendering leads to problems.

[5]Unfortunately, fluence uses the same symbol as flux [177]. Given its relative rarity in CG, this seems not problematic, though.

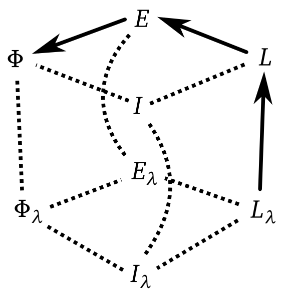

Figure 5.1: Relationship between radiometric quantities. Moving up/left by integration, or down/right by differentiation, we can arrive at different quantities. The path usually taken in physically based rendering from the light fields' (spectral) radiance to arrive at a sensor's incident radiant power is highlighted.

all quantities (a) are weighted by the luminous efficiency curve (which is one and the same as $\bar{y}(\lambda)$ used in Eq. 3.1) and (b) have different names. As an example of its application, photometry is often used to specify the brightness of display devices[6]. While one could do physically based rendering using photometric quantities, it is not a very popular choice in the literature, and we will stay with radiometry for the rest of this thesis.

5.2 Light Sources

Having looked at the basics of radiometry, we will now focus on light sources—idealized models of objects that emit electromagnetic energy. Almost every book on computer graphics discusses this topic [1,59,177].

While volumetric emission from physical processes such as flames has been used as a light source in computer graphics [152,232], the emitted radiance in a scene is normally left as a user-controlled, only *physically informed* model with various parameters. Along the same lines, while black body emission or measured spectra of actual luminaires can be used, having as much control as possible over the lighting is an important requirement.[7]

Ultimately, the goal of a light source model is to describe $L_e(x, \omega)$, the emitted radiance function, in a scene. As with the outgoing radiance function $L(x, \omega)$, there are three aspects to this: (1) where emission happens, (2) into which direction light is emitted and (3) the emission spectrum, which captures both the brightness and the color of the light source. As the case may be, these three aspects are often nicely factored, in order to "mix and match" different functions, but may also be implicit in $L_e(x, \omega)$, which is more general but harder to control.

In practice, most rendering methods require the emitted radiance function to be both evaluated when explicitly sampling points or rays on light sources, say for computing direct illumination, as well as when accidentally hitting a light source. (More on this in Sec. 5.11.) We will give a short list of popular light source models in the following.

[6]This was written in front of a display with a maximum brightness rating of 350 cd/m^2.

[7]As is the case with traditional photography. Remember that images are not created in a vacuum, but towards a specific purpose, artistic or otherwise.

Diffuse *area light* sources are one of the simplest models to integrate into a physically based renderer; they are usually part of an object's material description, stating how much light is uniformly emitted by the surface. Various directional distributions can be combined with this, from constant diffuse emission to emission smoothly varying according to the angle with the surface normal N_x.

Another popular type are *point lights* (more strictly known as omni lights [1]), which emit light uniformly into all directions, from a single point, thus introducing a Dirac distribution:

$$L_e(x,\omega) = \delta(x - x_l) \cdot L'_e, \tag{5.12}$$

where $L'_e = \text{const.}$ may be also be given as the total emitted flux Φ_l or intensity I_l.

Limiting positional and/or directional aspects in some way, including with Dirac distributions, naturally leads to a variety of models for spot lights, directional lights, collimated light (e.g. an idealized laser beam $L^e_\lambda(x,\omega,\lambda) = L'^e_\lambda \cdot \delta(x - x_0) \cdot \delta(\omega - \omega_0) \cdot \delta(\lambda - \lambda_0)$), etc.

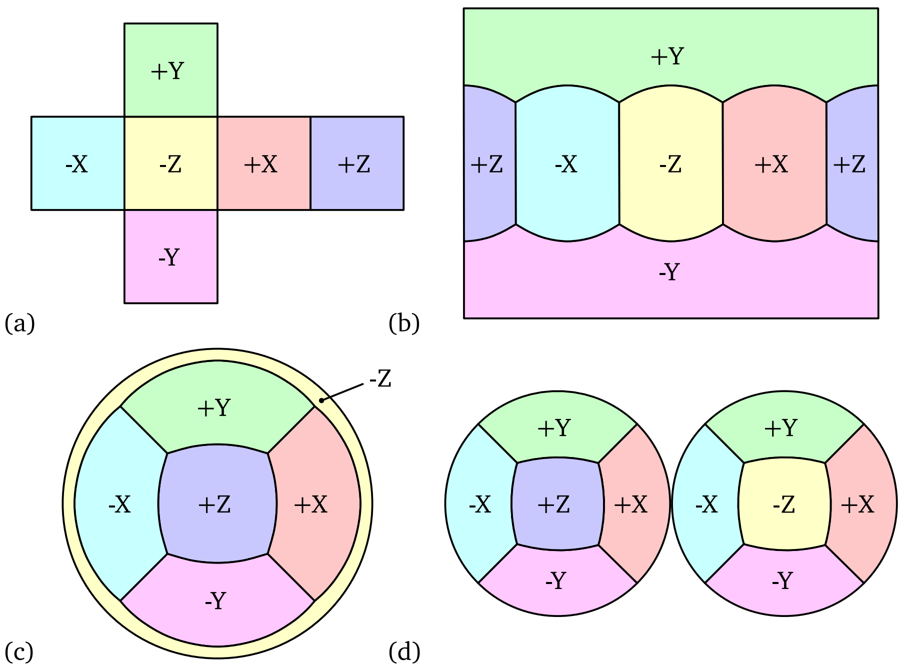

Figure 5.2: Various environment mapping schemes, translating from directions on the sphere $\mathcal{S}^2$ to the two-dimensional image plane: (a) cube map, (b) latitude-longitude map, (c) sphere map, and (d) dual paraboloid map. Labels like "-X" signify directions that are mostly along the negative x axis, etc.

One popular method to achieve realistic lighting is to capture an environments' incident radiance by, e.g. photographing a reflective sphere, using exposure bracketing techniques to capture its full dynamic range [190]. These images, also called *light probes*, can be used for image-based lighting (IBL). Ignoring any positional information, e.g., presupposing the environment is infinitely far away, the directional distribution and spectral intensity of L_e are taken from the image, utilizing one of many *environment mapping* schemes which translate from positions in image space to directions, and vice versa (cf. Fig. 5.2). While not using images, various procedural sun and sky models [77, 78, 156, 182, 241] also fall under this category.

5.3 Surface Geometry

This section gives a short overview of the surface geometry used for PBR; as with other topics, we only briefly touch on the most important topics. We note, however, that surface representations have been a major focus of computational geometry and computer graphics research over many decades, and point to [1, 79] as a starting point for more details.

Historically, there have been many different ways of representing surface geometry,[8] but usually, a given scene's surfaces are built out of simpler objects called *primitives*, a term that has carried over from ray tracing.

Various geometric primitives have been used for rendering, ranging from those given by implicit equations, such as spheres, cylinders, cones, etc., to polygons, parametric and subdivision surfaces, point clouds, voxels, etc., to more exotic representations such as distance fields or fractals.

Due to their relative flexibility, triangle meshes are a widely used common denominator that can represent a variety of surfaces, or rather, a wide variety of surfaces can be tessellated into triangle meshes to desired precision. Due to their popularity, there also exists a wide range of data sets in this format.

Following Veach [229], we will denote the set containing all surfaces in a scene $\mathcal{M}$, hinting at the fact that it is a union of manifolds. (Though it must not be a manifold by itself).[9] We also assume that for every point $x \in \mathcal{M}$, we have an associated unit surface normal N_x (sometimes written as $N(x)$). Surface points may have additional parameters such as a local tangent space, i.e., a normalized vector T_x orthogonal to N_x, and the bitangent vector B_x implicitly given by $T_x \times B_x = N_x$. Surface points may have other attributes such as material parameters, two-dimensional parameterizations (texture coordinates), etc.

Furthermore, surfaces are understood as the bounding interface between two media on either side, which may have different indices of refraction η_i and η_t. For completeness, we denote the volume in between all surfaces as $\mathcal{V}$ and assume it is disjoint from $\mathcal{M}$, i.e., $\mathcal{V} = \mathbb{R}^3 \setminus \mathcal{M}$ and $\mathcal{V} \cap \mathcal{M} = \emptyset$

Many PBR methods ignore the specific boundary representation used and work with a few abstract geometric operators instead, namely *ray tracing* and *surface sampling*. The ray tracing function is not only convenient for describing (ray tracing-based) PBR methods, but is also used in the light transport equation we will encounter later.

Ray Tracing By intersecting a ray, given by its position x and direction ω, with all surfaces in the scene, we may find the closest intersection distance:

$$h(x, \omega) = \min\{s | x + s \cdot \omega \in \mathcal{M}\}. \tag{5.13}$$

If no such point exists, we simply set $h(x, \omega) = \infty$.

[8] Also called boundary representation or simply "the scene."

[9] Naming the scene's surfaces "manifold" $\mathcal{M}$ is also convenient, as the letter $\mathcal{S}$ is already taken up by the unit (hyper)sphere.

The ray tracing function $r(x,\omega)$, can be defined in terms of h, returning the closest intersection point $y = r(x,\omega)$:

$$r(x,\omega) = \begin{cases} x + h(x,\omega)\cdot\omega & \text{if} \quad h(x,\omega) < \infty, \\ \mathcal{X} \quad \text{otherwise} \end{cases} \tag{5.14}$$

where we return a special "point at infinity," $\mathcal{X}$, when there is no intersection along the ray.

While one can test for intersection with all primitives and return the closest point, a critical component of ray tracing is to use intersection *acceleration structures*, which allow to conservatively avoid as many such tests as possible [48, 177]. Acceleration schemes can be classified in various ways:

- hierarchical vs. flat
- space-partitioning vs. object-partitioning
- adaptive vs. fixed

The most commonly used acceleration schemes for ray tracing in PBR are bounding volume hierarchies and k-D trees, though other representations such as grids are also encountered.

Other operations can be formulated in terms of ray tracing; for example, the (binary) visibility between two surface points $x \in \mathcal{M}$ and $y \in \mathcal{M}$ may be evaluated in terms of the ray tracing function:

$$V(x,y) = \begin{cases} 1 & \text{if } r(x, y-x) = y, \\ 0 & \text{otherwise.} \end{cases} \tag{5.15}$$

Depending on the situation, other cached representations of visibility information such as shadow maps [242] may be more beneficial, but from the perspective of light transport, $V(x,y)$ is mostly an abstract operation defined as above.

Surface Sampling Many rendering methods require sampling surface points according to some probability density $p_A(x)$. As with ray tracing, while an implementation of sampling surfaces actually depends on he boundary representation used, light transport may see this as an abstract operation to be performed on some surface $\mathcal{M}_i \subseteq \mathcal{M}$, returning a point x, and optionally its density $p(x)$. We note that we may want to sample other surfaces such as the unit (hemi-)sphere, or unit discs in PBR as well, which are not part of the scene per se.

5.4 Materials and the BRDF/BSDF

Visually, the perception of an object's material largely depends on the way its surface *reflects* light. The standard way of describing this is the *bidirectional reflectance distribution function* (BRDF), which gives the ratio of exitant differential radiance to incident irradiance (see Fig. 5.3 for the geometrical configuration involved):

$$f_r(x,\omega_o,\omega_i) = \frac{\mathrm{d}L(x,\omega_o)}{\mathrm{d}E(x)}. \tag{5.16}$$

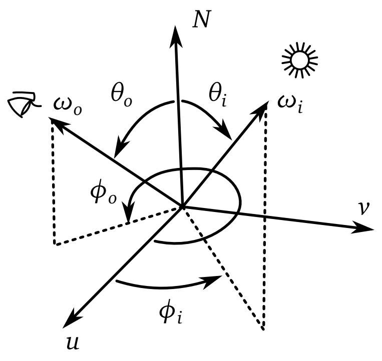

Figure 5.3: Geometry for local surface reflection.

This existence of the BRDF is a direct consequence of the linearity assumption of light transport, which requires that $\mathrm{d}L(x,\omega) \propto \mathrm{d}E(x)$ [177].

When we are interested in surfaces that transmit light, this is described by a *bidirectional transmittance distribution function* (BTDF), which captures the same idea as the BRDF, but the incident and exitant directions are in opposite hemispheres.

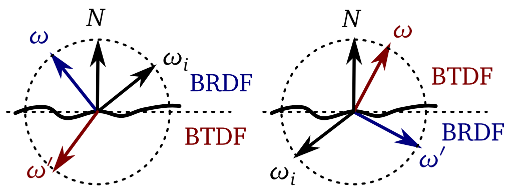

Figure 5.4: The bidirectional (surface) scattering distribution (BSDF) implicitly defines two reflectance and transmittance distributions (BRDFs and BTDFs), depending on which side of the hemisphere around surface normal N the incident direction ω_i is located. This greatly simplifies handling materials that both reflect and transmit light, which—due to the Fresnel effect–is quite common in transparent and translucent objects.

In general, we end up with four different distributions at a single point, which becomes a bit unwieldy. The *bidirectional scattering distribution function* (BSDF) is a more compact representation, and can be interpreted as two BRDFs and two BTDFs defined on different hemispheres (see Fig. 5.4). Conversely, if all objects are considered opaque and only surface reflection, but not transmission, is of interest, a single BRDF f_r can be “adapted” into a BSDF $f_s(\omega_o,\omega_i)$ that is zero except where the incident and exitant directions are in the same hemisphere as the surface normal, i.e., $N\cdot\omega_o \geq 0 \wedge N\cdot\omega_i \geq 0$. Note that the integration domain for BSDFs and BRDFs is different, hence care must be taken when sampling directions or normalizing PDFs.

Again following our black-box approach to light transport, the most important features of BRDFs/BSDFs are the ability to (a) evaluate them for given directions, and (b) sample incident or exitant directions when holding other parameters fixed. We do note, however, that materials and especially surface reflection models constitute a large part of computer graphics research, and one of the key changes during the adoption of physically based rendering was the introduction of physically valid reflection models.

To maintain physical correctness, a BRDF f_r must uphold the two constraints of non-negativity and energy conservation. Non-negativity simply means that: $\forall \omega_o, \omega_i \in \mathcal{H}_+^2 : f_r(\omega_o, \omega_I) \geq 0$, for all inputs. Energy conservation means that the BRDF cannot add energy to the system through reflection, i.e.:

$$\int_{\mathcal{H}_+^2} f_r(\omega_o, \omega_i)\,\mathrm{d}\omega_i^\perp \leq 1, \forall \omega_o \in \mathcal{H}_+^2. \tag{5.17}$$

Of the constraints mentioned, energy conservation is arguably the most important in the context of PBR, since it ensures that the solution method can actually converge,[10] and images will only over-saturate (even in the presence of indirect illumination) if the lighting is adjusted to attain such an effect, which allows lighting and material artists to predictably control the scale of tones and colors in a final image (cf. Figs. 7.1(c)/7.2(c)).

Reciprocity, given by $f_r(\omega_o, \omega_I) = f_r(\omega_i, \omega_o)$, is another property of physically based BRDF models that helps maintain correct images with bidirectional techniques; consistent use of adjoint BRDFs sidesteps this issue, as we will discuss in Sec. 5.7.

A common approach to derive physically correct BRDFs is to mix and match different components [1]. As an example, the Torrance-Sparrow model, one of the oldest physically based reflection models in CG [1, 177], is based on a statistical model of specular microgeometry and has several factors accounting for effects such as Fresnel reflectance, normal distribution of microfacets, and additional terms accounting for masking and shadowing effects:

$$f_s(x, \omega_o, \omega_i) = \frac{D(\omega_i)G(\omega_o, \omega_i)F_r(\omega_o)}{4\cos\theta_i \cos\theta_o}, \tag{5.18}$$

where $\omega_h = \frac{\omega_i + \omega_o}{\|\omega_i + \omega_o\|}$ is known as the *half-angle vector*.[11]

The spatial distribution of materials is usually a factored representation, i.e., one applies a single, parameterized BRDF model to a surface, but allows *texturing* to change the parameters spatially. Strictly speaking, the BRDF is only defined at a single point, e.g. $f_r(\omega_o, \omega_i)$; if the distribution also depends on the surface point x, it may be more correctly be called bidirectional texture function (BTF) or spatially varying BRDF (SVBRDF). Blurring this distinction is common in CG.

From a practical perspective, a consistent terminology for BRDF parameters based on radiometry should be used to avoid errors: The "diffuse color" ρ_d commonly found in older Lambertian shading models can be concretely and intuitively described as the *reflectivity* or *albedo*, i.e., the ratio of light directly reflected by a perfectly diffuse surface lit from a uniform distant white lighting environment.

This radiometrically consistent terminology can also lead to additional insights about the behavior of shading and reflectance models for practitioners, such as when realizing that energy conservation and (the non-physical) *ambient occlusion* ρ_x (i.e., unit incident radiance $L_i = 1$) are related by the same equation:

$$\rho_x = \int_{\mathcal{H}_+^2} 1 \cdot \frac{\rho_x}{\pi}\,\mathrm{d}\omega_i^\perp \quad (\leq 1)\,. \tag{5.19}$$

[10]As first noted in the radiosity context (Sec. 5.11.1).

[11]ω_h is related to perfect specular reflection, and shows up in other light transport-related contexts [70, 87].

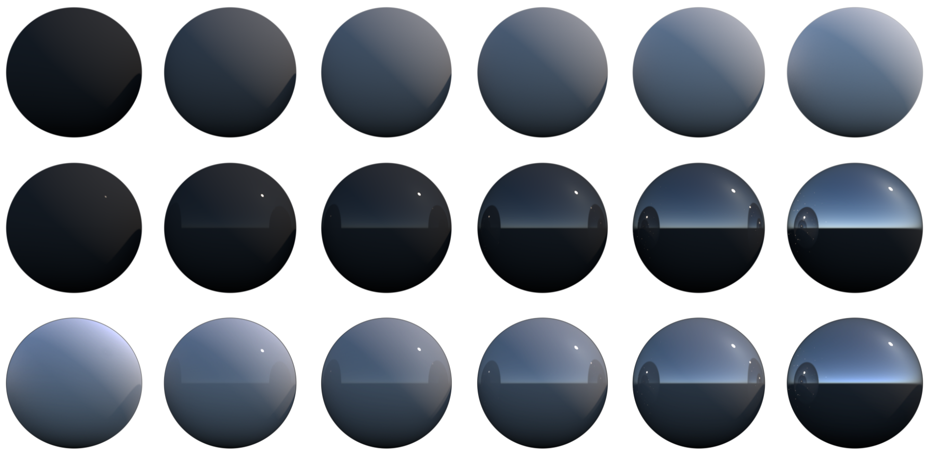

Figure 5.5: Controlling the amount of diffuse and specular reflection. If the underlying BRDFs are energy conserving, one can predictably control the amount of diffusely (top row) and specularly (center row) reflected light across different lighting scenarios by adjusting diffuse albedo ρ_d and specular reflectivity ρ_s. Bottom: Energy-conserving combinations of the two BRDF models are maintained while linearly interpolating between perfect diffuse reflection ($\rho_d = 1, \rho_s = 0$) and perfect mirror reflection ($\rho_d = 0, \rho_s = 1$). Image taken from [202].

Returning to the topic of designing BRDFs by mixing and matching components mentioned above, radiometrically relevant parameters also trivially lead to a sound *combination* of BRDFs, including for materials with spatially-varying parameters. If each BRDF is energy-conserving and the relationship between parameters and energy conservation is well understood, new materials can be authored by combining existing base BRDF models. Regarding the example in Fig. 5.5, the diffuse albedo ρ_d, which must take on values between 0 and 1, by definition, and a perfect mirror BRDF's similar parameter ρ_s, i.e., the amount of perfectly reflected light along the mirror direction are easy to combine. Since each model is independently energy conserving, then a composite BRDF $f_r = f_d + f_s$, will also respect energy conservation if $\rho_d + \rho_s \leq 1$, i.e., any convex combination of parameters. This generalizes to more complex combinations of more complex energy-conserving BRDFs

Parameterizing the reflectance behavior of a BRDF, i.e., the shape of its lobes, with energy conservation in mind allows one to predictably adjust parameters such as the glossiness of a material without having to worry about oversaturation or intensity peaks in final renders. On a more advanced level, one may, e.g., relate microfacet distribution parameters and phenomenological glossiness factors [84], and then allow exchanging BRDF models in a controllable fashion (cf. Fig. 5.6).

The advantages of radiometrically valid surface shading models we have just outlined has lead studios to adopt more "principled" BRDFs [15], spanning a wide range of materials including metals/conductors and dielectrics/insulators with only a handful of parameters.[12]

[12]Developments such as these were part of the move to *physically based shading* (PBS), which predated the PBR trend in part, and also affected real-time rendering.

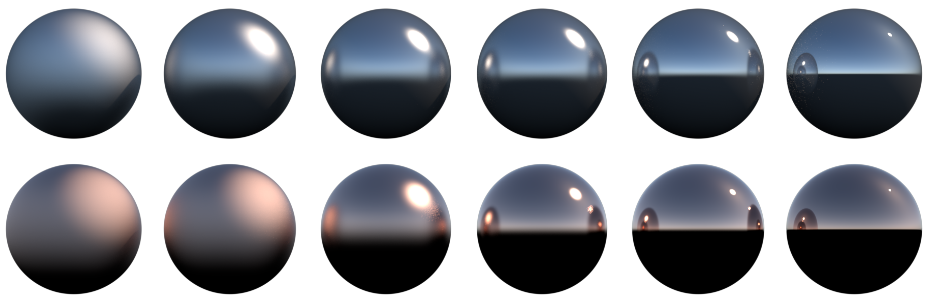
Figure 5.6: One can relate different BRDFs' parameters to one another, such that replacing BRDF models during authoring maintains a similar appearance of highlights and blur. Top row: Varying "glossiness" in a simple metallic BRDF model. Bottom row: Corresponding microfacet roughness of a more physically based BRDF model. Image taken from [202].

5.5 The Light Transport Equation

In this section, we will describe the surface form of the *light transport equation* (LTE) , which describes how light travels in a scene, under the assumptions of energy balance, non-interaction of photons, instantaneous transfer (e.g., time-independence), and the absence of participating media. The equation can be written in various forms and was introduced to the graphics community, under the name *rendering equation* by James Kajiya in 1986 [100]; our notation follows [59, 177, 229].

Making use of the concepts we have discussed in this chapter so far, the LTE states conditions for the equilibrium radiance (cf. Sec 5.1) in a scene, by the following mutually recursive equations:

$$L_i(x,\omega) = L(r(x,\omega),-\omega)), \tag{5.20}$$

$$L(x,\omega) = L_e(x,\omega) + L_s(x,\omega), \tag{5.21}$$

$$L_s(x,\omega) = \int\limits_{\mathcal{S}^2} f_s(x,\omega,\omega_i) L_i(x,\omega_i)\,\mathrm{d}\omega_i^{\perp}. \tag{5.22}$$

Eq. 5.20 states that incident radiance L_i arriving at a point x from direction ω, i.e., a ray (x,ω), is equal to the outgoing radiance L emanating from the closest surface point $r(x,\omega)$ along the ray, in opposite direction $-\omega$, as radiance along an unoccluded ray is constant in a vacuum. Eq. 5.21 states that outgoing radiance emanating from a point equals the self-emitted radiance L_e plus the locally scattered radiance L_s. Eq. 5.22, finally, states that the locally scattered radiance is equal to the incident radiance, convolved with the BSDF f_s (cf. Sec. 5.4) over the sphere $\mathcal{S}^2$ of directions around x, measured using projected solid angle.

We note that this system of equations is recursive. Various other formulations are possible, such as substituting L_i and L_s to arrive at a single equation:

$$L(x,\omega) = L_e(x,\omega) + \int\limits_{\mathcal{S}^2} f_s(x,\omega,\omega_i) L(r(x,\omega_i),-\omega_i)\,\mathrm{d}\omega_i^{\perp}, \tag{5.23}$$

which formally is a—still recursive—Fredholm equation of the second kind. In a scene with only opaque surfaces, we could have used the BRDF f_r instead of the BSDF f_s:

$$L(x,\omega) = L_e(x,\omega) + \int\limits_{\mathcal{H}_+^2(x)} f_r(x,\omega,\omega_i) L(r(x,\omega_i),-\omega_i)\,\mathrm{d}\omega_i^\perp, \tag{5.24}$$

which, due to the now position-dependent, hemispherical integration domain $\mathcal{H}_+^2(x)$, formally is a Volterra equation of the second kind. We also could have used only incident radiance and the solid angle measure:

$$L_i(x,\omega) = L_e(r(x,\omega),-\omega) + \int\limits_{\mathcal{S}^2} f_s(r(x,\omega),-\omega,\omega_i) L_i(r(x,\omega),-\omega_i) \left|cos\theta_i\right| \mathrm{d}\omega_i \tag{5.25}$$

All the formulations so far are largely equivalent, but useful in different contexts. Other formulations are possible to additionally include spectral rendering, time-dependence, phosphorescence, fluorescence, transient rendering, etc.

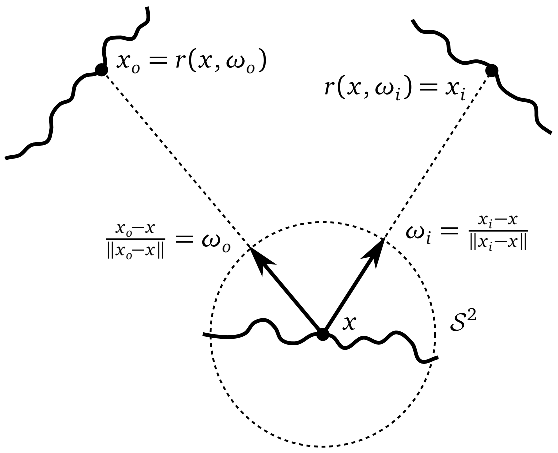
$x_o = r(x,\omega_o)$
$r(x,\omega_i) = x_i$
$\frac{x_o-x}{\|x_o-x\|} = \omega_o$
$\omega_i = \frac{x_i-x}{\|x_i-x\|}$
x
$\mathcal{S}^2$

Figure 5.7: Geometry for the light transport equation in three-point form. (Eq. 5.29).

The light transport equation(s) we have seen so far are in *directional form*. For a formulation in *three-point form*, we will need some additional notation for radiance functions and BSDFs, namely (see Fig. 5.7 for the geometry involved):

$$f_s(x_i \to x \to x_o) := f_s\left(x, \frac{x_o - x}{\|x_o - x\|}, \frac{x_i - x}{\|x_i - x\|}\right), \tag{5.26}$$

$$L(x \to x_o) := L\left(x, \frac{x_o - x}{\|x_o - x\|}\right), \tag{5.27}$$

$$L(x \leftarrow x_i) := L_i\left(x, \frac{x_i - x}{\|x_i - x\|}\right). \tag{5.28}$$

With this, we can write the LTE as:

$$L(x \to x_o) = L_e(x \to x_o) + \int\limits_{\mathcal{M}(x)} f_s(x_i \to x \to x_o) L(x \leftarrow x_i) G(x, x_i)\, \mathrm{d}x_i, \qquad (5.29)$$

where we have changed the integration domain from the sphere of directions $\mathcal{S}^2$ to $\mathcal{M}(x)$, the set of surfaces *visible from* x. As this domain is a bit unwieldy, one usually introduces the binary visibility function $V(x, y)$, which is 1 if x and y are mutually visible, i.e., a ray shot from one point to the other does not intersect any other surface, and 0 otherwise (cf. Sec. 5.3). This leads to the final form:

$$L(x \to x_o) = L_e(x \to x_o) + L_s(x \to x_o), \qquad (5.30)$$

$$L_s(x \to x_o) = \int\limits_{\mathcal{M}} f_s(x_i \to x \to x_o) L(x \leftarrow x_i) G(x, x_i) V(x, x_i)\, \mathrm{d}x_i. \qquad (5.31)$$

5.6 The Measurement Equation

We can render images by simply solving the light transport equation (LTE) for rays "seen through a pixel;" however, having a solid model of the image measurement process (1) more closely matches the behavior of actual imaging systems, (2) allows for taking other abstract measurements in a virtual scene, such as one would do with a light meter in real life, and (3) simplifies the theory of equivalent dual transport in the next section, which forms the basis of advanced rendering methods.[13]

The measurement equation formalizes the integral we have to solve for a single pixel's final value, I_j:

$$I_j = \int\limits_{\mathcal{M}} \int\limits_{\mathcal{S}^2} W_e^{(j)}(x, \omega) L_i(x, \omega)\, \mathrm{d}\omega^{\perp}\, \mathrm{d}x, \qquad (5.32)$$

which captures the fact that we collect light incoming from all directions and over the entire area of a (single-pixel) sensor. We will encounter various other formulations of Eq. 5.32 in the following sections.

If we assume that photographic cameras capture light fields, then all we need to do to render photorealistic images—images that could be reasonably assumed to have been taken in the real world—is to sample light fields at appropriate locations. This is implicitly modeled with the term $W_e^{(j)}(x, \omega)$, called *emitted importance*,

Different camera models have been suggested for use in computer graphics, ranging from the ubiquitous pinhole model [1], to thin-lens models [29, 177, 219], to the more or less accurate simulation of actual lens systems [69, 80, 120, 204, 217]. Other aspects of imaging systems such as shutter type and speed, or aperture shapes may also be modeled, and indeed are available in commercial renderers [178].

For the purpose of light transport, however, we will treat the emitted quantity $W_e^{(j)}(x, \omega)$ as another *black box*, similar to what was done with light sources in Sec. 5.2. Note that technically, we have one measurement equation per pixel.

[13]The sloppy treatment of this topic in many physically based renderers is a bit unfortunate; in contrast to the light transport ("rendering") equation, the measurement equations is often ignored altogether, making it hard to compare images generated by different renderers accurately, let alone implement *artistic editing of light transport by moving paths across the image plane*.

5.7 Importance and Dual Transport

By purporting another transport quantity, *importance*,[14] we arrive at an equivalent view of measurement (cf. Sec. 5.6) that is (1) more useful for forward methods such as light tracing, (2) forms the basis of bidirectional methods such as bidirectional path tracing, and (3) also lays the groundwork for the more general path integral formulation of light transport.

Analogous to radiance, we may write recursive equations for importance:

$$W_i(x,\omega) = W_o(r(x,\omega), -\omega), \tag{5.33}$$
$$W_o(x,\omega) = W_e(x,\omega) + W_s(x,\omega), \tag{5.34}$$
$$W_s(x,\omega) = \int_{S^2} f_s^*(x,\omega,\omega_i) W_i(x,\omega_i)\, \mathrm{d}\omega_i^\perp, \tag{5.35}$$

where we have taken Eqs. 5.20, 5.21 and 5.22, replaced radiance L by *importance* W, and the bidirectional scattering distribution function (BSDF) f_s by its adjoint f_s^*:

$$\frac{f_s^*(\omega_o,\omega_i)}{\eta_o^2} = \frac{f_s(\omega_o \leftarrow \omega_i)}{\eta_i^2} = \frac{f_s(\omega_o \rightarrow \omega_i)}{\eta_i^2} = \frac{f_s(\omega_i,\omega_o)}{\eta_i^2} \tag{5.36}$$
$$f_s^*(\omega_o,\omega_i) = \frac{\eta_i^2}{\eta_o^2} f_s(\omega_o,\omega_i) \tag{5.37}$$

where η_i and η_o are the indices of refraction of the (non-participating) media on the incident and outgoing side of the interface formed at the surface.

This leads to an alternative measurement equation:

$$I_j = \int_{\mathcal{M}} \int_{S^2} W_i^{(j)}(x,\omega) L_e(x,\omega)\, \mathrm{d}\omega^\perp \, \mathrm{d}x \tag{5.38}$$

As with the LTE, there are various equivalent forms of writing the importance transport equation, which we may casually use later without enumeration here.

5.8 Path Integral Formulation

This section will summarize the path integral formulation of light transport, an elegant framework for writing the measurement equation (Eq. 5.32), which will lead to the important concept of *path space*. The derivation loosely follows Veach [229] and Pharr and Humphreys [177].

We start with the measurement equation, in surface form:

$$I_j = \int_{\mathcal{M}} \int_{\mathcal{M}} L(x_0 \rightarrow x_1) W^{(j)}(x_0 \leftarrow x_1)\, \mathrm{d}x_0 \, \mathrm{d}x_1, \tag{5.39}$$

and recursively expand radiance using Eq. 5.30:

[14]Variously known as importance, sensitivity, importons, etc. in the literature.

$$\begin{aligned} I_j = &\int_{\mathcal{M}}\int_{\mathcal{M}} L_e(x_0 \to x_1)G(x_0,x_1)W^{(j)}(x_0 \leftarrow x_1)\,\mathrm{d}x_1\,\mathrm{d}x_0 \\ + &\int_{\mathcal{M}}\int_{\mathcal{M}}\int_{\mathcal{M}} L_e(x_0 \to x_1)G(x_0,x_1)f_s(x_0 \to x_1 \to x_2)G(x_1,x_2)\cdot \\ &\qquad W^{(j)}(x_1 \leftarrow x_2)\,\mathrm{d}x_2\,\mathrm{d}x_1\,\mathrm{d}x_0 + \dots, \end{aligned} \tag{5.40}$$

leading to:

$$\begin{aligned} I_j = &\sum_{k=1}^{\infty}\int_{\mathcal{M}}\dots\int_{\mathcal{M}} L_e(x_0 \to x_1)\,G(x_0,x_1)\cdot \\ &\left(\prod_{i=2}^{k} f_s(x_{i-2} \to x_{i-1} \to x_i)\,G(x_{i-1},x_i)\right)\cdot \\ &\qquad W_e(x_{k-1} \to x_k)\,\mathrm{d}x_k \cdots \mathrm{d}x_0 \end{aligned} \tag{5.41}$$

We can make the fact that we are summing the contribution of paths explicit by writing:

$$I_j = \sum_{k=1}^{\infty}\int_{\Omega_k} f(\bar{x})\,\mathrm{d}\mu_k(\bar{x}) \tag{5.42}$$

where $f(\bar{x})$ is the *measurement contribution function* and $\mu_k(\bar{x}) = \prod_{i=0}^{k} A(x_i)$ is the *product area measure* of a path $\bar{x} = x_0 \dots x_k$ of length k. The set of all paths of length k is denoted Ω_k.

The final step is to define *path space*, i.e. the infinite-dimensional space of all possible light paths, as:

$$\Omega = \bigcup_{k=1}^{\infty} \Omega_k, \tag{5.43}$$

i.e., the disjunct union of all possible finite paths, such that we get rid of the summation over path lengths.[15] We thus arrive at the *path integral formulation of the measurement equation*:

$$I_j = \int_{\Omega} f(\bar{x})\,\mathrm{d}\mu(\bar{x}) \tag{5.44}$$

On a high level, this formulation simplifies the problem of rendering to sampling light paths in some fashion and evaluating their contribution. Particularly, the sampling strategy may sample path vertices both locally and globally, and start at the camera, at the light sources, somewhere in the middle, etc.

The reformulation of light transport as an integral over an abstract space of paths is a central contribution of Veach, which has shaped the field and laid the

[15]Which implies that the PDF of a path also carries the probability of sampling its length k.

groundwork for many rendering methods in the time since its inception. Note that the derivation of path space can be extended to participating media, which we will cover in the next section; see, e.g., Jakob's dissertation [85] for details.

5.9 Participating Media

A wide variety of visual phenomena are caused by *participating media*, ranging from large-scale phenomena such as haze, clouds and smoke, to short-range effects such as the scattering of light in marble or skin. The familiarity of these effects to human observers makes them a vital component for realistic rendering.

Essentially, in participating media, light-transporting particles are allowed to interact with matter distributed in between surfaces, as opposed to the vacuum we have presumed so far. While this could be modeled with a lot of tiny surfaces, e.g. dust particles, we are more interested in the aggregate behavior of media, which means a statistical model of the behavior of light. One such model is given by the radiative transfer equation (RTE) [19]:[16]

$$\begin{aligned}(\omega \cdot \nabla)L(x,\omega) = {}& \epsilon(x,\omega) \\ & - \sigma_a(x)L(x,\omega) \\ & - \sigma_s(x)L(x,\omega) \\ & + \sigma_s(x) \int_{S^2} f_p(x,\omega,\omega_i)L(x,-\omega_i)\,\mathrm{d}\omega_i\end{aligned} \tag{5.45}$$

Broadly speaking, the RTE states that the change in radiance $(\omega \cdot \nabla)L$ along a ray (x,ω) consists of four terms (in the order shown in Eq. 5.45): (a) volumetric emission, (b) absorption, (c) out-scattering, and (d) in-scattering.

Compared to the surface LTE, additional components include:

Phase Function The phase function $f_p(.)$ can be thought of as an analogue to the BSDF we encountered earlier, i.e., this function models the local scattering behavior of (volumetric) materials. However, the phase function is normalized such that the integral over the entire sphere, while holding one direction fixed, is one. Thus, the phase function is a probability density function, giving the likelihood of an incoming particle scattering in a specific outgoing direction.

Scattering by the constant phase function $f_p(.) = \frac{1}{4\pi}$ is called *isotropic*, otherwise (by non-constant phase functions) *anisotropic*.[17] Phase functions found in the literature, such as the Henyey-Greenstein phase function, are often parameterized by deflection angle from incident to outgoing direction, rather than by direction vectors. Incident and/or outgoing direction vectors may also be reversed; for consistency's sake, we will follow the conventions we use with BRDFs/BSDFs, meaning ω_o and ω_i *always point away* from a scattering event in our notation.

[16]We follow the notation common in computer graphics [177].

[17]We briefly note that, e.g., Jakob et al. [86] model this differently, to arrive at "even more anisotropic" media.

Absorption and Scattering The *absorption coefficient* $\sigma_a(x) \geq 0$ models light transported along rays being randomly absorbed by the medium. Its reciprocal gives the average expected distance until such an event happens. Analogously, the *scattering coefficient* $\sigma_s(x) \geq 0$ models the light particle colliding with matter in the medium and being scattered (according to the phase function). Their sum $\sigma_t(x) = \sigma_a(x) + \sigma_s(x)$ is called the *extinction coefficient* and models both outscattering and absorption at the same time. Media where both $\sigma_a(x) = \text{const.}$ and $\sigma_s(x) = \text{const.}$ are called *homogeneous*, otherwise *heterogeneous*.

Returning to the RTE, note that the unknown radiance $L(.)$ appears both in a differentiation (left-hand side) and as an integrand (last term), hence it is an integro-differential equation. In order to solve it using, say, Monte Carlo integration, it must be reformulated as a pure integral equation, which can be done using the surface LTE as a boundary condition. [85].

In the interest of brevity, we will skip reproducing the derivation here, directly giving the resulting volumetric version of the light transport equation (LTE):[18]

$$L(x,\omega) = \begin{cases} L_e(x,\omega) + L_s(x,\omega) & \text{if } x \in \mathcal{M} \\ L_i(x,-\omega) & \text{if } x \in \mathcal{V} \end{cases} \tag{5.46}$$

$$L_s(x,\omega) = \int\limits_{\mathcal{S}^2} f_s(x,\omega,\omega_i) L_i(x,\omega_i)\,\mathrm{d}\omega_i^\perp \tag{5.47}$$

$$\begin{aligned} L_i(x,\omega) = \tau(x,x_0)\,L(r(x,\omega),-\omega) + \\ \int\limits_x^{r(x,\omega)} \tau(x,x_i)\Big[\epsilon(x_i,-\omega) + L_p(x_i,-\omega)\Big]\mathrm{d}x_i \end{aligned} \tag{5.48}$$

$$L_p(x,\omega) = \sigma_s(x) \int\limits_{\mathcal{S}^2} f_p(x,-\omega,\omega_i) L_i(x,\omega_i)\,\mathrm{d}\omega_i \tag{5.49}$$

The term $\tau(x,y)$, transmittance, is given by:[19]

$$\tau(x,y) = \exp\left(-\int_y^x \sigma_t(z)\,\mathrm{d}z\right). \tag{5.50}$$

We can think of this as a continuous version of the binary visibility function $V(x,y)$ (Eq. 5.15), which drops from one to zero in a smooth—for homogeneous media, exponential—fashion, as opposed to the harsh drop in binary visibility.

We note that as the volume LTE (Eqs. 5.46–5.49) contains the surface LTE as a boundary condition, it is identical to Eq. 5.21 if there is no participating medium (i.e., $\sigma_t(x) = 0$ everywhere). (Also, Eq. 5.47 is identical to Eq. 5.22.)

The volumetric LTE can also be written in a more compact form by introducing generalized functions. For example, the generalized scattering function f is given by:

[18]In analogy to Sec. 5.5, this may be called the "volumetric rendering equation."

[19]There is some confusion over the terms *transmittance*, *optical thickness*, and *attenuation* in the (CG) literature. More consistently, we should use $T = e^{-\tau}$ for transmittance, and $\tau = \int_y^x \sigma_t(z)\,\mathrm{d}z$, for optical thickness [177], but simply follow the lead of others here.

$$\hat{f}(x,\omega,\omega_i) = \begin{cases} f_p(x,\omega,\omega_i)\,\sigma_s(x) & \text{if } x \in \mathcal{V} \\ f_s(x,\omega,\omega_i) & \text{if } x \in \mathcal{M} \end{cases} \tag{5.51}$$

Along the same lines, one may introduce a generalized geometry factor $\hat{G}$, visibility function $\hat{V}$, and emission $\hat{L}_e$:[20]

$$\hat{G}(x,y) = \frac{D_x(y)\,D_y(x)}{\|x-y\|^2}, \tag{5.52}$$

$$\hat{V}(x,y) = \tau(x,y)\,V(x,y), \tag{5.53}$$

$$D_x(y) = \begin{cases} 1 & \text{if } x \in \mathcal{V}, \\ \left|N_x \cdot \frac{y-x}{\|y-x\|}\right| & \text{if } x \in \mathcal{M}, \end{cases} \tag{5.54}$$

$$D_y(x) = \begin{cases} 1 & \text{if } y \in \mathcal{V}, \\ \left|N_y \cdot \frac{x-y}{\|x-y\|}\right| & \text{if } y \in \mathcal{M}. \end{cases} \tag{5.55}$$

$$\hat{L}_e(x,\omega) = \begin{cases} \varepsilon(x,\omega) & \text{if } y \in \mathcal{V}, \\ L_e(x,\omega) & \text{if } y \in \mathcal{M}. \end{cases} \tag{5.56}$$

With this, we arrive at a (slightly) simpler form of the volumetric LTE (cf. Fig. 5.8):

$$L_i(x,\omega) = \underbrace{\int_x^{x_0} \tau(x_i,x)L(x_i,-\omega)\,\mathrm{d}x_i}_{\text{volume transport}} + \underbrace{\tau(x_0,x)L(x_0,-\omega)}_{\text{surface transport}}, \tag{5.57}$$

$$L(x,\omega) = \hat{L}_e(x,\omega) + \int_{\mathcal{S}^2} \hat{f}(x,\omega,\omega_i)L_i(x,\omega_i)D_x(\omega_i)\,\mathrm{d}\omega_i. \tag{5.58}$$

where we use $x_0 = r(x,\omega)$ as a shortcut for the closest surface intersection. Analogously to the surface LTE, one could derive a three-point form that would then naturally lead to path space. We will refrain from doing so here and refer, e.g., to Jakob [85].

We note that participating media have been extensively studied in computer graphics, and overviews are given by [17, 170, 183]. We will give a short overview of some works here, but defer enumeration of most related work to talking about specific PBR *methods* (not concepts) in Sec. 5.11.

Early work in computer graphics on rendering participating media was based on tracing rays [102, 195] or paths [169], which is general and leads to unbiased results. Other research has focused on overcoming the high dimensionality of light transport through diffusion theory [97, 216] or discrete ordinates methods [50]. While these approximate methods work for dense media, they are not as suitable for smoke or clouds. Simplifications such as *single scattering* media are only valid for a low-albedo medium, but can be efficiently solved with subsampling [45] or line space gathering [223]. Other works have focused on acceleration through caching and density estimation [90, 94, 96], analytic integration [171, 220], or hierarchical evaluation [7].

[20]Note that $\hat{L}_e$ is merely a convenient shortcut, but does not make sense radiometrically, as it would have to both have the unit $\left[\mathrm{W}\cdot\mathrm{m}^{-3}\right]$ and $\left[\mathrm{W}\cdot\mathrm{m}^{-2}\right]$ at the same time!

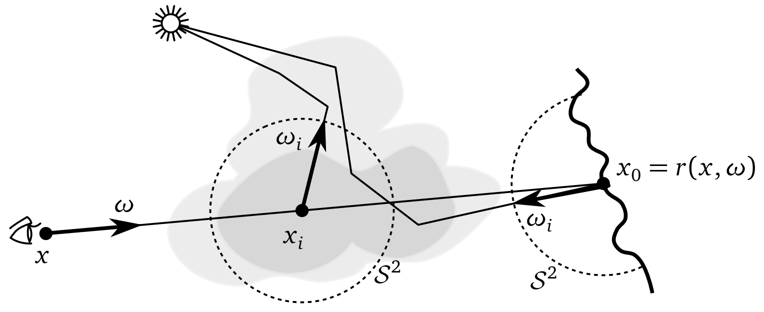

Figure 5.8: Light transport in participating media. The radiance seen along a ray consists of radiance in-scattered in the volume and background radiance from the nearest surface. Both are attenuated by the medium.

Most of the visual realism of rendering scenes with participating media is caused by multiple scattering, i.e., a large number of interaction inside the medium. Its simulation is generally very costly, as scattering may happen practically everywhere in space: Efficiently rendering general, dense, heterogeneous, and anisotropic participating media remains a challenge and an open research area for PBR.

5.10 Subsurface Scattering and the BSSRDF

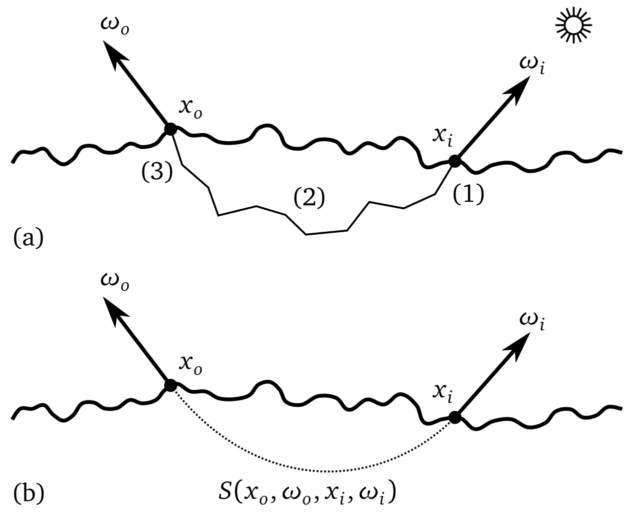

Figure 5.9: (a) Subsurface scattering is the result of light (1) entering a surface at point x_i, (2) multiply scattering—often several dozens of times—below the surface, and (3) exiting at another point x_o. (b) The aggregate behavior is commonly simplified to a BSSRDF $S(x_o, \omega_o, x_i, \omega_i)$, which only depends on incident and exitant light rays.

Creating convincing images of certain commonly-found materials, such as marble, skin, or milk, requires the simulation of *subsurface scattering*, where light (1) enters a surface at one point, (2) scatters many times in the volume beneath the surface, and (3) exits the surface at another point [97] (cf. Fig. 5.9). One way to model this is using the radiative transfer equation (RTE), which we have just seen in Sec. 5.9.

Unfortunately, light transport in very dense media—such as the materials just mentioned—is computationally expensive, and as such, a more efficient approach is to capture the aggregate behavior of the multiply-scattering material as the so-called *bidirectional scattering-surface reflectance distribution function* (BSSRDF) [154].

On one hand, this approach effectively adds another surface integral to our local reflection equation:

$$L_o(w_o, \omega_o) = \int\limits_{\mathcal{M}} \int\limits_{\mathcal{H}_+^2(x_i)} S(x_o, \omega_o, x_i, \omega_i) L_i(x_i, \omega_i) \, \mathrm{d}\omega_i^\perp \, \mathrm{d}x_i, \tag{5.59}$$

where the BSSRDF $S(.)$ replaces the simpler BRDF (or BSDF) model of scattering. (Note the directional integration is over $\mathcal{H}_+^2(x_i)$, the upper hemisphere around x_i, as we are only interested in light arriving from *above* the surface). On the other hand, we do not have to concern ourselves with inscattering, outscattering and absorption in the volume beneath the surface anymore.

A BSSRDF model commonly used in computer graphics was introduced by Jensen et al., together with a method for measuring the BSSRDF of real materials [97]:

$$S(x_o, \omega_o, x_i, \omega_i) = S_d(x_o, \omega_o, x_i, \omega_i) + S^{(1)}(x_o, \omega_o, x_i, \omega_i). \tag{5.60}$$

This model splits the BSSRDF into a single-scattering component $S^{(1)}(.)$, based on the radiative transfer equation, as well as a diffusion component

$$S_d(x_o, \omega_o, x_i, \omega_i) = \frac{1}{\pi} F_t(\eta, \omega_o) R_d(\|x_o - x_i\|) F_t(\eta, \omega_i) \tag{5.61}$$

accounting for multiple scattering, where $F_t(.)$ is the Fresnel (transmission) term, η is the relative index of refraction at the boundary, and $R_d(.)$ is a radially symmetric scattering profile using the diffusion approximation.

Ignoring the (incident) Fresnel term, the model can often be further simplified to a surface integral of irradiance weighted by a single radial scattering profile R [116]:

$$\begin{aligned} L_o(w_o, \omega_o) &= \int\limits_{\mathcal{M}} R(\|x_o - x_i\|) \int\limits_{\mathcal{H}_+^2(x_i)} L_i(x_i, \omega_i) \, \mathrm{d}\omega_i^\perp \, \mathrm{d}x_i \\ &= \int\limits_{\mathcal{M}} R(\|x_o - x_i\|) E(x_i) \, \mathrm{d}x_i. \end{aligned} \tag{5.62}$$

Various other models and integration strategies for subsurface scattering have been suggested [37, 98].

5.11 Rendering Methods

This section will give a high-level overview of various physically based rendering (PBR) methods. Except for Radiosity (Sec. 5.11.1), all of them are based on sampling (in various ways) random light transport paths in order to solve the integral equations given earlier, which we will take advantage of in our artistic light transport visualization and editing system (Ch. 9). In the literature, solving the full light transport in a scene is also known as *global illumination* (GI), in contrast to *direct illumination* (only paths of length 2, e.g. single reflections), or even *local illumination* (ignoring visibility to light sources).

The number of methods and literature on the topic is vast and research is an ongoing effort; with the exception of one particular many-lights method for participating media, we will only give the basic "flavor" of each method, without getting into too much detail about current trends in research.

5.11.1 Radiosity

Historically, the radiosity method [25, 61, 185] was one of the first methods in CG to capture global illumination[21] and spawned a lot of research in the field

The basic premise is to first assume all materials are diffuse reflectors. Starting with the recursive light transport equation in three-point form and moving the diffuse BRDF out of the integration domain (Eq. 5.63). Noting that for diffuse reflectors $B = \pi L$, we replace radiance with radiosity and arrive at the continuous radiosity equation (Eq. 5.64).

$$L(x \to z) = L_e(x \to z) + \frac{\rho(x)}{\pi} \int_{\mathcal{M}} L(x \leftarrow y) G(x,y) V(x,y) dA(y) \tag{5.63}$$

$$B(x) = B_e(x) + \frac{\rho(x)}{\pi} \int_{\mathcal{M}} B(y) G(x,y) V(x,y) dA(y) \tag{5.64}$$

The second assumption is that the scene $\mathcal{M}$ is partitioned into a finite set of N surface patches $\mathcal{M}_i$ *with constant radiosity, emission and reflectivity*:

$$B_i = \frac{1}{A_I} \int_{\mathcal{M}_i} B(x)\, \mathrm{d}A(x) \tag{5.65}$$

$$B_i^e = \frac{1}{A_I} \int_{\mathcal{M}_i} B_e(x)\, \mathrm{d}A(x) \tag{5.66}$$

$$\rho_i = \frac{1}{A_I} \int_{\mathcal{M}_i} \rho(x)\, \mathrm{d}A(x), \tag{5.67}$$

where $A_i = \int_{\mathcal{M}_i} \mathrm{d}A(x)$ is the patch area. Substituting terms, one can arrive at a linear equation for the radiosity B_i of a single surface patch $\mathcal{M}_i$:

[21] The seminal paper by Goral et al. is also the origin of the now-famous *Cornell Box* scene (cf. Figs. 7.1 and 7.2).

$$B_i = B_i^e + \rho_i \sum_{j=1}^{N} B_j F_{i,j}, \tag{5.68}$$

$$F_{i,j} = \frac{1}{\pi A_i} \int_{\mathcal{M}_i} \int_{\mathcal{M}_j} G(x,y) V(x,y) \, \mathrm{d}A(x) \, \mathrm{d}A(y), \tag{5.69}$$

where $F_{i,j}$ in Eq. 5.69 is the non-symmetric *form factor*, encoding the amount of flux transported from $\mathcal{M}_i$ to $\mathcal{M}_j$.

Given this linear system of equations, one can apply standard matrix inversion techniques from numerical analysis [184] to arrive at the resulting patch radiosity B_i (and hence radiance $L_i = B_i/\pi$), though more specialized solutions methods exist (cf. [25, 185]). The solution is also commonly smoothed by interpolation.

A strong advantage of the radiosity method is the view-independence of the solution, to the effect that, say, architectural walk-throughs can be rendered in real-time.[22] Various improvements of the basic method have been developed over the years, such as extensions to non-diffuse BRDFs [82], (isotropic) participating media [195], mesh-less patch representations [133], or even implicit visibility through negative light transport [35].

However, its high memory demand and limitation to mostly diffuse materials has lead the radiosity method to fall out of favor in both professional rendering and research communities. We also note that this method is the only one in our recital that is not based on sampling light paths, and so the applicability of our artistic editing approach in path space (Ch. 9) would be very limited.

5.11.2 Path Tracing

Together with the rendering equation (cf. Sec. 5.5), Kajiya presented a practical solution method of said equation in the same article [100].[23] This method is now commonly known as *path tracing* (PT) and a very basic version, sometimes called *naive path tracing*, can be implemented as a single-sample estimator of the measurement equation (Eq. 5.32) and light transport equation (Eq. 5.25):

```
# render an image of resolution w, h with N samples per pixel
def render_image(w, h, N):
    # for each pixel...
    for j in range(h):
        for i in range(w):
            # compute measurement integral (via Monte Carlo)
            I_j = 0
            for k in range(N):
                I_j += estimate_measurement_equation(i, j)
            I_j /= N

            put_pixel(i, j, I_j)


# compute a single measurement for pixel i,j
```

[22]Indeed, precomputing light maps in computer games has been a popular use in the past.

[23]Kajiya even anticipated, though not in any practical detail, Metropolis Light Transport, which was developed and presented by Veach and Guibas eleven years later [231].

```
def estimate_measurement_equation(i, j):
    x, w, W_e, pdf = sample_emitted_importance(i, j)
    L_i = estimate_incident_radiance(x, w)

    return (W_e * L_e) / pdf

def estimate_incident_radiance(x, w):
    # see if we hit something
    y = trace_ray(x, w)
    if y:
        # surface emission
        L_e = evaluate_emitted_radiance(y, -w)

        # sample incident direction
        # (projected solid angle measure)
        w_i, pdf = sample_direction(y)

        # recursively compute reflected radiance
        f_s = evaluate_bsdf(y, w_i, -w)
        L_i = estimate_incident_radiance(y, w_i)
        L_r = (f_s * L_i) / pdf

        return L_e + L_r
    else:
        # image-based lighting etc.
        return evaluate_env_radiance(-w)
```

Note the recursive call to `estimate_incident_radiance()`. As long as the scene is not closed, this will work, as eventually our ray will miss geometry and hence terminate the recursion. Incidentally, we can see that the name path tracing stems from the fact that there is at maximum one recursive call, hence we will have implicitly built a path of vertices, whenever we have hit an emissive object.

One can extend this basic sketch and, e.g., make it usable for real scenes by adding Russian roulette, replace recursion with iteration by tracking the path's throughput, make it progressive by looping over samples *outside* looping over pixels, sample directions proportional to the BSDF to reduce variance, extend it to participating media by sampling distances, etc. However, while this basic version will produce very noisy images, it does eventually converge to the correct solution of the LTE.

One notable extension that has made path tracing practical is *next event estimation*. By recursively expanding the LTE once, one arrives at a version that splits reflected radiance into a direct and an indirect component; direct illumination is then evaluated using dedicated techniques for light sampling, whereas indirect illumination is computed recursively, most often leading to a drastic reduction in variance.[24]

While certainly not the most efficient rendering method, path tracing with next event estimation (and combination with multiple importance sampling (MIS) [229]) is very robust, has long supported participating media [129], and can easily be extended to handle new emission and material models, or new sampling strategies. As such, it also lends itself well as a ground truth reference which—especially, approximate—other methods have to match. Transport with near-Dirac distributions is a major challenge, however [104].

Mostly of historical note now, *distributed* (or *distribution*) *ray tracing* by Cook, Porter and Carpenter [29] shaped ideas in 1984 which are now standard in path

[24]Strictly speaking, with next event estimation, we are not tracing a path anymore, but a tree. We will revisit the topic in Sec.8.2.

tracing algorithms, such as sampling the aperture for depth of field, sampling the shutter interval for motion blur, and sampling the hemisphere of directions for glossy reflections. With the more recent PBR trend, path tracing has become popular in movie production, a major paradigm shift for the industry [22].

5.11.3 Light Tracing

We can follow exactly the same simple steps used to implement path tracing above to develop its dual, *light tracing* (LT, see e.g., [42]). By exchanging radiance with importance, and the BSDF with its adjoint, we arrive at an algorithm following the "natural" flow of light from light source to sensor.

Unfortunately, compared to the naive path tracer we have seen above, the chance of accidentally hitting the camera's aperture/entrance pupil *and* a specific sensor pixel is minuscule. While this can be remedied by splatting a path's contribution to the corresponding pixel if we hit *any* point on the sensor, and can be further improved to some extent with next-event estimation, light tracing is not very practical as a stand-alone rendering method.

However, light tracing is important as a building block of other rendering methods, specifically: Bidirectional Path Tracing (Sec. 5.11.4), Photon Mapping (Sec. 5.11.6), and Instant Radiosity (Sec. 5.11.5).

5.11.4 Bidirectional Path Tracing

As its name suggests, the idea of *bidirectional path tracing* [128, 230] (BDPT[25]) is to combine paths by sampling both from the sensor (i.e., path tracing) and the light sources (i.e., light tracing), in the hope of thereby sampling path space (cf. 5.8) in a more intelligent manner.

The basic approach would just connect two paths $x_0x_1...x_k$ and $y_0y_1...y_l$, at the end points, evaluating two BSDFs, the geometry term $G(x_k, y_l)$ and visibility function $V(x_k, y_l)$. However, a core idea to make bidirectional path tracing more efficient is to reuse samples by enumerating (and evaluating) all possible combinations of generating a path of the same length. The key improvement by Veach and Guibas is to use *multiple importance sampling* to combine these path samples in an optimal way, e.g. non-uniformly weighting each path by the power heuristic [230]. [26]

While samples are more costly to evaluate in BDPT, compared to PT, the ability to capture difficult light transport paths improves overall efficiency in scenes where such light transport paths, e.g. very indirect illumination or complicated caustics, are prevalent.

The reason bidirectional methods are better at finding important light paths can be seen when interpreting them as a form of graph search [205]: Given a pair of vertices, one on the sensor, one at a light source, is there a path connecting the two? Depending on the topology of the graph, this is inherently easier when simultaneously searching from both ends, as we might otherwise explore a large subgraph from one end that does not lead to a (useful) result. Unlike classical graph search problems, the number of vertices is infinite, i.e., every surface point is a vertex, every mutually visible pair of vertices form an edge.

[25]The abbreviation is due to the spelling "bi-directional path tracing" in Lafortune and Willems' article [128].

[26]Multiple importance sampling is used to improve MC estimators in other contexts as well, such as combining BSDF sampling and light sampling in direct illumination.

As with other path sampling methods, BDPT can be extended to include participating media [129], however, as with path tracing, overall efficiency in scenes with heterogeneous, dense media, i.e., time until reaching a visually converged image, is a major concern and various methods outside path tracing have been suggested [94, 158, 159, 186].

5.11.5 Many-Lights Methods

Instant radiosity [111] (IR)[27] forms the basis of so-called *many-lights methods*, a class of techniques where indirect illumination in a scene is represented by direct illumination of *virtual point lights* (VPLs).[28] We will give an overview of many-lights methods in this section, but refer to a recent review [33] for a more thorough discussion.

Despite their historically different origins, many-lights methods like IR can be seen as a bidirectional estimator in the spirit of BDPT (Sec. 5.11.4) which only evaluates a very specific combination of eye and light paths, namely eye paths x_0x_1 of length-1 and light paths $y_0y_1...y_l$ of arbitrary length l, connecting x_1 to all path vertices $y_j, j \in [0, l]$ (cf. Fig. 5.10). At first sight, these limited path sampling abilities would likely just increase variance, i.e. image noise.

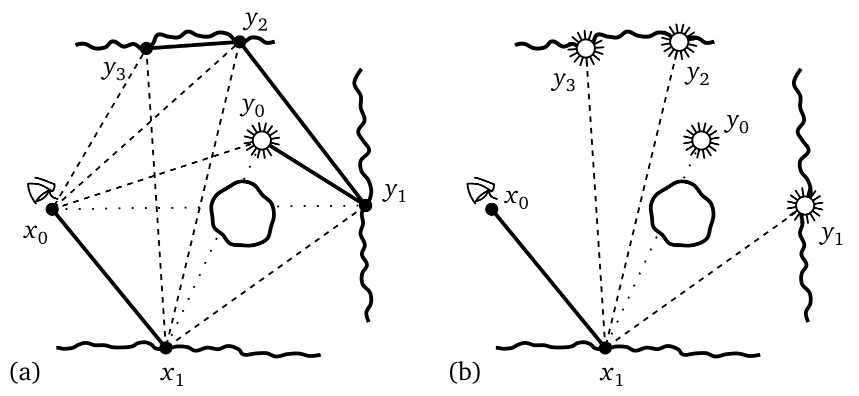

Figure 5.10: Instant radiosity as a specialized bidirectional estimator. (a) Bidirectional path tracing (BDPT) considers all possible connections of eye and light path vertices (x_i and y_j, resp.). (b) Many-lights methods such as instant radiosity [111] consider the light path vertices y_j to be virtual point lights (VPLs), limit eye path length to one, and only consider direct connections to VPLs.

The key observation by Keller was that, at least in a mostly diffuse setting, the light path vertices connected in this limited way could be also interpreted as point lights, which are very efficient to evaluate using graphics hardware. Indeed, when reusing a light path's vertices for all eye paths as VPLs, i.e., correlating the light path samples, one can very efficiently test the visibility of each "shading point" x_1 and a VPL y_j via shadow maps [242], a technique efficiently implemented in real-time using GPUs' *rasterization pipeline*, as opposed to tracing more costly rays. Thereby,

[27]Although the name contains "radiosity," IR is very unlike the finite-element method we encountered in Sec. 5.11.1; we should probably read this as "a faster method to render diffuse scenes."

[28]We also note that placing point lights in a scene has traditionally been a popular technique for *faking* indirect illumination, e.g., in movie production [22].

instant radiosity provides scalable rendering, ranging from fast interactive previews to high-quality final renders.[29]

Instant radiosity is a two-pass method. The first pass, also referred to as a random walk, samples N light paths. This is analogous to BDPT (Sec. 5.11.4), but light path vertices are interpreted as $M \geq N$ virtual point lights. Together with the VPL position y_j, additional quantities such as incident radiance L_i, incident direction ω_j, surface normal N_{y_j}, and local BSDF/phase function (parameters) f_s/f_p need to be stored for later use.[30]

Specifically, the random walk samples a Monte-Carlo estimator of outgoing equilibrium radiance (using a slightly different form of the volumetric LTE, Eq. 5.46):

$$L^\infty = \frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{\infty}\frac{L_e(y_0,\omega_0)}{p(y_0)}\prod_{j=0}^{k-1}\frac{\tau(y_j,y_{j+1})\hat{C}_j}{p(\omega_j)p(s_j)},$$

where

$$\hat{C}_j = \begin{cases} 1 & \text{if } j=0, \\ f_p(y_j,\omega_{j+1},-\omega_j)\ \sigma_s(y_j) & \text{if } x_j \in \mathcal{V}, \\ f_s(y_j,\omega_{j+1},-\omega_j)\ \left|\omega_j \cdot N_{y_j}\right|) & \text{if } x_j \in \mathcal{M}, \end{cases}$$

and $p(y_0)$, $p(\omega_j)$, and $p(s_j)$ are the probability density functions for sampling positions, directions, and distances, respectively. The random walk starts at a position y_0 on a light source and then proceeds by alternating between sampling directions ω_j and distances s_j to the next scattering event $y_{j+1} = y_j + s_j\omega_j$, until eventually terminating the path. Distance sampling in homogeneous media can be done analytically, while in heterogeneous media, e.g., using Woodcock tracking [243]. The VPL positions y_j can be either in the volume or on the closest surface; in either case, one stores the incident radiance $L_i(y_{j+1},\omega_{j+1}) = L_i(y_j,\omega_j)\tau(y_j,y_{j+1})\hat{C}_j/p(\omega_j,s_j)$, direction ω_{j+1} and scattering coefficient $\sigma_s(y_{j+1})$. Alternatively, for the initial y_0 on a light source, one stores $L_e(y_o,\omega_o)/p(y_0)$. Finally, the VPLs' radiance is normalized by dividing by the total number of paths, N.

We note that this random walk is virtually identical to photon mapping (Sec. **??**

In the second pass, illumination for all primary rays is computed simultaneously by accumulating the VPLs' direct illumination at "shading points" x_i, treating each VPL as a point light source with emission profile shaped by its outgoing radiance, i.e., the convolution of incident radiance and the local BSDF/phase function, and also taking into account the geometry term $\hat{G}(x_i,y_j)$, visibility function $\hat{V}(x_i,y_j)$, and, if applicable, attenuation $\tau(x_i,y_j)$. In general, one needs to integrate the VPLs' contribution along the eye ray to account for in-scattering, which can be done using the Monte-Carlo estimator:

$$L_i(x,\omega) = \frac{1}{K}\sum_{i=0}^{K-1}\frac{\tau(x,x_i)}{p(x_i)}\sum_{j=0}^{M-1}\hat{f}(x_i,\omega,\omega_i)\hat{G}(x_i,y_j)\hat{V}(x_i,y_j)L_j(y_j,-\omega_i),$$

where K is the number of samples x_i along the eye ray (which may be in the volume or on the nearest surface), M the number of VPLs, L_k the j'th VPL's outgoing

[29]Which brings us back to our short discussion on Monte Carlo efficiency in Sec. 2.4.5, which stated that a higher-variance technique may actually outperform a low-variance technique overall.

[30]The original IR only supported (mostly) diffuse surface transport, and hence fewer quantities were stored per VPL.

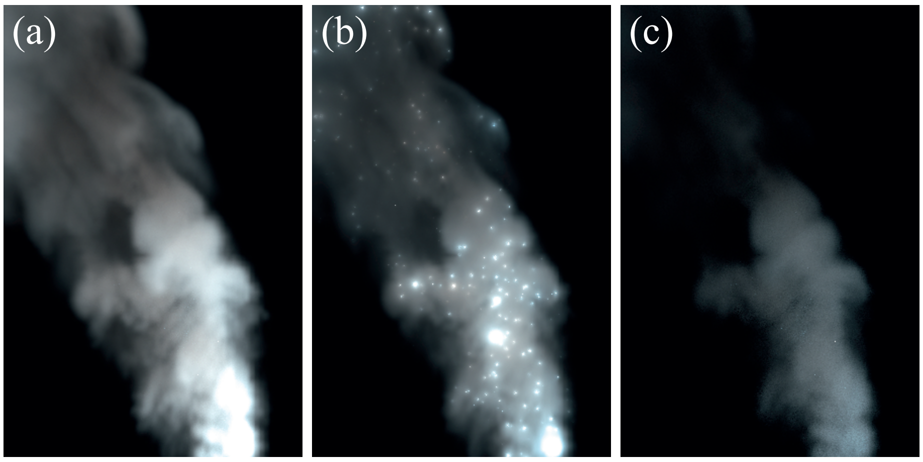

Figure 5.11: Clamping artifacts in VPL-based methods. (a) Unbiased rendering of heterogeneous smoke. (b) Rendering multiple scattering with VPLs causes characteristic "splotches." These can be avoided by clamping, however, (c) this removes a significant amount of energy. Image taken from [46].

radiance towards x_i (evaluated using the stored quantities mentioned above), and ω_i the normalized direction from x_i to the VPL's position y_j. To reduce variance, the positions along the primary ray may be sampled proportional to attenuation τ or, alternatively, according to the inverse squared distance term between eye ray and VPL position (a factor of $\hat{G}$).[31]

Implementing IR as described is unbiased, i.e., there is no systematic image estimation error, and trivially progressive: One may simply add VPL contributions to the image one at a time, alternating between splatting VPL contributions and sampling new light paths (when running out of VPLs), until the image has converged. As another advantage, images rendered in this manner, even with only a few hundred VPLs, are much less prone to noise, because the same light path is *reused* for many pixels.

Variance in sampling light paths, however, *does* manifest itself in many-lights rendering, via bright, "splotchy" artifacts (cf. Fig. 5.11.5) caused by the geometry term $G(x_1, y_l)$, which was introduced in our discussion of hemispherical integration (Sec. 2.6) and the surface form of the LTE (Sec. 5.5). The squared distance term in the denominator of the geometry term is called a *weak singularity* [121, 186], because G may become arbitrarily large if the two input points approach one another. This is what happens with VPLs' contributions when they are close to the primary ray hit points, where they are evaluated.[32]

A pragmatic solution to the problem of the weak singularity is to introduce *clamping*, whereby the geometry term G is replaced by a new term G':

$$G'(x_1, y_l) = \min\left\{b, G(x_1, y_l)\right\}, \tag{5.70}$$

for some constant b, which effectively caps the geometry term's value in a region

[31]When clamping the geometry term as shown below, the latter is not very effective [46].

[32]This is not the same problem as sampling very low-probability light paths, though that *may* additionally happen.

around the VPL.

For scenes with diffuse and mildly glossy materials, instant radiosity with clamping is a good technique for interactive global illumination previews. As given above (and as with the other rendering methods we have encountered so far), the IR/VPL algorithm is easily formulated in terms of participating media [186]; a self-contained description can also be found in [46].

Clamping the contribution introduces bias into the images, and—what is especially unfortunate from a practical perspective—effectively changes the perceived material of objects, especially for glossy transport [124]. Bias compensation for IR with surfaces has first been described by Kollig and Keller [121] and later extended to participating media by Raab et al. [186] (as we will outline in Sec. 6.1). In recent work, Davidovič et al. [36] and Novák et al. [157] introduced fast approximations based on placement of new light sources and bias compensation in screen-space, respectively, but both approaches are limited to surfaces only. We will revisit this topic in Ch. 6.

A different approach to overcoming the weak singularity is to "inflate" VPLs into *virtual spherical lights* (VSLs) [71] around a light path vertex. To further reduce variance in the context of participating media, this approach was later extended into *virtual ray lights* (VRLs) [159] and *virtual beam lights* (VBLs) [158], which spread energy over a whole light path segment, and (in the case of VBLs) additionally inflate it into a cylinder. These inflation approaches lead to bias, however, and one needs to carefully reduce the inflation radius to make these methods consistent, a topic which we will discuss when talking about photon mapping (Sec. 5.11.6).

Finally, the performance of many-lights methods may be improved by cheaper, imperfect visibility testing [192], or incremental updates [130]. While these methods reduce the cost per VPL, many-lights methods' performance may be made *sublinear* in the number of VPLs by clustering them into hierarchies and evaluating representative inner nodes instead of individual VPLs at leafs; a partition representing all (virtual) lights via inner or outer nodes is called a *lightcut* [234–236]; we note such techniques do have guaranteed error bounds, but are limited to homogeneous media and perform no bias compensation for clamping, if employing the latter. Other techniques have investigated how to further cut down the number of lights evaluated [72, 73] or how to place them more efficiently [56].

We also find that many-lights methods are a good technique for still images, but for animations, using only a few thousand VPLs may lead to objectionable flickering. This can be partially remedied by reusing random seeds or only incrementally updating VPLs [130], but the proper solution is to sample the light field more densely. Similarly, glossy transport dramatically increases the number of VPLs needed to achieve acceptable image quality, though recent, costlier VPL representations can improve on otherwise severely undersampled light fields and animation flickering [209]. Other methods only slightly related to IR, such as reflective shadow maps [34], also fall under the umbrella of many-lights method. We again refer to a more comprehensive recent article on the state of the art in many-lights rendering for further information [33].

5.11.6 Photon Mapping

Photon mapping (PM) [95], similarly to many-lights methods (Sec. 5.11.5), can be seen as a variant of light tracing (Sec. 5.11.3), in the sense that it constructs transport paths in a *random walk* starting from light sources. In contrast to light tracing, however, where one accumulates a path's contribution when hitting a sensor, in PM we store a contribution *whenever* we hit some non-specular object. The distribution of light so collected is called a photon map.

Photon mapping is a two-pass rendering method. In the first pass, populations of photons[33] are emitted at light sources, carrying some initial flux $\Delta\Phi$ (inversely proportional to the number of light paths). They are subsequently traced through the scene, and whenever a photon is scattered at a surface, its flux is reduced. Alternatively, one may keep the flux of photons constant and terminate the path with probability proportional to the local reflectance, so as to avoid large variance in the photon map. At every non-specular interaction, an entry in the photon map is stored with the photon's flux $\Delta\Phi_k$, position x_k and incident direction ω_k.

In the second pass, the photon map is used to estimate the outgoing radiance at surfaces hit by ray tracing from the camera, after potentially recursing on specular interactions. To compute the outgoing radiance towards the ray origin at a hit point, we start by assuming we can collect N nearby photons in a sphere of radius r, and also recall the definition of (incident) radiance (cf. Sec. 5.1):

$$L_i(x,\omega) = \frac{\mathrm{d}^2\,\Phi_i(x,\omega)}{|\cos\theta|\,\mathrm{d}\sigma(\omega)\,\mathrm{d}A(x)}. \tag{5.71}$$

With this, the scattering equation we want to evaluate can be progressively simplified:

$$\begin{aligned}
L_s(x,\omega) &= \int_{S^2} f_s(x,\omega,\omega_i) L_i(x,\omega_i) \left|\cos\theta_i\right| \mathrm{d}\sigma(\omega_i) && (5.72)\\
&= \int_{S^2} f_s(x,\omega,\omega_i) \frac{\mathrm{d}^2\,\Phi_i(x,\omega_i)}{\mathrm{d}A(x)} && (5.73)\\
&\approx \sum_{k=1}^{N} f_s(x,\omega,\omega_k) \frac{\Delta\Phi_k(x,\omega_k)}{\Delta A} && (5.74)\\
&= \sum_{k=1}^{N} f_s(x,\omega,\omega_k) \frac{\Delta\Phi_k(x,\omega_k)}{\pi r^2}. && (5.75)
\end{aligned}$$

The sum over nearby photons' energy (Eqs. 5.74 and 5.75) is called *density estimation*, in this case using a simple constant kernel. The nearest-neighbor search, i.e., for photons in a small sphere around the point x, is usually accelerated with a k-D tree.

Photon mapping is excellent at capturing intricate specular light transport phenomena, including specular-diffuse-specular paths such as the light pattern seen at the bottom of a water pool, which are difficult to sample for many other methods. This ability to capture difficult paths however comes at the cost of *bias*, i.e., a systematic error in the radiance estimate, which vanishes only in the limit of using infinitely many photons. Bias shows itself by blurring of the light field, which is caused by the finite density estimation radius; this blurriness can be reduced by shrinking the radius, but this at the same time will produce more variance.

[33] These are only "photons" by name, as opposed to the ones we encountered in Sec. 5.1.

Photon maps of smooth diffuse illumination also tend to look very “splotchy,” due to the uneven distribution of photons. This can be improved by *final gathering*, i.e., an extra ray tracing pass that uses the photon map only for indirect illumination. As this may remove caustics too, one usually generates and uses multiple photon maps for different phenomena, taking care that the set of all photon maps collectively still fully partitions path space.[34]

Later research introduced *progressive photon mapping* [66, 119], augmenting the basic technique by progressively shrinking the density estimation radius in a controlled fashion, thereby ensuring the method is at least *consistent*, i.e., in the limit of rendering for an infinite amount of time, the systematic error vanishes. Further improvements include better handling of glossy materials [64] and optimally selecting estimation radii and shrinkage rates [103]. The latter also showed that the asymptotic convergence rate of photon mapping is worse than that of path tracing, despite producing acceptable images faster.

Density estimation may also be seen as a special connection method in the bidirectional path tracing framework, and so ideas from PM found their way into a technique commonly called vertex connection and merging (VCM) [55, 67]. Photon mapping has also been extended to participating media, following the same evolution from biased [96] to consistent technique [92], more elaborate methods [91, 94], and attempts to unify it with different rendering methods [125].

In movie production, photon mapping and its extensions have traditionally only been used in isolated shots, but were never popular as a general rendering method [22].

5.11.7 Metropolis Light Transport

In contrast to the methods we have seen so far, *Metropolis light transport* (MLT) [231] builds on an entirely different mathematical framework, utilizing the Metropolis-Hastings algorithm to sample light transport paths. As such, it is an example of so-called Markov chain Monte Carlo (MCMC) methods. While path tracing and its various extensions sample light transport paths independently—though possibly reusing some samples—, MLT inherently works with correlated samples, given by successive states of a Markov chain.

The Metropolis-Hastings algorithm roughly works as follows [23, 177]: Given a function $f : \Omega \to \mathbb{R}$ and an initial state $X_0 \in \Omega$, first generate a new state X', called a *mutation*, from the current state X_i by some strategy. Now, the next state X_{i+1} is either the proposed state X' or the original state X_i, subject to whether the mutation is *accepted* or *rejected*. This process is then repeated subsequently. Without going into detail, if the probabilities $T(X \to X')$ for proposing a new state and $a(X \to X')$ for accepting the proposal hold certain conditions,[35] then in the limit, the sequence of states X_i, i.e., the *Markov chain*, approaches a distribution which is proportional to the original function f.

Building on the concept of path space, the original MLT by Veach and Guibas [231] creates a sequence of states that are light transport paths, by continually proposing mutations of some initial path and either accepting or rejecting them. Paths are thereby allowed to move across the image plane, and by recording a histogram

[34]Final gathering may also be applied to other “low-quality” methods such as radiosity (Sec. 5.11.1).

[35]Specifically, $T(.)$ must be computable (unless symmetric); T and a must uphold the *detailed balance* property; and furthermore T must also be *ergodic*. See [23, 177, 231] for more details.

of path contributions for each pixel, an image proportional to a solution of the measurement integral is formed.

A key factor to making MLT effective is designing good mutation strategies that ensure that (a) the entire state space is explored (called *ergodicity*), and (b) that once an important feature is found, it is explored thoroughly before “jumping out” too soon. These two goals oppose each other; however, one may use multiple mutation strategies targeted at capturing specific lighting features, which the Metropolis-Hastings algorithm will then eventually find and explore.

The problem of finding a starting state X_0 can be solved by initially running the Markov chain for a while in a burn-in phase before recording samples, or more unbiasedly by sampling paths with, e.g., bidirectional path tracing and reweighting the histogram, thereby also finding the appropriate scaling factor for the histogram [177].

An interesting variant is due to Kelemen and Szirmay-Kalos [110], who use the (infinite) space of random numbers $[0,1]^\infty$ as a state space. The advantage is that the state space is independent of the underlying path sampling-based rendering method, and can, e.g., also be used with unidirectional path tracing or many-lights methods. At the same time, however, this is also a disadvantage for designing mutations for specific lighting features, as the connection between the Markov chain's state and the generated paths is only incidental, a problem which has only recently been addressed [166]. As with other methods, MLT has been extended to participating media [169]. More recent work focuses on exploring different state spaces [70, 87, 105] or combining multiple importance sampling with MLT [65].

MLT is notorious for being difficult to implement,[36] but an even more important practical issue is its uneven convergence behavior with few samples. The Markov chain(s), moving more or less randomly over the image plane, may cause features to suddenly appear. This behavior makes it both impractical for fast, predictive rendering, as well as for animations: If light suddenly disappears in one frame and suddenly reappears in another, this causes flickering, which is more objectionable to human observers than the more uniform noise of pure Monte Carlo methods. Hence, it has met with only limited popularity in commercial rendering [126], and some works thus focus on improving the uniformity of convergence behavior (e.g. [24, 167]).

5.12 Practical Aspects

Monte Carlo Noise One challenge with using Monte Carlo (MC) integration for rendering is that the resulting images may be still very noisy even after taking a large number of samples, due to the comparatively slow MC convergence behavior. This unwanted artifact is only enhanced with animations, and leads to flickering between frames. Generally, advanced denoising techniques that selectively smooth the image have made some MC noise in movie production tolerable [22, Sec. 6.2].

However, an extreme form of MC noise are so-called “fireflies.” By way of example, imagine the following MC estimator:

$$F = f(X) = \begin{cases} \frac{c}{\alpha} & \text{if } X < \alpha, \\ \frac{c}{\beta} & \text{if } \alpha \leq X < \alpha + \beta, \\ 0 & \text{otherwise,} \end{cases} \tag{5.76}$$

[36]Indeed, there have been “metapapers” written on it [23].

with $0 < \alpha \ll \beta < 1$, $\alpha + \beta < 1$ and $X \propto U(0,1)$, which computes just the constant value c:

$$E[F] = \alpha \cdot \frac{c}{\alpha} + \beta \cdot \frac{c}{\beta} + (1 - \alpha - \beta) \cdot 0 = c. \tag{5.77}$$

If we select, say, $\alpha \approx 10^{-6}$ and $\beta = 0.8$, and use F to compute a uniformly flat, gray image with millions of pixels, we will eventually get a very high contribution in some pixels, for which $X < \alpha$. Once we have such an (unlucky) firefly, even additional thousands of samples per pixel cannot compensate for the huge c/α contribution of one very unlikely sample to the Monte Carlo sum.

Sampling very low-probability light paths in physically based rendering essentially amounts to the same problem. We note that debugging what causes such improbable paths from just the rendered image is difficult, as the low-probability event may have happened not at the camera ray hit point, but somewhere else entirely.[37]

Progressive Rendering Even in a production pipeline that is fully physically based, that is, adhering to the principles outlined above, the final effect of specific choices of lighting and material parameters can be hard to predict. Hence, interactive feedback during both early look development and later rendering of (final) frames is highly beneficial to the process of image and animation production, by cutting down iteration times and allowing better exploration of the space of all available material, lighting, and renderer parameters. See Enderton and Wexler [44] for a discussion how improving interaction speed leads to various step-wise shifts in workflow quality.

Some effort has been put in creating dedicated relighting engines [72, 176, 187] that allow fast, interactive parameter changes. However, as they are separate systems from the final production renderer, this incurs additional maintenance and pipeline complexity costs. Partially aided by the recent transition to PBR and the dominance of ray tracing-based renderers, nowadays it is more common to utilize *progressive image updates* in the core renderer. Commercially available production renderers, such as *Arnold* [211] and *RenderMan* [178], support this workflow and our interactive artistic visualization/editing system (Ch. 9) also relies on this fact as a prerequisite.

Progressive updates are straightforward to implement in pure unbiased Monte Carlo (MC) rendering algorithms such as path tracing (Sec. 5.11.2), bidirectional path tracing [128] (Sec. 5.11.4), or basic many-light methods such as instant radiosity [111] (Sec. 5.11.5). We remember that Monte Carlo integration is a summation of independent terms, as in a standard MC estimate with N samples:

$$I = \int_\Omega f(x)\,\mathrm{d}\mu(x) \approx I_N = \frac{1}{N}\sum_{i=1}^{N} \frac{f(x_i)}{p_\mu(x_i)}. \tag{5.78}$$

Given this, we can compute our next estimate as:

$$I_{N+1} = \frac{1}{N+1}\sum_{i=1}^{N+1} \frac{f(x_i)}{p_\mu(x_i)} = \frac{N}{N+1} I_N + \frac{1}{N+1}\frac{f(x_{N+1})}{p_\mu(x_{N+1})}, \tag{5.79}$$

using *a single* new sample x_{N+1}. In practical terms, the transition from I_N to I_{N+1} is our progressive rendering update, only we have of course one separate estimator per pixel.

[37] Indeed, this is why we suggest to employ light transport visualization (Ch. 8) as a debugging aid for non-artists, as well.

We further note that MC integration is essentially just computing a weighted mean of random samples; as both weighted mean and weighted variance of a finite sample size can be calculated incrementally [51], we can thus also accumulate and visualize the image error (given by the square root of variance) in progressive rendering.

More complex rendering algorithms, however, require more careful consideration, not only in terms of making them progressive, but also how inherently predictive the progressive updates are.

As an example, Markov chain Monte Carlo (MCMC) techniques like Metropolis light transport (MLT) [231] (cf. Sec. 5.11.7) are very efficient at sampling important but difficult-to-find light paths and are implicitly progressive: the image formation amounts to recording a histogram of correlated states. Unfortunately, the convergence behavior over the image plane is uneven, which makes MCMC methods less useful in a production setting with frequent restarts of the renderer and progressive updates [126].

Specifically, while, e.g., path tracing converges slower in difficult scenes, it exhibits very predictable behavior and relatively uniform noise across the image.[38] In contrast to this, the convergence behavior of MCMC techniques can be much less predictable. While rendering, the image may seem deceivingly smooth and converged for a long time, until suddenly an entirely new lighting feature is discovered by the Markov Chain which was previously unseen due to being well isolated in path space. This “erratic” behavior has a dramatic effect on appearance, making editing thereof more difficult.

Advanced rendering techniques that introduce bias into the Monte Carlo method are usually motivated by more efficient convergence behavior. We have already seen examples of such methods in our discussion of many-lights methods (Sec. 5.11.5) and photon mapping (Sec. 5.11.6), and both classes of methods can provide sub-linear render time in the number of simulated light paths by employing hierarchical data structures when looking up VPLs or photons, respectively.

Biased methods—indeed, all stochastic methods—can be trivially made progressive by averaging multiple images, provided they were rendered in a statistically independent fashion (which boils down to selecting different pseudo-random number generator seeds per image.) This approach, however, only removes variance over time, but not bias. Furthermore, hierarchical data structures are most beneficial for large numbers of light path vertices; splitting a fixed photon or VPL budget into multiple passes may significantly lower efficiency.

Bias may be handled by judiciously adjusting algorithms’ parameters in every iteration, to ensure that both variance and bias diminish to zero in the limit. This was first popularized in photon mapping by Hachisuka et al. [64,66], who suggested progressively *shrinking the density estimation kernel radius* across passes, based on extra statistics collected at camera ray endpoints. Knaus and Zwicker [119] later realized that if render passes are independent and identically distributed, the same asymptotic behavior can be achieved without collecting additional statistics, as the kernel shrinkage rate solely depends on the dimensionality of the blur. This allows treating the rendering method largely as a black box with a single adjustable estimation radius parameter. Along these lines, Jarosz et al. used the same probabilistic approach in a progressive variant [92] of the photon beams algorithm [91], and similarly the radius of virtual spherical lights [71] could be reduced across passes.

[38] Artists may also work around complex light transport situations, as they are inherently less controllable.

More applications include arbitrary path vertex connections [55, 67], and virtual beam lights [158].

Another approach for interactive rendering applications is to compensate only as much bias as necessary, in an approximate way [46, 157]. We will discuss this further in Ch. 6.

One final detail worth mentioning is that progressive rendering does not mesh well with certain variance reduction (blue-noise or quasi-Monte-Carlo) schemes, because the prefix of a well-stratified sample sequence may be not well-stratified at all. Indeed, as these methods presume a larger number of samples to achieve their stratification, the short prefixes implied by progressive rendering may generate large gaps in between samples [22, Sec. 5.1.2].

Chapter 6

Scalable Volume Rendering

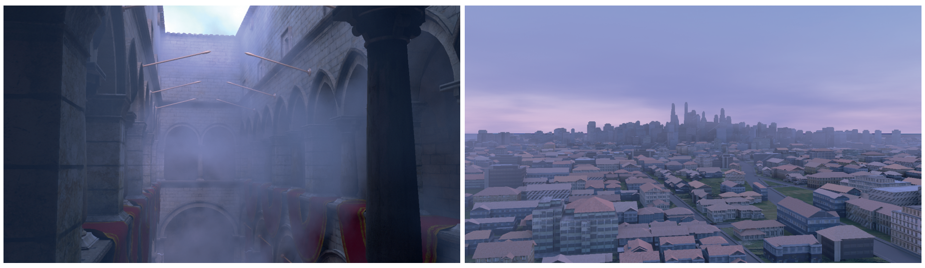

Figure 6.1: Compensation for the energy loss due to clamping virtual point light contributions, which is efficiently recovered with our *approximate bias compensation* (ABC) technique. When computing full global illumination in moderately complex environments featuring heterogeneous media and image-based lighting, such as the *Crytek Sponza* (262k triangles) and *City* (823k triangles) scenes seen, only a fraction of less than 40% of render time is devoted to bias compensation, here computed using a two-bounce ABC, while the majority of time was used for evaluating the clamped illumination from about 110k VPLs. Image taken from [46].

In this chapter,[1] we present a practical many-lights method based on instant radiosity (IR) for heterogeneous participating media. The method is highly scalable, with performance ranging from fast, interactive previews to high-quality final images. Due to the inherent difficulty of general light transport in participating media, such a broad performance spectrum is challenging to achieve for other methods, but critical for the interactive design and manipulation of realistic scenes.

As we have seen in Sec. 5.11.5, many-lights methods suffer from a weak singularity, which causes objectionable image artifacts ("splotches") when virtual point lights (VPLs) and evaluation points are too close. When extending many-lights methods to participating media, this artifact becomes even more pronounced, as the weak singularity may now happen practically everywhere in space, not just near surfaces.

As such, clamping is a vital component in producing visually acceptable images, but at the same time leads to systematic bias.

[1]This chapter is based on the publication [46].

6.1 Bias Compensation

As has been noticed by Kollig and Keller when introducing their bias compensation (BC) technique for surface light transport [121], the energy loss due to clamping is significant and non-trivial to account for. This work was later taken up by Raab, Seibert and Keller to also include BC for participating media [186]. We will outline their technique in this section.

We recollect that the clamped (generalized) geometry term G' is arrived at by bounding the regular geometry term G from above:

$$G'(x,y) = \min\left\{b, G(x,y)\right\}, \tag{6.1}$$

where $b > 0$ is the bound, which is a fixed value (but can also be more elaborately chosen according to the BRDF [121]). Due to the inverse squared distance fall-off of the (generalized) geometry term, this region can be conservatively estimated to be a sphere of radius $r = \frac{1}{\sqrt{b}}$ around x (or y by symmetry).

At a shading point x, e.g., a point along a primary ray from the camera, the method by Raab et al. first evaluates the biased solution for outgoing radiance L' via the clamped VPL contribution, and then adds and estimator for only the bias L_B, using a correction term $B(x,y)$:[2]

$$B(x,y) = \max\left\{0, \frac{G(x,y) - b}{G(x,y)}\right\} \tag{6.2}$$

$$L_B(x,\omega) = \int_{S^2} f(x,\omega,\omega_i) B(x, r(x,\omega_i)) L_i(x,\omega_i)\, \mathrm{d}\sigma(\omega). \tag{6.3}$$

When evaluating L_B, a new path vertex y is sampled and $B(x,y)$ is used instead of the geometry term, thus only accounting for the transport missing in the clamped light transport. When evaluating incident radiance at y, one again adds *both* the clamped solution at y *and* the bias compensation term at y, and this process repeats recursively until a sample y is generated that is outside the bounding region.

6.2 Approximate Bias Compensation

Our *approximate bias compensation* (ABC) was developed by more closely studying the behavior of the bias compensation (BC) technique for participating media suggested by Raab et al. [186], outlined above. While leading to unbiased results, it unfortunately imposes a high overhead, thus ruining the efficiency and elegance of instant radiosity. As an example, the unbiased solution in Fig. 5.11.5(a) took about 8 hours 50 minutes to render, about only 50 minutes of which were dedicated to computing the clamped VPL contribution.

Several aspects combine to cause this overhead: (1) BC uses recursive ray tracing; at each compensation vertex, (2) BC requires access to all VPLs at the same time, implying memory access overheads; (3) in the worst case, BC degenerates to bidirectional path tracing [36].

[2]To keep similar notation as for surface BC, we assume not only using the generalized scattering distribution f, but also a generalized ray tracing function $r(x,\omega)$, which returns the next interaction event, which may be at a surface or in the medium.

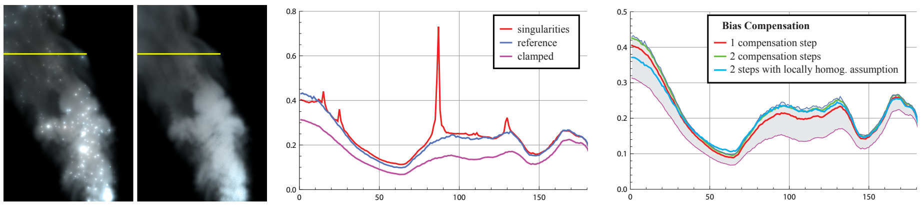

Figure 6.2: Left: we use the highlighted scanline to summarize our observations. Center: radiance plot with non-clamped VPLs (exhibiting singularities, red), clamped VPLs (purple), and ground truth (full compensation, blue). Right: ground truth vs. clamping (top and bottom boundary of the band), with 1 (red), 2 (green), or 2 locally homogeneous (blue) additional compensation steps. Image taken from [46].

In analyzing the energy recovery due to bias compensation (cf. Figs. 6.2, 6.3 and 6.6), we were able to derive several optimizations, improved sampling strategies and simplifying assumptions. Our later empirical evaluation showed these approximations to still lead to renderings very close to ground truth.

Limiting Recursion As noted in Sec. 6.1, bias compensation is a recursive process, because clamping may also happen when connecting to VPLs at compensation vertices, thus forming a path of compensation vertices. (As Raab et al. always connect to *all* VPLs from every compensation vertex, this is quite likely to happen.) We noticed, however, that path throughput diminishes quickly, as every scattering interaction removes energy (via $\sigma_s(x)$ and $f_p(x,\omega,\omega_i)$), and the exponential drop-off of transmittance (via $\tau(x,y)$) further enhances this effect. Our experiments confirmed that *most* of the clamped energy is already recovered after one compensation step, and our practical recommendation of using just two steps is visually almost identical (cf. Fig. 6.2).

Locally Homogeneous Media The original bias compensation method creates a new path vertex y by first sampling a random direction ω proportional to the generalized scattering function f at evaluation point x, and then sampling a distance s proportional to transmittance τ (using analytic inversion or Woodcock tracking, depending on the medium). If this new compensation vertex $y = x + s\omega$, however, lies outside the region where the geometry term was clamped, the vertex will have zero contribution, thus increasing variance. While this additional variance could be accounted for by, say, sampling more compensation vertices, this implies a substantial overhead.

As in a heterogeneous medium it is not possible to efficiently sample a distance proportional to transmittance while being limited to the bounding region, we instead propose to assume the medium is *locally homogeneous* within this region. Sampling a distance s within a homogeneous, spherical region with average extinction $\bar{\sigma}_t$ and radius $r = \frac{1}{\sqrt{b}}$ can be derived analytically and is realized by:

$$s = -\frac{1}{\bar{\sigma}_t}\ln(1-\xi\cdot(1-\exp(-\bar{\sigma}_t r))), \tag{6.4}$$

where $\xi \propto U(0,1)$ is a uniform random deviate. The average extinction $\bar{\sigma}_t$ can easily be obtained from, e.g., a downsampled version of the medium, if stored in a

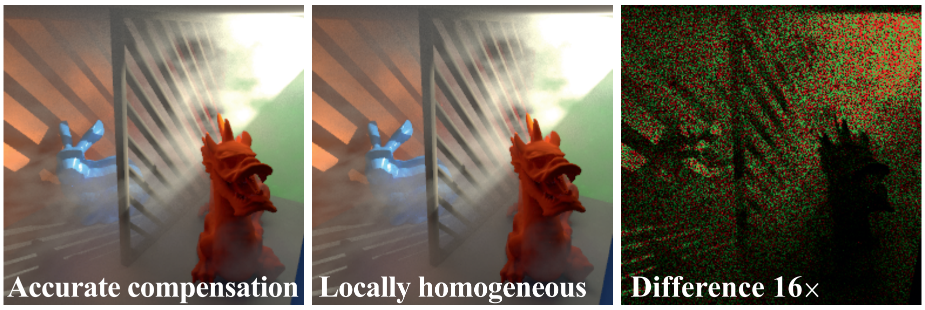

Figure 6.3: Bias compensation with accurate transmittance and with locally homogeneous assumption. Image taken from [46].

3-D texture.

While sampling distances in this fashion and still computing the correct heterogeneous transmittance $\tau(x, y)$ would still lead to unbiased results, we found that directly using the locally homogeneous transmittance hardly compromised our results (cf. Fig. 6.3). One obvious fail case would be regions with strongly varying extinction $\sigma_t(x)$; this could be detected by analyzing its gradient $\nabla\sigma_t(x)$, though.

Integration Strategies The sampling strategy used for bias compensation has a large impact on its overall efficiency. In Figs. 6.4 and 6.5, we illustrate the results of comparing four different sampling strategies at roughly equal rendering time:

- 1-to-N: Raab et al. connect *all* VPLs to a single compensation vertex, which apart from the memory access penalty also causes high variance if outside the bounding region.

- N-to-1: Lower variance can be achieved by generating more vertices while connecting each vertex to a single VPL.

- 1-to-1: A similar, but GPU-friendly approach is to generate only one vertex connected to a single VPL.

- 1-to-1 (loc.hom.): As mentioned above, using the average extinction σ_t ensures both that vertices are always within the bounding region, and also avoids the expensive evaluation of transmittance.

Omitting Visibility When sampling a new compensation vertex, one has to also perform a costly visibility test, as the sampled distance may be farther than the closest surface intersection. However, this obviously can only happen when being close to surfaces. In our experiments, we found it very hard to construct scenarios where *omitting visibility* had any visual impact (cf. Fig. 6.6). However, given that one (a) has to be near an opaque object smaller than the bounding radius, (b) in a relatively thin medium, and that (c) the VPLs' inverse squared distance fall-off further dampens light bleeding, these results are not as surprising.

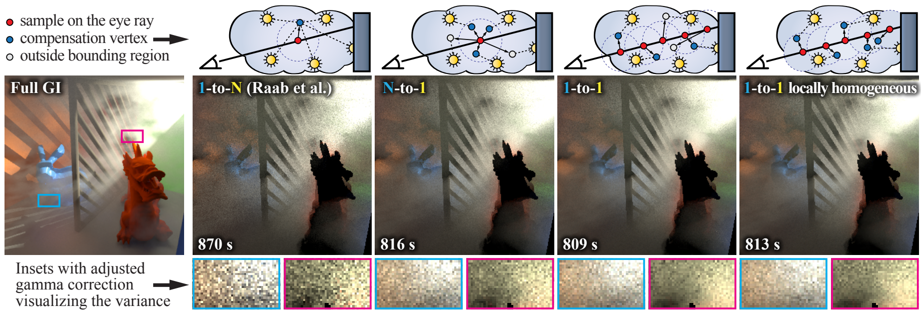

Figure 6.4: Different compensation strategies at equal rendering time, which was achieved by adjusting the number of samples along the eye ray to 3, 1, 115, and 78 for 1-to-N, N-to-1, 1-to-1, and 1-to-1 (loc.hom.), respectively. Image taken from [46].

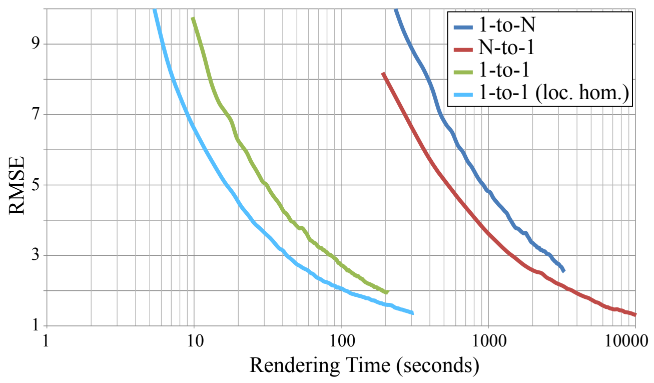

Figure 6.5: RMSE plot for different compensation strategies (see text) used in Fig. 6.4, computed against a converged 1-to-N reference solution after [186]. The 1-to-1 strategy clearly outperforms other strategies. Image taken from [46].

6.3 Results

We implemented our technique in both a custom CPU-based offline renderer and a GPU-based version. Computation is split into illumination and compensation from primary light sources, and iterative accumulation of indirect illumination from VPLs. The GPU-based version uses a variant of adaptive volumetric shadow maps [198] to efficiently perform from-point transmittance and visibility queries. Figs. 6.1, 6.7, and 6.8 illustrate final images and comparisons; please see the original publication [46] for additional results and a more thorough discussion.

6.4 Conclusion

We have presented a novel extension of instant radiosity in heterogeneous, participating media, which, because of its *approximate bias compensation*, is able to also efficiently account for the energy loss due to clamping VPLs' contributions.

With previous methods [121, 186], accurately compensating for bias dominates the otherwise fast performance of instant radiosity. As with the original instant

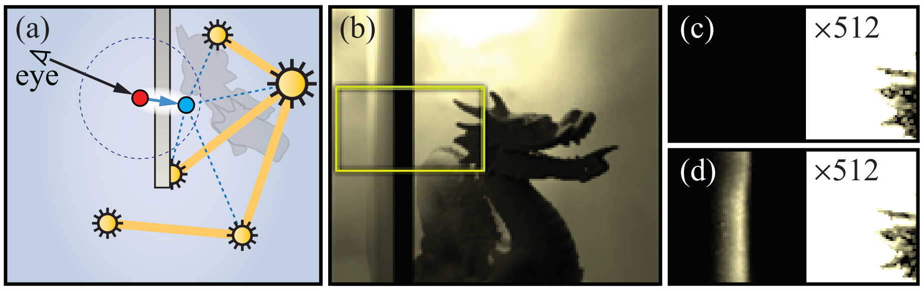

Figure 6.6: (a, b) A rare scenario where omitting visibility to compensation vertices caused artifacts. (c) The accurate computation. (d) ignoring visibility to new path vertices causes artifacts only revealed with tremendous scaling (×512). Image taken from [46].

radiosity, our method requires no precomputation and access to only a single VPL at a time, being trivially progressive and benefiting from a GPU-based implementation,[3] thus causing only fractional overhead for bias compensation.

Visually, the method achieves results very close to ground truth. While technically, our approximate method does not remove *all* bias from the image, by carefully analyzing the behavior of VPLs in heterogeneous participating media, we are able to concentrate our method's efforts to eliminate bias *at the most salient parts* of the scene.

For large scenes, bidirectional VPL placement schemes have been suggested, to avoid the overhead of VPLs that do not have any contribution to the image. We used a surprisingly simple but effective variation of [56], which stochastically rejects VPLs based on distance to camera.

Inheriting from instant radiosity (IR), our method sub-samples path space for indirect illumination, while single scattering and transmittance are evaluated at high resolution and it is thus that we can stay close to ground truth. Global transport, however, highly depends on the number of VPLs, and in dense, heterogeneous, anisotropic media one needs more VPLs than in thin, homogeneous, isotropic ones. Strong anisotropy also causes problems, as sub-sampling only is effective for smooth illumination, and otherwise a large number of VPLs is required. The interdependence between the number of VPLs and the perceived material attributes [124] likely transfers to media as well, but to our knowledge has not been studied yet.

[3]It hence could further improve from approximate visibility methods such as imperfect shadow maps [192].

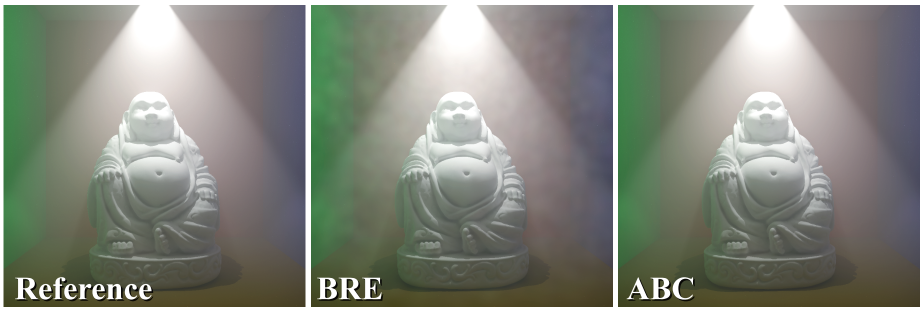

Figure 6.7: Left: Reference CPU solution with full bias compensation [186] computed in 31 hours. Middle/right: beam radiance estimate (BRE) [94] with 1 million volume photons (135 seconds) vs. GPU-accelerated 2-step ABC with 7887 VPLs (125 seconds). The BRE image shows the typical artifacts of undersampled illumination in photon mapping-based methods, whereas our method is amenable to a straightforward GPU implementation and able to give smooth results almost identical to the reference, in roughly equal render time. Image taken from [46].

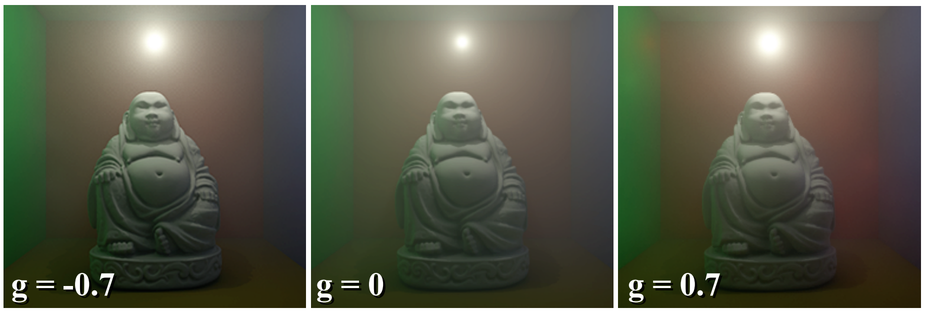

Figure 6.8: The *Buddha* in a homogeneous medium ($\sigma_s = 0.075$, $\sigma_a = 0.001$) with anisotropy parameter g of the Henyey-Greenstein phase function varying from backward, to isotropic, to forward scattering. Image taken from [46].

Part III

Visualization and Artistic Editing

Chapter 7

Artistic Editing Approaches

Physically based rendering (PBR) which we have extensively covered in Ch. 5, has important applications in feature-film, architecture, medical, and other industries, as well-designed images are an important tool for conveying information. Specifically, the availability of efficient (commercial) PBR systems with progressive previews has promoted rapid adoption of PBR standards in the feature-film and gaming industries [22, 123, 139]. As such, lighting and shading artists are now increasingly familiar with technical PBR concepts and can communicate in the same "language," allowing them to more rapidly collaborate and iterate towards a common artistic goal, all while adhering to a PBR workflow composed of unified concepts. These PBR workflows include the design and placement of light sources with complex shapes and emission profiles, the iterative design of realistic local reflectance models (BSDFs), extended models of subsurface scattering, and intricate indirect illumination effects like color bleeding and caustics.

In practice, however, authoring the *input data* that is ultimately fed into PBR methods is a tedious task usually undertaken by large groups of highly-trained individuals. Hence, numerous methods and approaches for artistically adjusting the appearance of rendered images, via editing of materials and lighting, have been suggested in the computer graphics (CG) literature, to both make this task more manageable and also find new ways of artistic expression. Intimately connected to these methods are the interaction paradigms employed (and the rendering techniques used).

This chapter[1] will give a comprehensive review of artistic editing methods for lighting and material, also entailing *whole appearance* approaches. Figs. 7.1 and 7.2 give simple examples that already illustrate some of the challenges associated with appearance editing in PBR.

7.1 Introduction

Synthesizing realistic images is among the longstanding goals of computer graphics, and its ambitious nature is evidenced by the advancements of the field towards realism with still a significant number of open problems. The acquisition and editing of detailed geometry, its animation, the careful modeling and reproduction of real-world material and lighting profiles, and the efficient simulation of physically accurate

[1]This chapter is based on our previous publications [201, 202].

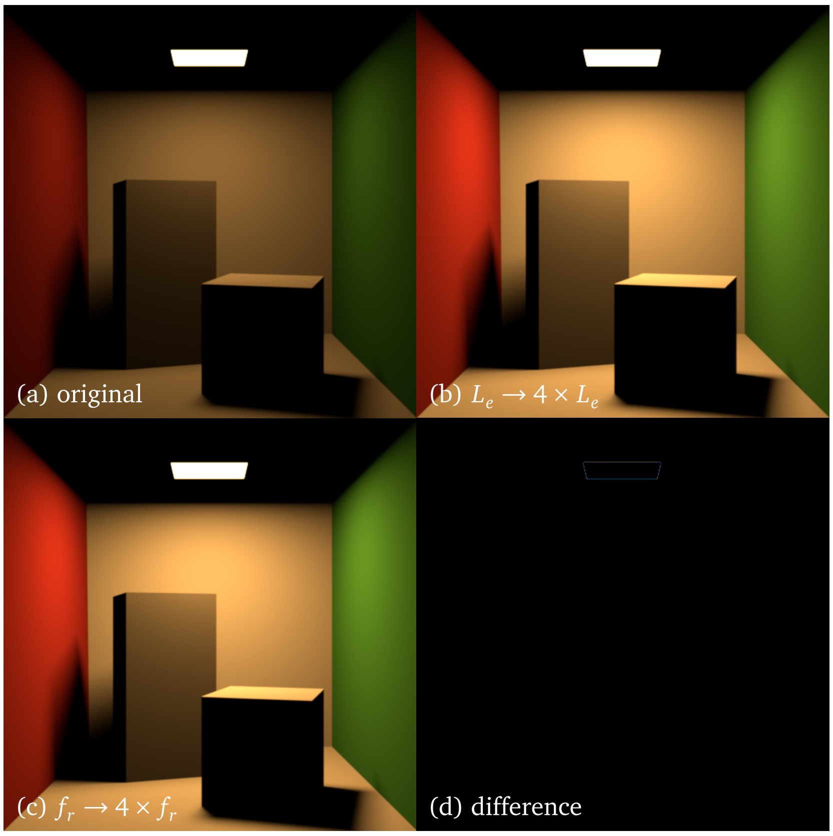
(a) original
(b) $L_e \rightarrow 4 \times L_e$
(c) $f_r \rightarrow 4 \times f_r$
(d) difference

Figure 7.1: The effect of appearance editing in the classic Cornell Box scene, (a) rendered with direct illumination in *Mitsuba* [84]. The same edited image can be obtained by either (b) manipulating light parameters, say, by increasing emission or (c) manipulating material properties of the scene, say, by increasing reflectivity. (d) The only—barely discernible—difference is the sampling of bright edges around the light source, due to imperfect filtering of the sudden step in brightness.

light transport are still in need of robust solutions. But, as our field progresses, so do its goals: while realistic image synthesis remains an important challenge, so too does the ability to *design* a (potentially realistic) image that conveys an explicit mood or information to the viewer.

One of the aspects at the core of scene design is defining the appearance of objects, which comes from the interaction of surface materials and scene lighting. Appearance design is the process by which artists edit material and lighting properties in order to achieve a desired look. In general, this is a complex and laborious process, since artists are manually solving an under-constrained inverse problem: given a desired appearance, determine the material and light settings to achieve it. In fact, even for simple scenes and highly-trained digital artists, appearance design may take several hours. Furthermore, in cases where the design goals cannot be obtained in the confines of physically accurate simulation models, more flexible artistically motivated models need to be developed. Many different approaches, ranging from physically based to purely artistic, have been proposed to intuitively edit the appearance of

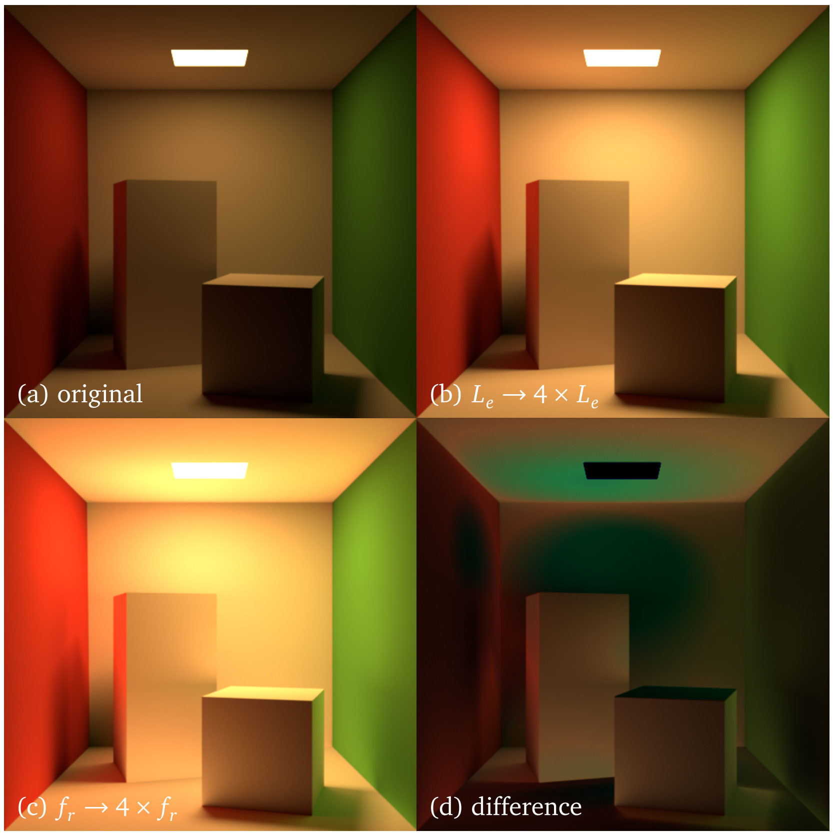

Figure 7.2: The effect of appearance editing on global illumination. (a) The same scene as in Fig. 7.1, rendered with global illumination, and (b, c) applying the same edits. (d) Note how multiple light bounces lead to significantly different results, due to the global, nonlinear influence of the BRDF; thus, greater care must be taken to achieve a local edit that will not affect other parts of the scene as well.

individual objects as well as entire scenes.

In the following, we present a summary of the state of the art in artistic editing of lighting and material that includes the following topics:

- *lighting design*: the editing of lighting parameters to define a final scene appearance, which is fundamental to computer cinematography, architecture visualization, etc.;
- *material design*: the definition of the reflectance properties of a surface or the scattering properties of materials, ranging from whole surface changes to precise adjustment in textured regions;
- *whole appearance design*: the coupled editing of the interaction between surface materials and scene lighting, when it may be difficult to segment and treat separately;[2]

[2]These methods will still be listed under lighting or material design, depending on our subjective assessment what is their main intent.

We organize prior work along two axes, defining first *what* is edited or manipulated, and second *how* these elements are edited, including the interaction paradigms they rely on. We also provide an overview to the methods covered in this chapter, providing a quick way to assess their usefulness for different practical scenarios (see Tab. 7.1, Tab. 7.2, and Fig. 7.3).

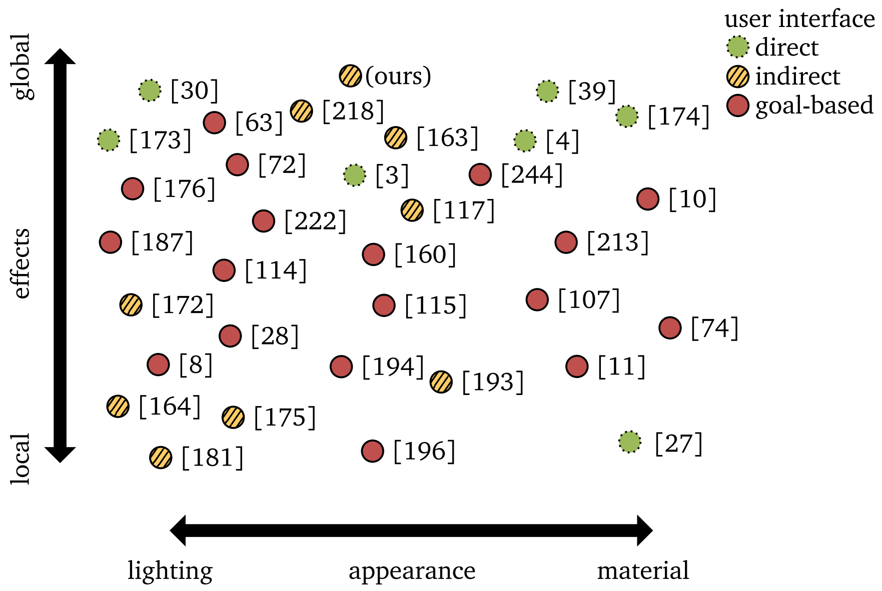

Figure 7.3: Subjective categorization of some appearance editing methods, according to the criteria discussed in this chapter, as further detailed in Tabs. 7.1 and 7.2.

7.2 What is Appearance Design?

Appearance design is a fundamental task at the tail end of digital content creation: given objects' surfaces and their relative placement in space and time, the goal of appearance design is to define the look of the final images that meets specific stylistic or artistic requirements.

The *appearance* of an image depends on complex local and global interactions of light in a virtual scene. Light emitted from light sources travels in the scene, and is subsequently reflected, transmitted or absorbed locally at the surfaces of the objects, until it finally reaches an image sensor. When participating media are present, light can also be emitted, scattered, and absorbed in the volume surrounding surfaces. This combination of global transport and local interactions repeats indefinitely until light reaches a state of equilibrium.

Given this light transport process, it is clear that both the *initial lighting* emitted from sources, as well as the *local material* interactions, play a significant role in the final appearance of a scene. As such, modifying the initial *lighting* state and/or the local *material* reflectance behaviors is a simple way to affect both the local and global appearance of the final image. In general, the final image appearance relies on several controllable *appearance parameters*:

- the position, orientation, and emission profiles of light sources, ranging from simple point sources to realistic area and environment illumination;
- the camera parameters, including position, framing, aperture, lens model, shutter time, etc.;
- the materials that define the potentially spatially-varying shading response (e.g. via BRDFs, shaders, node-based networks, etc.) of each object;
- the light transport simulation algorithm and its settings.

As we have seen in Ch. 5, in the context of PBR, final images are computed by ultimately solving the *light transport equation* (LTE), which implicitly captures the appearance of a point x as the incident radiance $L(x_0 \leftarrow x_1)$ towards a viewer at point x_0 as (cf. Eq. 5.28):

$$\begin{aligned} L(x_o \leftarrow x) = {} & \underbrace{L_e(x_o \leftarrow x)}_{\text{lights \& camera}} + \\ & \int_{\mathcal{M}} \underbrace{f_r(x_o \leftarrow x \leftarrow x_i)}_{\text{materials \& camera}} \underbrace{L(x \leftarrow x_i)}_{\text{transport}} \underbrace{G(x, x_i) V(x, x_i)}_{\text{geometry}} \mathrm{d}x_i, \end{aligned} \tag{7.1}$$

where $L_e(x_o \leftarrow x)$ is radiance emitted from light sources at x, $f_r(x_o \leftarrow x \leftarrow x_i)$ is the BRDF, modeling local material interaction with incident light $L(x \leftarrow x_i)$, and $G(x, x_i)$ and $V(x, x_i)$ account for the scene geometry's relative orientation and its relative visibility, respectively. The integration domain $\mathcal{M}$ is the set of *all* surfaces in the scene, hinting at the global nature of light transport. Similar effects are captured by the volumetric LTE (Eq. 5.46), but additionally account for the scattering (and emission) of light caused by matter in between surfaces, which can be thought of as another type of material.

The recursive definition of radiance means that an object's appearance also depends on the appearance of all other points in the scene. Indeed, we note that the *appearance parameters* (i.e. material and lighting) affect each term in the image formation process. In any nontrivial scene, predicting the final appearance as a result of directly editing these parameters quickly becomes intractable for even the most skilled and experienced artists.

There have been efforts to catalog the appearance of highly diverse objects from photographs with the aid of *crowdsourcing*, for applications such as surface re-texturing and material and image browsing [9].[3] From the point of view of appearance design, this can be seen as a useful database for retrieving appearances of already-existing real-world objects as a source of inspiration, but the key responsibility of actually selecting and editing (i.e. *designing*) the appearance of a specific scene remains on the artists.

In our discussion, an *appearance design approach* is a semi-automatic process for editing the final appearance of an image or animation sequence that abstracts the task of determining suitable settings of the lighting and/or material settings in a scene. Specifically, any such approach will take some higher-level input specification of the appearance edits desired by the user, and then automatically computes the lighting (Sec. 7.4) or material (Sec. 7.5) settings, or both, in order to best meet the user's requests.

[3]Incidentally, photographs are just a tiny portion of the light field, viz. radiance, measured with a camera.

Challenges & Complexity Appearance design tools inherently deal with different rendering challenges than standard rendering. In a typical renderer used for generating animations, mostly the camera, geometry, and, to a lesser extent, lighting change, while the appearance of materials remains mostly static during a shot. Furthermore, though lighting and material may change, they have a predefined evolution. This is fundamentally different from the requirement to dynamically explore the entire parameter space during appearance design.

Historically, the strategy to bridge this gap has been to perform some precomputation which is then cached using more flexible intermediate representations. Typically, the system first enforces certain constraints, e.g. fixed camera, fixed lighting, or fixed materials, and caches the possible space of parameters for the remaining free variables. The choice of what is cached and its representation varies significantly across the proposed techniques, and is also highly dependent on the provided editing functionality.

Relighting systems' primary function is to allow interactive editing of the lighting parameters while typically keeping the scene and materials static. Early examples include parameterized ray tracing [155], ray trees [14, 206], and the G-Buffer approach [57, 197]. The Lpics [176] and Lightspeed [187] systems also fall within this category. Direct-to-indirect transfer techniques [72, 133] exploit the ability to compute direct lighting efficiently and leverage a possible precomputation to extend this to indirect illumination. Most of these methods gain efficiency by exploiting the linearity of light transport and they often capitalize on the assumption that camera movement occurs much less frequently than shading changes.

Although it may initially seem conceptually similar, material editing is inherently different than relighting. In contrast to relighting, BRDF editing is fundamentally nonlinear when global illumination is considered (cf. Fig. 7.2). In particular, editing n BRDFs in a scene with d light bounces leads to an n-variable polynomial of degree d [10]. Unfortunately, representing this explicitly is only practical for a small number of bounces. Several researchers have investigated this problem for both surface BRDFs [10, 11, 222], and more recently for editing participating media parameters [74, 213, 247].

Relighting, and to some extent material editing, systems have exploited a vast set of techniques developed in the *precomputed radiance transfer* (PRT) literature [2, 20, 108, 122, 150, 151, 188, 210, 221, 239]. These techniques typically exploit the linearity of light transport and the fact that light (transport) is often sparse in a suitably chosen basis space (e.g., frequency or wavelet domain). In return for the efficiency gained through precomputation, these methods typically restrict the lighting (e.g., environment only), or material properties (e.g., diffuse only).

Although PRT techniques can provide interactive feedback when editing a specific set of parameters, once the parameter set changes, a new, expensive precomputation must be performed. For interactive design, this can lead to slow interaction times, for instance, a level designer for a game must wait for an overnight simulation to see interactive lighting changes when the scene geometry is modified. The recent Modular Radiance Transfer [135] approach addresses this challenge by trying to decouple the precomputation from the scene, but introduces approximations.

Common Non-physical Rendering Models We will briefly discuss a few models commonly employed in computer graphics that are a deviation from the "physical correctness" we explored in Ch. 5. Historically, many of these models have appeared

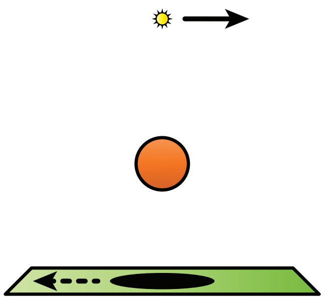

Figure 7.4: Direct vs. indirect manipulation of a point light source. With direct manipulation, the user moves the light source (solid arrow) and the shadow follows accordingly (dashed arrow). *Direct interfaces* are trivial to implement, but quickly become counter-intuitive in complex scenes. With indirect/goal-based manipulation, the user moves/sketches the shadow (dashed arrow) and the system solves for the new light source position (solid arrow). Image taken from [202].

long before physically based rendering (PBR) was a common term. They are still commonly applied in real-time rendering [1], even though physically based shading (PBS) has changed terminology, concepts, and workflows in that field as well [139, 140]. Christensen and Jarosz [22, Chapter 4] also discuss further practical, but non-physical "hacks" in the context of production rendering.

Regarding lighting, a common artistic control is to apply a custom fall-off curve to point lights, which violates the correct inverse squared distance behavior. Apart from shaping the brightness of different regions based on distance, another motivation is that potentially expensive light shaders have to be evaluated only in a small part of the scene if their contribution drops to zero quickly. Similarly disabling lights for certain render passed or objects is also a common way to quickly remove unwanted lighting [8]. We will discuss more elaborate non-physical lighting further in Sec. 7.4.

Shading equations are the traditional approach to implementing custom materials (and lighting), and while very flexible and powerful, the inherent coupling of material, lights, and light transport make it all too easy to violate physical constraints. Shading equations are also useful in replacing otherwise expensive operations such as shooting rays by reflection maps. While these techniques have their purpose in solving practical rendering problems, they have more and more fallen out of favor [21,22] having been replaced by more principled approaches. Nonetheless, material design approaches (Sec. 7.5) may purposefully (re-)introduce non-physicality to achieve a certain look.

7.3 Interaction Paradigms

Artistic editing is inherently interactive.[4] Hence, beyond simply classifying methods as "lighting design" or "material design", the three common interaction paradigms employed by various methods serve as a good secondary categorization [112, 114]:

- *Direct* interfaces expose light/material parameters, such as positions and surface colors, to artists. This is a popular interface in commercial software and, while easy to implement, direct interfaces are neither efficient nor intuitive,

[4]Or at least, it should be [44].

since the relationship between final appearance and these parameters is often unpredictable, barring very simple scenarios.

- *Indirect* interfaces let users modify certain aspects of appearance, e.g., shadow positions or material contrasts, and the system computes the necessary rendering parameters such as light positions (cf. Fig. 7.4).

- *Goal-based* interfaces allow artists to define the rendered colors directly, for example by painting, while the system solves a complex and typically non-linear optimization to determine the rendering parameters (cf. Fig. 7.5). This goal-based approach can be thought of as an extreme version of indirect interface; however, both user inputs and parameter outputs are rather abstract and imprecise, and the search space is usually vast.

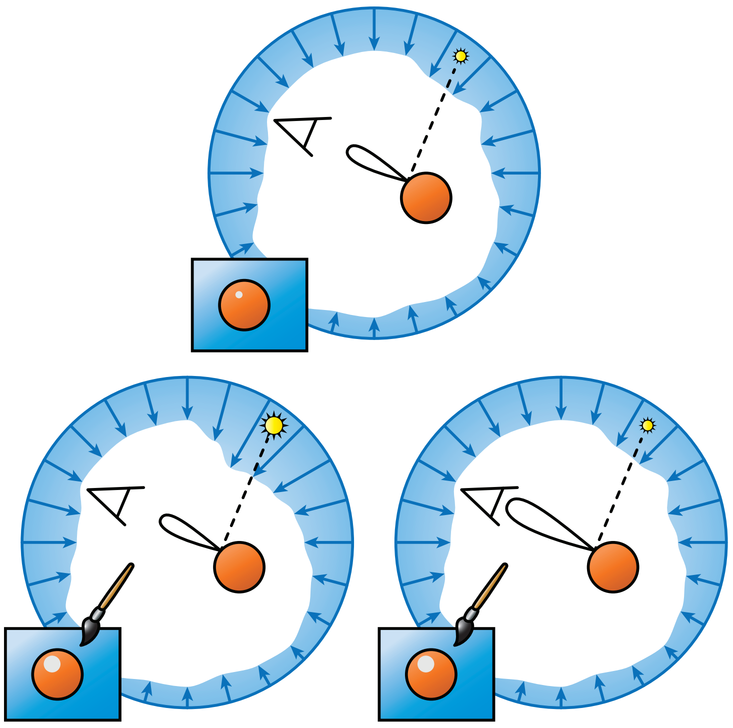

Figure 7.5: Goal-based interaction. Top: original scene illuminated by an environment map; the inset to the bottom left depicts a rendering which shows a highlight. A *goal-based interface* allows the user to paint a new highlight. Bottom left: After painting the desired appearance, the system solves for new light parameters, e.g., a brighter area around the sun in the environment map. Bottom right: After painting the desired appearance, the system solves for new material parameters, e.g., by modifying the BRDF lobe. Image taken from [202].

Investigation of these paradigms involved both the effectiveness of user interaction [112, 114], as well as the selective application of edits to complex materials and lighting [3, 172, 174].

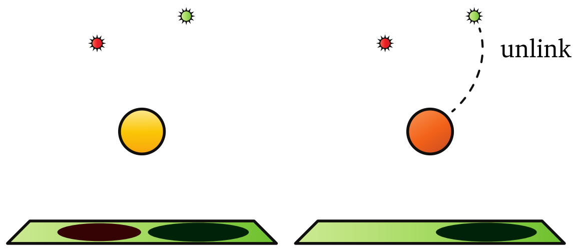

Figure 7.6: Light linking. Left: Original configuration with two colored point light sources and two objects (a sphere and a ground plane). Right: The green point light (upper right) is unlinked from the sphere object, hence also casting no shadow onto the bottom plane anymore. Image taken from [202].

7.4 Lighting Design

Lighting design focuses on modifying the parameters of lighting models under fixed geometry and material conditions. These models can be categorized by the complexity of effects they support, namely direct illumination and shadows from point and directional lights [175, 181], area- and image-based (direct) illumination [164, 172], and (full) global illumination including diffuse interreflections and caustics [63, 194]. Finally, some systems [74, 117, 160] allow manipulation of volumetric effects.

Due to otherwise high computational demands, most works have focused on direct illumination. Previous lighting design works leverage sketch-, click-and-drag, and paint-based editing concepts. Kerr and Pellacini's studies [112] stress that, although painting interfaces are useful in some scenarios, typical editing operations can be better achieved using *direct* and *indirect* manipulation. We follow this distinction in our discussion below.

7.4.1 Directly Controlled Lighting

We first focus on methods to directly control *lighting features* (not to be mistaken with direct lighting parameter control). While indirect, goal-based interfaces allow artists to roughly sketch the desired appearance of lighting features and let the underlying system solve for the model parameters, sometimes more direct control over the illumination, e.g. to exactly (dis)place features, is beneficial.

A straightforward technique to directly manipulate lighting is what is commonly called "light linking" [8]; here, users can select which light sources affect which objects in the scene, allowing to explicitly set shadow caster/receiver relations among them (see Fig. 7.6).

Apart from directly activating and deactivating light sources, the simplest and arguably most intuitive kind of direct interaction with the scene illumination normally arises from click-and-drag interfaces. For example, the system presented by Ritschel et al. [194] is an object-space appearance-guided editing tool for manipulating shadows, caustics and indirect light with a custom interface that couples space warping effects for reflection and shadowing with inter-object markups for indirect light exaggeration. More recently, Mattausch et al. [137] begin with physically based shadows and provide a tool to artistically edit the shadow boundaries in a scene akin to free-form curve editing. Consistency within the scene is achieved by computing and using shadow volumes [32] to render the edited shadows.

In designing direct user interfaces and interaction paradigms for lighting design, one important aspect is that—in contrast to materials and scene geometry—the illumination (and thus appearance) is only a by-product of the rendering process and usually not explicitly hand-authored by artists. Therefore, lighting design methods for non-trivial scenarios have to introduce abstractions and visualizations of the underlying light field, which is a five-dimensional, complex function and an effective visualization thereof is difficult [189].

That said, if the transport is limited to, e.g., a fixed viewing direction (as in cinematic lighting preview systems [72, 176, 187, 222]) or direct lighting from a finite set of directions, then good visual mappings can be found. For example, Kerr et al. [114] control spot or directional light sources using guided visualizations of the underlying user-deformable lighting volume. Another editing approach is lattice-based deformations, as in Obert et al.'s work [164]. Here, a factored representation of visibility is efficiently stored in compressed matrices, enabling interactive shadow editing under environment illumination.

As industry-leading studios adopt physically based rendering (PBR) in their art generation pipelines, the usefulness of simple manipulation approaches that address direct effects without considering underlying PBR concepts and constraints decreases. Our own work falls into this new context, but we will defer discussion of it to Ch. 9. Since the original publication of our method [200], Subileau et al. [218] introduced a formulation of *ray portals* in path space to achieve similar effects to ours. Günther and colleagues [63] introduced a smooth interpolation of caustic photons which allows to directly specify target light distributions in a goal-based fashion (e.g., with image textures).

The work by Ritschel et al. [194] is a flexible tool for manipulating shadows, caustics and indirect illumination with a click-and-drag interface. Tabellion and Lamorlette [225] use shader falloff-function editing on the hue of indirect color bleeding effects. Similarly, Nowrouzezahrai et al. [160] edit the underlying physical processes of volume rendering.

Lastly, goal-based approaches have also been developed using painting methods in high dynamic range [26, 28], to sketch both highlights and directly paint and modify environment illumination.

7.4.2 Indirectly Controlled Lighting

Lighting design may also be approached in a more indirect fashion, by exposing a set of intuitive, abstract parameters to users, which an underlying system them uses to solve for concrete light parameters using inverse image formation models and (non-linear) optimization methods [8, 30].

Poulin and Fournier [181] and Pellacini et al. [175] infer light positions in the context of a direct illumination model, allowing users to sketch desired shadow or highlight boundaries in a final rendered image. Light direction and emission profiles can be optimized in a similar spirit to shadows, starting with painted [203] or higher-level [109] user annotations of initial renderings, searching the physically feasible solution space. Pellacini et al. [173] use painted color, light shape, shadows, highlights, and reflections as an input for solving for more complex lighting and reflectance parameters.

Some more recent work allow users to mark, identify and isolate shadow features, automatically inferring (direct) environment lighting from its coupled relationship to all-frequency shadows [164, 165, 172]. Ritschel et al. [193] expose a custom user interface enabling users to precisely place mirror reflections, using the inputs to smoothly deform reflection directions, at real-time speeds. Bousseau et al. [13] optimize illumination from environment maps to improve the *perception of material characteristics* without the need for user intervention, which can be interesting e.g., for product design, where isolated key objects are presented in an otherwise simplified background context. This work is an example of editing the coupled appearance of material and lighting.

In the context of more global illumination effects, Nowrouzezahrai et al. [160] generalize photon beam primitives [92] to non-physical effects, allowing artist-driven sketching of heterogeneous volumetric densities with art-parameterized shading and emission models. Obert et al. [162, 163] present an appearance manipulation framework for global illumination, allowing artists to change the intensity and color of indirect illumination. An optional labeling interface allows users to identify and alter sender/receiver relationships while maintaining certain visual constraints, in order to achieve plausible rendering results, while implicitly also affecting perceived material characteristics.

7.5 Material Design

Material interactions are *local* interactions that modify the distribution of light at a surface or in a volume. Under this definition, material models include (a) spatially-varying BRDFs and BSDFs that model the local reflection and transmission profiles at a surface (cf. Sec. 5.4), (b) BSSRDFs that capture subsurface scattering effects (cf. Sec. 5.10), (c) the scattering coefficients and phase function profiles in participating media (cf. Sec. 5.9), or (d) some implicit or explicit combination of these properties. Note that while (a)–(c) relate primarily to physically based rendering, non-physical or *artistic* models of local light interaction are also valid material descriptors in the context of artistic material design and captured by category (d).

Usually, material design tools and approaches enable editing the final appearance of a scene *while holding scene geometry and lighting fixed*. As with lighting design tools, interfaces can be categorized into direct, indirect, or goal-based approaches. Unlike lighting design, the development of sophisticated material design is a more recent development and the literature is relatively sparse. We can imagine various reasons for this.

First, changing appearance through material editing is mathematically more complex than light editing; whether with direct or global illumination, changing light sources is always linear with respect to final appearance, but changing materials is non-linear (cf. Fig. 7.2).

Second, traditional workflows first assign materials to objects, based on their "physical composition" and only later illuminate them in the context of the scene.[5] These two tasks may even be split into separate "shading" and "lighting" departments, with artists specializing in one or the other.

Finally, the number of mathematical light transport models is much smaller than the number of material models, so the former have been researched much more in depth; the number of material representations has increased steadily over

[5]We presume this is because in the real world, a physical object and its material are intertwined to the point of getting cognitively fused into *one and the same thing*, making arbitrary change of material somewhat of an alien concept.

time [15, 40], with new methods for surface, subsurface, or volume materials appearing more often. Some recent approaches include basis-space reflectance models for interactive shading [238], new microfacet distributions that better fit real-world BRDF data [237], and new volumetric distributions covering a wider range of subsurface reflectance behaviors [86]. This wide gamut of models makes it more difficult for any concrete material design approach to be proposed and adopted.

Direct Editing Changing, say, the reflectance of a Lambertian BRDF is a form of direct interface, and one of the simplest forms of material design. This approach, however, quickly becomes counter-intuitive with more complex BRDF models, as they usually expose a large number of parameters. As this implies more degrees of freedom and a larger search space, the effect of parameter changes become harder to predict. Furthermore, material appearance may also depend on geometry [228], which further masks parameters' perceived effects. Another problem is that inconsistencies exist between different material models with respect to parameter value ranges and interpretation, as for example with the "Phong exponent" which controls glossiness of specular BRDFs, compared to "surface roughness" in other models.

The gap between material parameters and final appearance is even greater in participating media. The absorption coefficient, for example, defines the color of light that is *removed* from the transport, which is opposite to what reflectance parameters specify, and is hence better handled with a model-driven approach, e.g., inferring coefficients from user-sketched color constraints [160]. Predicting appearance from phase function parameters is similarly difficult, suggesting remapping for effective translucent material design [58].[6]

Perceptual Sliders Kerr and Pellacini [113] evaluated (by study) different material editing paradigms with the goal to identify the one with the best workflow for novel users. Their results indicate that editing physical BRDF parameters directly or through *perceptually linearized* sliders works better than image-based material selection; it is unclear whether this was caused by limitations of the underlying image-space tool tested. Despite this, their evaluation clearly shows that the user interface exposed plays a crucial role in user effectiveness of completing material design tasks.

Model-Driven Approaches Similar to indirect, goal-based lighting design, more sophisticated material design methods work with user-specified constraints on appearance, which are then used in an underlying optimization process to *automatically* derive best-match material parameters. Pellacini and Lawrence [174] present a system where users sketch appearance constraints on the image in order to infer, warp, and transfer appearance from different spatially- and temporally-varying reflectance data sets. Song et al. [213] edit heterogeneous subsurface scattering, starting from the simplified diffusion model of subsurface scattering [97], but approximate the BSSRDF as a product of two blur kernels which can then be edited using image-based and other techniques. Sadeghi et al. [196] present an artist-driven model for hair rendering under complex light transport scenarios that exposes more intuitive controls and is build around a (simplified) high-performance rendering model which allows for fast iteration times. Of note, their model explicitly allows to violate energy conservation, as it is sometimes required for a certain look. Obert et al.'s [163]

[6]For illustration, can you tell from Fig. 6.8 whether the medium is forward- or back-scattering?

painting interface for indirect illumination, discussed in Sec. 7.4.2, can also be seen as material design, as the user-editable *transfer functions* implicitly encode both incident lighting distribution and reflection from the surface. Colbert's thesis [26] discusses several techniques based on BRDF lobe sketching under environment lighting, including spatially varying BRDFs and appearance matching. BRDF-Shop [27] allows to sketch spatially varying BRDF distributions coupled with an interactive rendering tool for environment lighting. Khan et al. [115] present an image-based method to estimate indirect illumination from a single image, and use this to enable plausible material changes. Muñoz et al. [148] approximately capture BSSRDFs from single images and then fit them to a smooth diffusion model, which can be seen as a useful model for prototyping new BSSRDFs.

Data-Driven Approaches Several data-driven methods exist in the literature. An and Pellacini [3] formulate material editing as an optimization problem, letting users sketch rough appearance constraints and then search for similar patterns in unedited data, which are smoothly warped/blended with edits. Their method supports both (high dynamic range) images and higher-dimensional spatially-varying reflectance. Dong et al. [39] solve for spatially varying BRDFs from a simple reflectance model, an image of a directionally lit surface patch, and user markups. They decompose results into a product of shading and reflectance, which then can be edited by users separately. An et al. [4] warp measured material reflectance data according to template reflectance behaviors, thus quickly prototyping complex behaviors. Kautz et al. [107] generate large-scale bidirectional texture functions (BTFs) from a combination of microgeometry and reflectance profiles, in a data-driven process based on existing measured BTF data. Similarly, Iwasaki et al. [83] allow to edit bi-scale BRDFS, which are the product of a microfacet normal distribution and an analytic small-scale BRDF model. As they represent both components using spherical Gaussians, they support highly glossy reflection, and are able to render results in real-time, thus facilitating interactive design exploration.

Ben-Artzi et al. [10,11] represent per-pixel outgoing radiance (with fixed camera) as a combination of basis functions based on the scene's materials. After precomputing this representation, users can edit materials by simply reweighting them in the final per-pixel expression. Sun et al. [222] follow a similar idea, but perform the decomposition in object space. While limiting themselves to only one to three bounces of illumination, their representation supports interactive view changes. Wu et al. [244] combine micro-and macro-scale editing for different levels of detail, and later [245] showed how to perform inverse bi-scale BRDF design, i.e., how to derive microgeometry distributions and BRDFs from given reflectance, leveraging precomputed libraries to efficiently find representative exemplars.

7.6 Measuring Interface Effectiveness

The previous sections covered works with a wide variety of interfaces for lighting and material design. In 2009, Kerr and Pellacini [112] established a user study methodology to measure user interface effectiveness, which has been adopted in several other studies [89,106,112,113,168,189] and which we will briefly summarize here, not the least as a hint where future work in the field may lead.

First off, they focused on user interface paradigms, not on individual implementation differences. As such, they tried to keep interfaces consistent for a given user study, by e.g. keeping window layout, keyboard accelerators, etc. the same for all methods tested. Furthermore, features not common to all methods in a trial were removed. Another practical constraint was to focus on real-time rendering methods only, under the assumption that instantaneous feedback is vital. This also limits geometrical scene complexity, but not necessarily that of appearance (in the form of light fields, which can quickly become complex even in simple scenarios).

A major concern in measuring interface effectiveness is to record precise, objective observation, but, due to the targeted user base, at the same time leave enough freedom for artistic exploration. They suggest to achieve this by splitting studies into two parts: In *matching trials*, users are given reference images and are asked to reproduce them exactly with the given user interface, with the (hidden) goal of objectively measuring the ability for precise control. The images and initial parameter settings are based on the researchers' assessment of typical, real-life use cases. In *open trials*, users are only given a desired look to be achieved, via a vague, verbal description of the task at hand. During these trials, performance is recorded to later assess users' workflows.

During studies, both objective and subjective criteria are collected. Objective criteria include time to completion and final image error; user interactions such as button presses (or full video capture) are only recorded for later cross-validation of results. Subjective criteria are assessed via questionnaires, asking participants to rank methods in different criteria.

The collected data is then analyzed to find statistically significant trends using various methods, with ANOVA [52] being most popular. The analysis results give insights into which interface users prefer for what appearance editing tasks. It is also useful to correlate recorded data sets, e.g. subjective ratings are found to be higher when users achieved lower error in shorter time. Finally, the recorded performances are also analyzed manually to discover patterns in user workflow, though due to the usually small sample of test subjects, these results are not statistically significant and only informal, but may point to interesting directions for future work.

Most studies are performed with novices, as they are (arguably) the majority of users, and also the most likely to dramatically benefit from interface improvements. This is not to say that experts wouldn't benefit from these methods; but this has not been tested formally yet, more relying on informal feedback.[7] This focus on novices enforces a delicate balance on task design, where one has to trade off complexity of real-world usage scenarios with user fatigue and the question whether tasks are even achievable. Usual tasks are designed to be completed within a few hours by participants.

In review, the published literature mostly answered concrete questions on effectiveness of existing interfaces, but more interesting would be to discover new workflows, and make statistically significant statements and categorizations about them. This, however, requires studies of grander scale, likely modifying experiment design and also increasing analysis effort, and thus seems to not have been explored, yet.

Finally, there is still no conclusion what interface is overall *the best*, i.e., able to control all possible lighting–material interactions in a unified manner, including environment, area, and point lights, spatially varying opaque and translucent materials, and not the least, participating media, at the same time. While more

[7]One reason may be that for practical reasons, novice users are simply easier to find in bulk, as confirmed by personal experience.

advanced methods focused on global light transport, are promising in this respect, their complexity also makes adoption in the real world more of a challenge.

It is even unclear whether there exists one such optimal user interface for all effects. Studies found in the literature indicate that the different interfaces tested are more or less effective *depending on the task at hand*. As such, our own work (discussed in Ch. 9) offers complementary tools and approaches to light transport selection and editing, as well.

Table 7.1: Overview of different techniques, grouped by primary goal of edits (lighting or material or combined appearance), as well as complexity of light transport (surface graphics only or participating media). For the *Interaction* columns: *Paradigm* refers to the kind of interface provided, according to Sec. 7.3; *Scope* states whether the editing/interaction has only local effects or also includes subsequent global effects; *User Interface* outlines the type of interface provided (parameters refers to traditional parameter tweaking). The *Manipulation* column states which part of the scene description is being modified. Abbreviations used: Global Illumination (GI); Non-Photorealistic Rendering (NPR); Image-Based Lighting (IBL); Bidirectional Reflectance Distribution Function (BRDF); Spatially Varying BRDF (SVBRDF) = Bidirectional Texture Function (BTF); Temporally & Spatially Varying BRDF (TSVBRDF); Bidirectional Scattering Surface Reflectance Distribution Function (BSSRDF).

		Interaction			Manipulation
Class/Method		Paradigm	Scope	User Interface	
Lighting					
Lights from Highlights	[181]	indirect	local	click & drag	direct lighting
Lighting Controls	[8]	direct	local	parameters	direct lighting
Interactive Shadow Design	[175]	indirect	local	click & drag; constraints	direct lighting
Lpics	[176]	direct	global	parameters	cinematic relighting
Direct-to-indirect Transfer	[72]	direct	global	parameters	cinematic relighting
Lightspeed	[187]	direct	global	parameters	cinematic relighting
Dynamic BRDF Relighting	[222]	direct	global	parameters	cinematic relighting
Goal Based Lighting Design	[30]	goal-based	global	place light sources	surface GI
Lighting with Paint	[173]	goal-based	global	painting	surface (NPR; GI)
BendyLights	[114]	direct	global	manipulators	surface GI
HDR Painting	[28]	direct	local	painting	IBL
All-Frequency Shadow Design	[164]	indirect	local	click & drag deformation	IBL shadows
envyLight	[172]	indirect	global	painting; parameters	IBL

Table 7.2: Overview of different techniques, continued from Tab. 7.1.

		Interaction			Manipulation
Class/Method		Paradigm	Scope	User Interface	
Material					
Real-time BRDF editing	[11]	direct	local	parameters; curve editing	BRDF
BRDF-Shop	[27]	goal-based	local	painting	BRDF
Interactive BTF Editing	[107]	direct	local	drill-down; curve editing	SVBRDF
AppWand	[174]	goal-based	global	constraint painting	TSVBRDF
Polynomial BRDF	[10]	direct	global	parameters	BRDF
AppGen	[39]	goal-based	global	component sketching	SVBRDF
Bi-scale Material Design	[244]	direct	global	parameters; visualization	BRDF
SubEdit	[213]	direct	global	masking; selection	SVBSSRDF
AppWarp	[4]	goal-based	global	template sketching	B(SS)RDF
Interactive Albedo Editing	[74]	direct	global	painting	heterogeneous media
Appearance (Lighting & Material)					
Image-Based Material Editing	[115]	direct	local	specify matte & material	image (photograph)
AppProp	[3]	goal-based	global	painting	image (photograph)
iCheat	[163]	indirect	global	painting; labeling	surface light transport
Path-Space Manipulation (ours)		direct	global	filters; manipulators	surface light transport
Reflection Editing	[193]	indirect	local	click & drag	reflected light
On-Surface Signal Deformation	[194]	direct	local	constraints; click & drag	on-surface signal
Artist-Friendly Hair Shading	[196]	direct	local	falloff curves; parameters	hair scattering
Artistic Beams	[160]	direct	local	spatial curves; shaders	heterogeneous media
Volume Stylizer	[117]	indirect	global	painting	heterogeneous media

Chapter 8

Light Transport Visualization

A lot of relevant facts can be gathered from a (rendered) image alone, such as the geometry, relative placement, and material of objects present in the scene. Essentially, this is what human perception is made for. However, conveying how light travels in a scene, and thus affects surface appearance on a *global* level, requires additional techniques. Indeed, *visualization* is the science of conveying otherwise hidden facts through visual representations; it applies to light as well.

Beyond passive insights into illumination, visualization is also important in the sense that *one can only effectively manipulate what one can grasp visually*.

In this chapter,[1] we will first overview existing approaches to light transport visualization, with a focus on the work most closely related to ours. Afterwards, we will discuss data structures for the light paths created by physically based rendering methods, and finally present our novel light transport visualization technique, which is based on *direct visualization* of light paths.

While light transport can be represented mathematically in many forms, few works explicitly use *visually* informative representations of transport, especially not for non-academic end users of rendering systems. A notable exception is the work of Chajdas et al. [18], who devised intermediate visualizations of spatially-varying irradiance for the purpose of assisting artists in light probe placement. Another notable work is due to Reiner and colleagues [189], developing and validating (via user survey) several methods of visualizing spatially and angularly varying radiometric quantities. We will discuss this further in the next section.

Some work also focuses exclusively on light transport in a two-dimensional "flatland" space [12, 93]. In this domain, visualization is straightforward, as one can easily display fluence (Eq. 5.8) directly on the display. The work by Zirr and Dachsbacher [248] has adapted visualization techniques from vector field topology to 2-D slices of the light field. These visualizations, while insightful, are however not as practical in 3-D.

Formally, the light field of a three-dimensional scene at a specific point in time is represented by its radiance, which may be given in fully spectral form as $L_\lambda(x, \omega, \lambda, t)$ or more simply (for a fixed wavelength and time range) as $L(x, \omega)$ (cf. Sec. 5.1). This effectively 5- to 7-dimensional function is also known as the *plenoptic function*.[2] It is generally not given in explicit form, but estimated using Monte Carlo

[1]This chapter is based on our previous publication [200], and also contains an extended review of [189].

[2]Counting three positional dimensions, two rotation angles for direction, plus an optional wavelength

integration, evaluated by the rendering technique at exactly those domain points, i.e., rays, relevant for the final image. As we have seen in Ch. 5, these rays form *light transport paths*. Due to the numerous interactions light experiences as it is emitted, reflected, scattered, and absorbed in the scene, global light transport creates a lot of discontinuous, overlapping structure in the domain of the light field.

As also noted by Reiner et al., there is considerable difficulty in applying existing visualization approaches from other computational physics domains. As an example, fluid flow visualization [131] cannot be applied directly to light transport, in part due to the higher-dimensional nature of radiance and the more complex local transformations it undergoes. Our path-space filtering and visualization (Sec. 8.3) is instead influenced by work in information visualization for undirected graphs [76]. We extend their approach to the higher-dimensional path space representation of spatially and angularly varying radiance distributions, and introduce simplifications to help cope with the interactive feedback cycle of our artistic editing system.

Finally, we note that in traditional lighting design, users mostly rely on their experience, following rules of thumb, and (if warranted) trying out renderings with different settings, enabling/disabling or moving light sources, (if applicable) adjusting light linking, possibly separating different lighting components such as direct and indirect light, or diffuse and glossy reflections [163, 189].

In artistic light manipulation approaches (see Ch. 7), on the other hand, light is often implicitly visualized through a user interface *manipulator* for the phenomenon one is editing. *BendyLights* [114] visualizes a user-deformable lighting volume used to control direct illumination. *Interactive Reflection Editing* [193] and *On-Surface Signal Deformation* [194] both use custom UIs to specify spatial and directional constraints during editing. These visualizations are, however, intimately tied to the task at hand, and far from a general approach to light field visualization.

8.1 Selective Inspection

Noting the close relation to our system, we will review the work of Reiner and colleagues [189] in some detail here.

Their light visualization system is built around an extension to progressive photon mapping (cf. Sec. 5.11.6), gathering additional data needed to (a) analyze the light field and (b) create visualization representations. Light information is gathered at user-positioned, sphere- or disk-shaped *light probes*, gathering and storing photons in a directional representation via cube maps. At least for some of their tools, light probes are additionally low-pass filtered, to avoid flickering and fill in missing data.

As with our system, they implemented their tools in an existing commercial *dynamic content creation* (DCC) application familiar to artists, i.e., *Autodesk Maya*. They also use an interactive GPU-based renderer integrated into the DCC application.[3] Finally, they also offer the ability to selectively filter the light transport by spatial region and interaction type, albeit the latter in six simple, fixed categories: only (1) direct illumination, (2) indirect illumination, (3) diffuse, (4) glossy, (5) caustics, or (6) all types.

and time. By ignoring participating media, we could theoretically drop another dimension; on the other hand, light flow in free space is interesting *even in a vacuum*.

[3]In all fairness, with the overlap in authors and research group between [189] and [200], we in fact started off with an advanced version of said renderer, which was then further developed to accommodate, among other things, *light transport manipulation*.

User Feedback and Study Overall, they present five different non-trivial light inspection tools; their tools were evaluated in a formal user study,[4] the findings of which are discussed in their article. The study group consisted of 20 novel users (with background in CG), while the domain experts they separately consulted for feedback were experienced CG artists, and in lesser number, architects. After a short training session, the user study required participants to perform six different lighting design tasks: (1) finding sources of uncomfortable glare, (2) placing a reading desk in a room, (3) analyzing the composition of light on a character's face, (4) finding the source of a strong reflected caustic, (5A) finding the source of a particular anisotropic reflection, and (5B) adjusting material anisotropy to achieve homogeneous illumination. The tools they investigated for these tasks are as follows:

Plain Rendering As the name suggests, plain rendering only shows an interactive, progressive preview of the image being synthesized. This was used as a baseline to evaluate the actual five techniques against.

Their results suggest that plain rendering is already good to get familiar with the basic lighting setup of a scene, as well as perform simple, direct edits.[5]

False-Color Rendering The false-color rendering tools they showcase are similar to what engineering applications provide, namely a mapping of some quantity—e.g., surface irradiance—to color. As color maps have a scalar as an input [208], they first compute the luminance (or average) of the irradiance given as RGB color, and subsequently map it to either a rainbow scheme, pairs of colors interpolated in a perceptively linear space, or hue maps. Users can select which color map to use.

Intuitively, the study results confirm that false-color rendering is only useful to assess the brightness of the light field (on surfaces).

Spherical Plots In CG, spherical plots are often used for visualizing BRDFs, which have only angular dependence. Here, they use light probes as input for spherical plots of the incident intensity. The visual representation is a colored surface implicitly given by its radius in spherical coordinates (e.g. $r = f(\theta, \phi)$). They map the magnitude of radiant intensity to radius, and the chrominance of the input signal to surface color of the plot.

According to their findings, spherical plots are good for directional inspection of light, but fail to give any big-picture insights.

Clustering with Arrows Called "light path inspection" in the paper, this tool gathers a short history of past path vertices, and hierarchically clusters path segments into a tree of relevant light interactions. The tool can be placed both on surfaces and in free space, using disk- and sphere-shaped light probes, respectively. They reduce the data gathered at light probes by (1) clustering directions according to last interaction type and (2) using a cut-off threshold for low-contribution paths. Arrow glyphs are used to visualize clusters, snapping start- and endpoints to nearby photons, and coloring by chrominance of the clusters' average radiance signal, mapping brightness/cluster size to arrow thickness.

[4]Following the methodology we outlined in Sec. 7.6.

[5]In our current context, an *edit* means to change the parameter settings or position of a light source, *not* to artistically edit light paths, as in Ch. 9.

The study results suggest that this tool is effective, but fails to deliver global insight, unless placed at the right location. They note that their heuristic for clustering could be improved semantically, e.g., merging a lamp and its shade into one cluster.

Direct Volume Rendering of Fluence Called "volumetric inspection" in their article, this tool splats light paths overlapping a box-shaped region into a regular grid of 32^3–256^3 voxels, effectively computing (an approximation of) fluence at grid points, utilizing the rasterization pipeline of GPUs for performance (cf. [12]).[6] These are then mapped to densities and ray-marched using OpenGL shaders, a classic form of direct volume visualization.

This tool is mainly useful to analyze highly directional light flow in free space, e.g. shadows and caustics.

Particle Flow Inspired by vector field visualization, their particle flow tool randomly deposits small particles in a velocity field (as is also done real-life wind tunnels) and simulates their flow through the scene. However, the velocity field is "improvised" on the fly from the dominant direction of incident or outgoing radiance, which is a necessary dimension reduction step from light field to vector field. The particles thus follow curved paths, not the piece-wise linear segments of the actual light paths.

They note that the particle flow tool is of limited help with smooth, diffuse illumination (there is no dominant direction of light flow) and it needs to be combined with filtering by interaction type to be significantly useful.

Conclusions The main conclusion they draw is that, even though participants had a slight preference for light path clustering with arrow glyphs, one needs different tools for different tasks in lighting design and that interactivity is crucial.[7] Anticipating our later work, they also suggest to extend their system to *light transport manipulation*.

The major difference we see between their light transport visualization and our work is that all their tools perform a mapping from light paths (as generated by the renderer), to a light field (collected at light probes), to an abstract representation (displayed to the user). In contrast to this, we opt to directly visualize the inherently geometrical light path structure, i.e., the data extracted from the interactive preview renderer.

8.2 Light Path Data Structure

Before detailing our novel light transport visualization technique in Sec. 8.3, we will briefly discuss data structures for storing light paths. Virtually all physically based rendering methods stochastically sample light paths, but many of them do so implicitly. While in theory a large path data set is computed, most renderers do not retain it, discarding paths after computing their contribution was added to the image. As our goal is to directly analyze and visualize this data for the user's benefit, we need to separately maintain light paths in an appropriate structure. The following discussion of PBR methods will roughly mirror Sec. 5.11, but from the perspective of what path data they generate.

[6] As such, they may need to account for directional bias along axes.
[7] Which follows the findings of other researchers, see Sec. 7.6.

While the measurement equation only considers full paths, i.e., chains of vertices starting at emitters and ending at sensors, in reality the way paths are created and represented in physically based rendering methods is different. We believe that even by just gathering and analyzing this path data alone, one can already gain educational insights into the structure of path space, and how different rendering methods explore it by sampling.

Many methods use correlated samples of path vertices and connect them in some fashion. Connection edges between vertices can be formed in three ways [55, 126]:

- Local sampling: sampling a direction at a vertex and tracing a ray to the next scattering event,
- Global connection: connecting two independently sampled vertices and performing a visibility test, and
- Vertex merging: collapsing two vertices which are close to each other.

The latter is only part of biased methods such as photon mapping [95] or VCM [55].

In the following, we will discuss the various rendering methods of Sec. 5.11 regarding the path structure they create, for later use in direct light path visualization. The illustrations use solid lines to represent path segments created through local sampling, and dashed lines for global connections, as appropriate.

Naive Path Tracing. Apart from clearly impractical methods like independently sampling path vertices and connecting them,[8] naive path tracing is the only practical method to independently create full light paths, i.e., to sample them without correlation, starting at the camera and ending on light sources (see Fig. 8.1). Using this to our advantage for efficient path data storage—representing paths as arrays of vertices—is problematic, however, as unless our scene description explicitly forbids emission and scattering happening at the same point, the recursive evaluation of incident radiance will continue after finding a light source until (1) rays leave the scene or (2) Russian Roulette cuts off the path. Hence, whenever we create a new full path, it may share common prefixes with previously created ones, and thus many vertices will be redundant.

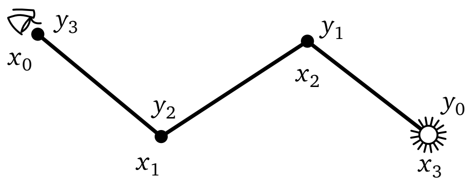

Figure 8.1: Naive path tracing methods create full paths $x_0 \dots x_k$ (length $k = 3$ in the example) starting from the camera/eye and ending at emissive objects by chance—or vice versa for light tracing ($y_0 \dots y_k$).

If we need to store a collection of full paths, as for the visualization and analysis techniques detailed later, a flexible linked-list representation can handle arbitrary path lengths well. In this scheme, we store path vertex data and an additional back

[8]The visibility function $V(x, y)$ would be zero for some connection almost always.

pointer to the vertex that it was sampled from. A full path is then simply a pointer into such a linked structure. We note that when collecting additional vertex attributes, and not just positions or normals, it may be beneficial to store them in another linked structure to avoid redundancy. As an example, when using our extended path classification scheme (Sec. 9.4.1), interaction types are not unique for each vertex, as the same vertex may have emitted light, but also have been used in sampling the next path vertex from a, say, diffuse BRDF.

Key to making the linked structure efficient is (1) storing all vertices in a single array, (2) using indices rather than pointers (which allows migrating path data in heterogeneous compute platforms, such as transferring paths from GPU to CPU, or when sending paths over the network in a render farm scenario), and (3) ensuring that every parallel path-generating processor thread maintains its own data structure, to both avoid over-locking as well as keeping stored paths compact in memory to support later cache-efficient traversal despite their linked structure.

In applications where the path length is capped, using a fixed-length array for each path may be more beneficial.

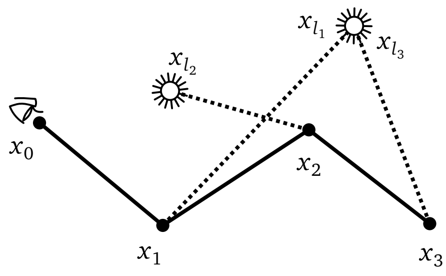

Figure 8.2: Next event estimation creates multiple correlated paths, each starting at the camera and ending at a light source. In the example, the renderer has sampled full paths: $x_0x_1x_{l_1}$, $x_0x_1x_2x_{l_2}$, $x_0x_1x_2x_3x_{l_3}$. The union of all paths forms a tree or, in the presence of point lights, a directed acyclic graph.

Path Tracing with Next Event Estimation. Next event estimation samples light sources at every vertex along a main path. This will create a tree of vertices rooted at the camera vertex, with leafs on light sources, as can be seen in Fig. 8.2.[9] This scheme also allows for multiple light source samples per vertex, which is not uncommon to be a user-defined parameter in renderers (cf. Fig. 8.3).

Again, the linked structure outlined for naive path tracing naturally handles this higher branching factor as well. We can easily reconstruct full paths by following their link into the tree and then working backwards to the root. As the number of paths can easily reach billions in realistic scenarios,[10] efficiency is important and our comments above on memory layout and thread access patterns apply as well.

Not illustrated here, the same data structure also adapts to distribution ray tracing (cf. Sec. 5.11.2), which creates a very dense tree, or light tracing (cf. Sec. 5.11.3), which is, approximately, "just path tracing in reverse."

[9]The second vertex after the camera may also accidentally hit a light source, which has to be accounted for to incorporate paths of length $k = 1$.

[10]Even at moderate settings, e.g., 1920×1080 pixels $\times 500$ samples per pixel $\times 7$ full paths per sample, on average.

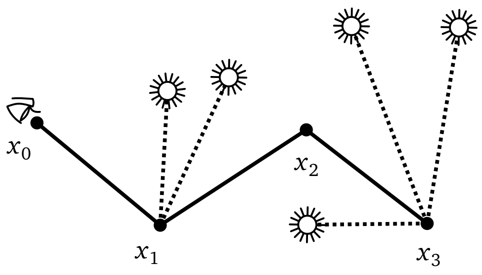

Figure 8.3: Varying the number of light samples at each path vertex alters the branching factor of the corresponding tree.

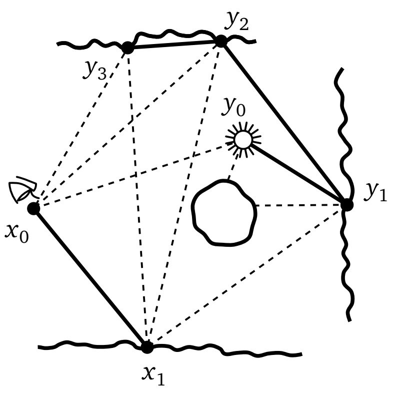

Figure 8.4: Bidirectional path sampling creates two core paths ($x_0 x_1 \ldots$ and $y_0 y_1 \ldots$) and a set of connections edges ($x_i y_j$) between vertices in each path.

Bidirectional Methods. Bidirectional path tracing (BDPT, cf. Sec. 5.11.4) samples chains of vertices from both ends, connecting vertices from either chain. Such bidirectional methods can be represented by two unidirectional path data structures, plus a list of cross connection edges, e.g pairs of pointers to vertices (see Fig. 8.4, dashed lines).

VPL-based methods (Sec. 5.11.5) and photon mapping (Sec. 5.11.6) can be handled analogously, though the possible path lengths and connections are are defined by the specific version of the method (and oftentimes a rather limited subset of those given by BDPT). Furthermore, for the basic version of photon mapping and instant radiosity, the connections between eye and light paths are implicitly given by the method itself. In photon mapping, we could rebuild edges on the fly by performing density estimation on (light) path vertices, using e.g., k-D trees [48, 177], thus not even having to store the cross-connections, as they are implicit (see Fig. 8.5). With instant radiosity, every terminal eye path vertex is connected to *every* light path vertex, i.e., VPL. If we do not mind false positives (edges without mutual visibility of vertices) for visualization, we do not have to store any connection edges; if we do, the number of edges missing (due to the visibility function between VPLs and shading points being zero) may be far lower than the number of visible edges, so we might store just these, and later query edges for their non-existence.

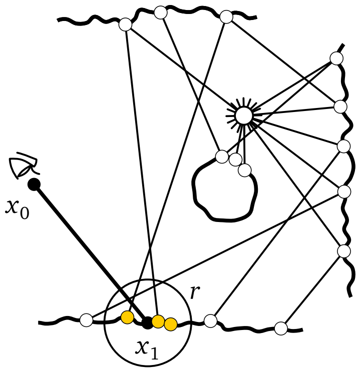

Figure 8.5: Photon mapping creates “merging” type vertex connections, which are implicitly given by the density estimation radius r (also known as kernel bandwidth) around x_1.

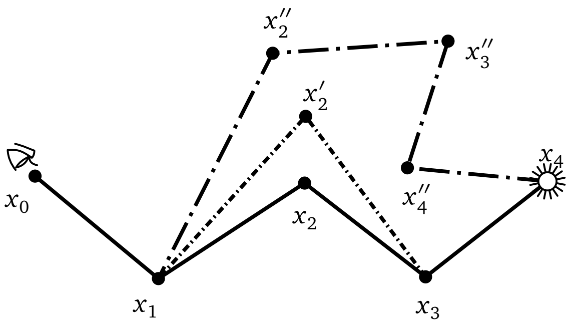

Figure 8.6: Metropolis light transport works with full paths, mutating and accepting them according to the Metropolis-Hastings algorithm. Simple mutations of a single vertex (ex. $x_2 \rightarrow x_2'$) suggest a lot of memory reuse potential for path storage. However, mutations that involve rebuilding entire subpaths (ex. $x_2 \rightarrow x_2''$), imply a more complicated path data structure.

Metropolis Light Transport and Related Techniques. At a basic level, handling Metropolis light transport (MLT, cf. Sec. 5.11.7) is quite straightforward in terms of path data structure, as the Markov chain state space already represents paths explicitly (see, e.g., Fig. 8.6), though not necessarily in a way that is optimal for visualization. We note that depending on the mutation strategies, (1) there are a lot of duplicate, i.e., correlated path vertices, (2) mutations may change path length and topology, and (3) variants such as primary-sample space MLT [110] do not explicitly represent paths in memory. We have not investigated this in detail; as noted in Sec. 5.11.7, MLT is not as relevant for production rendering due to its uneven convergence behavior.

Virtual Sphere Lights, Beam Lights, etc. Methods such as virtual spherical lights (VSLs) [71], virtual ray lights (VRLs) [159], virtual beam lights (VBLs) [159], and their later unification [125] provide connections that are inherently not line-shaped, but rather the volume swept by lines connecting two primitives. We have not investigated how to effectively visualize light transport with these methods, but as one goal of our visualization is to *make the behavior of the underlying renderer visible*, they pose an interesting avenue for future work.

8.3 Bundled Direct Visualization

In this section, we will describe a novel light transport visualization technique, which was inspired by 2-D graph visualization, and forms a core part of our artistic editing system described in Ch. 9. Our work is influenced by the approach by Reiner et al. [189], however we extend their inspection concept to a more complex path space analysis, directly utilizing the data already generated by the renderer, i.e. *light paths*, as a source. Light paths are visualized using a novel *path bundling* technique to provide immediate feedback to the artist throughout the entire light editing session.

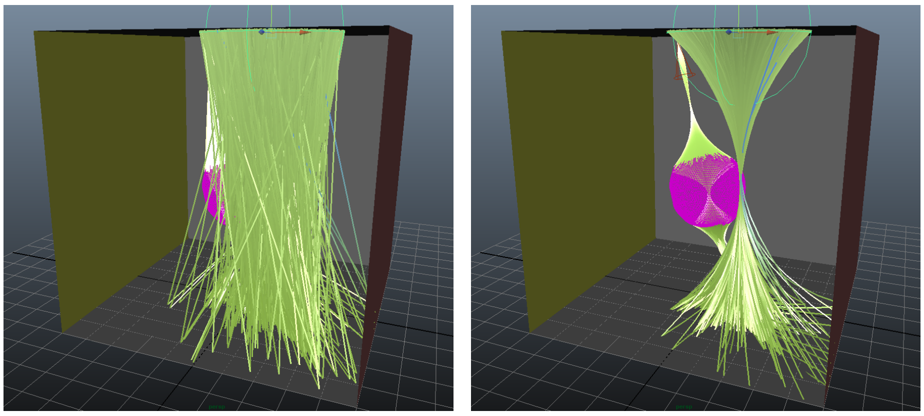

Figure 8.7: Path bundling in the *Bumpy Sphere* scene (final renderings shown in Fig. 9.11). Left: Directly visualizing light paths as straight-line segments leads to visual clutter, even when selecting only a small subset of paths, e.g., those intersecting a user-specified region (seen as the spherical widget on top). Right: Clustering paths segments and bundling them together towards their central axis removes this clutter and allows to reason about how the light (transport algorithm) is behaving in the scene. Image taken from [200].

Naively rendering light transport paths, i.e., with line segments, results in visual clutter (see Fig. 8.7, left). We were motivated by work in the information visualization community on *force-directed edge bundling* [76]. This approach performs (expensive) physical simulations to improve the visualization of dense two-dimensional graphs by drawing contracted edges. We present a variant more suitable to our problem which, in contrast, is efficient to compute and readily generalizes to 3-D light paths. We additionally use line shading to help visualize the spatial structure of bundles, which is not necessary with two dimensions.

As the renderer samples light paths, we analyze *path segments* as follows:[11] We first classify segment end points, i.e., path vertices, by interaction type (e.g., diffuse, glossy, specular) and object ID, and use this information to assign the segment to a specific "bundle," representing a specific type of light energy being transported

[11]This cluster analysis for visualization is different from the analysis for the light editing user interface. We will discuss the latter in Sec. 9.4.2.

from one object to another. The classification scheme could be easily extended to include additional attributes such as surface normals or material IDs, but we found the combination of interaction type and object IDs to be sufficiently robust, not the least because object IDs (i.e., names) usually have a salient semantic interpretation for authors and users of a virtual scene.

Once we have assigned all light path segments to their respective bundles, for each bundle we find the average of start and end points of included segments. As our classification scheme is discrete, this immediately gives us the bundles' medial axis (cluster center). One could iterate cluster assignment in the spirit of k-means clustering [53, 134], but since our classification criteria are stable *by construction*, cluster assignment is also stable after the first iteration. Our cluster assignment also avoids the costly search for neighboring segments, as is the case with the work by Holten et al. [76].

We then render each path segment as a curve using its own attributes and the cluster axis; the entirety of all segments visually bundle together similar light transport paths, which effectively reduces clutter (cf. Fig. 8.7, right). In the following, we detail how curves are formed.

Starting from the end points of a path segment x_0x_1 and the bundle's medial axis defined by c_0c_1 (the cluster center), we first find target points x'_0 and x'_1 by orthogonal projection onto the bundle axis. To form our parametric bundle curve, $p(u)$, we first interpolate linearly in the u direction to find $x = \text{lerp}(x_0, x_1, u)$ and $x' = \text{lerp}(x'_0, x'_1, u)$.[12] We then evaluate our influence value:

$$t(u) = a \cdot \left(\frac{1}{2} + \frac{1}{2} \cos(2\pi u - \pi) \right), \tag{8.1}$$

where a is a user-specified *bundling amount* that allows to smoothly adapt how much to pull the path segment from its original position towards the bundle center. The influence value (Eq. 8.1) is used in another linear interpolation to find the final curve point (see Fig. 8.8 for an illustration.):

$$p = \text{lerp}(x, x', t(u)). \tag{8.2}$$

We provide users an option to flatten the influence value curve for internal points, such that bundles only fan out at the beginning and end points of entire light paths, which is helpful for longer specular chains. (This is implemented by adjusting Eq. 8.1 accordingly.) Extensions of the general idea are possible, and we experimented with different approaches to generate curve geometry, including other projection points, influence curves, as well as approaches based on splines of quadratic and cubic Bézier curves, but found that our model gave us the most freedom in shaping the bundles.

Curves are shaded using Banks' model [6, 136], for which we evaluate tangent directions numerically. Shading optionally uses colors coded according to the interacting objects, or the chrominance of the path throughput (mapping bundle colors to diffuse reflectance k_d in Bank's simple shading model). This visually links bundle colors to colors in the image, which are caused by corresponding paths. An isolated example of the visualization was shown in Fig. 8.7; Fig. 9.2 shows the visualization in context with the user interface of our editing system.

[12]Either of these may lie inside or outside of the segment c_0c_1. We use the shorthand notation $\text{lerp}(a, b, u) = (1 - u) \cdot a + u \cdot b$.

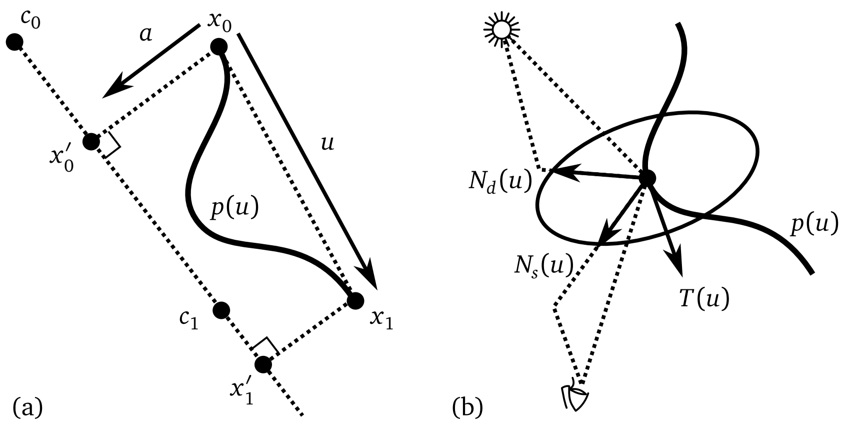

Figure 8.8: Light path bundling. (a) For every path segment x_0x_1, we render a parametric curve $p(u)$ based on the medial axis c_0c_1, which is formed by clustering similar light path segments. Path segments are smoothly pulled towards the medial axis by a user-controllable amount a. The entirety of curves so form "bundles," visualizing a major flow of light between two surfaces (cf. Fig. 8.7). (b) Shading is based on Bank's model for illuminated curves [6, 136], picking two different normal directions orthogonal to the curve's tangent direction $T(u)$, so as to maximize the observed specular and diffuse illumination.

In evaluating our overall system, curve rendering time was not a bottleneck, despite being implemented on the CPU and utilizing OpenGL only for rasterizing final curves; even so, curve evaluation could be ported to GPU shaders easily, potentially using hardware tessellation, allowing to render massive numbers of bundles.

The number of light paths to be stored for visualization is user-definable. In case the rendering back-end produces more paths than can be stored or visualized reasonably, we use reservoir sampling [233] to retain an unbiased subset of all paths. This is especially important in the context of progressive rendering, which produces a potentially infinite number of paths.

It remains to be (formally) tested how our technique would fare as a stand-alone visualization technique,[13] but our approach of making the actual data generated by the renderer directly visible, as opposed to a more abstract representation of the light field as in [189], proved indispensable in the overall system, which will be described in Ch. 9.

We also note that applying subpixel rendering to our bundle visualization could be beneficial, however in our experience, for a reasonable number of light paths, bundles quickly form thicker, volumetric conglomerates, where subpixel filtering would only enhance the display of their edges.[14]

[13]We did experience situations were uninitiated observers thought the visualization actually represented curved, i.e. *manipulated* light paths.

[14]Also, in practice we are limited by the content creation or lighting tool we integrate into, where even rendering non-subpixel-aware anti-aliased lines may be considered a luxury.

Chapter 9

Artistic Path Space Editing

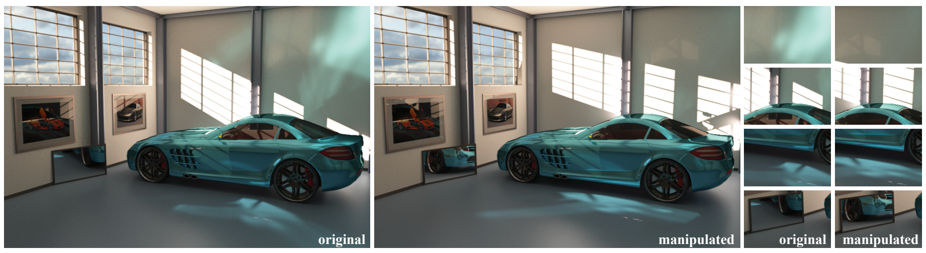

Figure 9.1: Transport manipulation in the *Garage* scene. Before/after close-ups (right): removing indirect highlights caused by the car, re-directing sunlight after it refracts through the windows, moving and rotating a glossy inter-reflection, and altering the mirror reflection. Image taken from [200].

In this chapter,[1] we present a novel light transport manipulation technique that operates directly on path-space solutions of the light transport equation, which forms the basis of all state-of-the-art physically based rendering (PBR) methods (Ch. 5).

The creation of high-quality images for applications such as the entertainment industry (e.g. movies and video games) requires tools for lighting artists to quickly prototype, iterate, and refine final renders. As of this writing, most industry-leading studios have adopted PBR across their art generation pipelines, and many of the existing tools have become unsuitable for their purpose, since they address only simple effects, while neglecting the underlying PBR concepts and constraints.

In our work, we expose to artists intuitive direct and indirect manipulation approaches to edit complex lighting effects such as (multiply-refracted) caustics, diffuse and glossy indirect bounces, and direct or indirect shadows, which are difficult to alter effectively in traditional PBR workflows. Our sketch- and object-space selection is built on top of a parameterized regular expression matching engine with which artists can effectively search and isolate shading effects to inspect and edit. We classify and filter light paths on the fly, as they are generated by a (progressive) rendering method, and visualize the selected transport phenomena. We present two light path manipulation approaches we named *path retargeting* and *path–proxy linking*, which are examples of our more general framework of editing lighting effects

[1]This chapter is based on our previous publication [200].

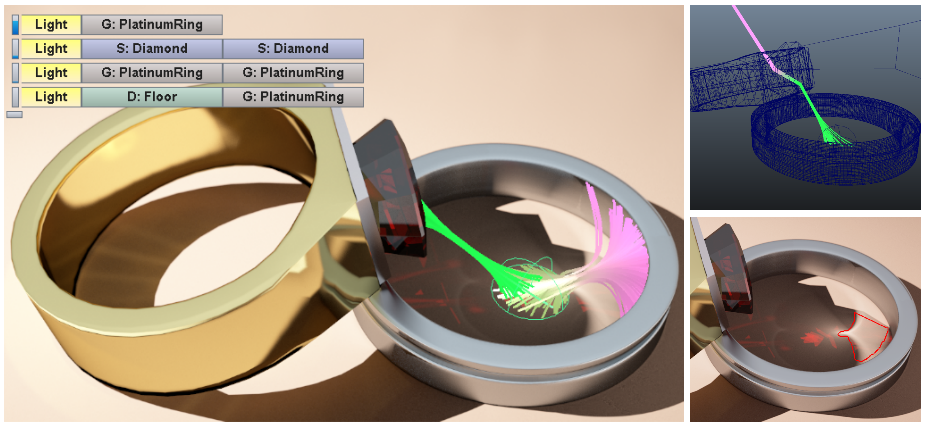

Figure 9.2: Left: our interactive GI preview suggests a list of path classifications (using our extended notation) ranked by the contribution to the user-specified region; paths matching the top two classifications are visualized as bundles. Right: a caustic multiply refracted by the gemstone; sketch-based selection of the ring's caustic. Image taken from [200].

in path space. Finally, we surveyed artists who used our tool to manipulate complex phenomena in both static and animated scenes.

9.1 Introduction

Increasing artists' productivity by minimizing iteration count and time, in order to meet creative goals, depends more on the flexibility of their lighting tools than on the underlying PBR system. As we have seen in Ch. 7, many existing tools either target non-PBR systems or consider only very specific PBR effects.

We present an artistic lighting tool, built on physically based global illumination solutions, that enables rapid and intuitive selection and manipulation of light transport, including effects that result from complex light paths (see Fig. 9.1). We integrate atop digital content creation (DCC) tools prevalent in the industry and support various PBR algorithms.

Our approach interactively clusters light paths according to user selected feature(s) and provides a visualization of clustered paths using the technique discussed in Sec. 8.3. This instant feedback links an artist's selection to its underlying PBR constructs (see Fig. 9.2).

To cope with the complexity of general light transport, we complement selection with subpath ranking and filtering based on path type as well as path–object interactions (Sec. 9.4). We also extend existing 2-D and 3-D interaction approaches with additional editing features, such as the ability to sort and filter transport subpaths based on type and/or on the objects they interact with.

In contrast to previous work, our manipulation operates entirely on path space, rather than targeting specific shading phenomena. We present two complementary editing concepts, each of which permits light transport editing from a different perspective (Fig. 9.3):

- *Path Retargeting* allows the user to *directly* select and transform grouped paths.

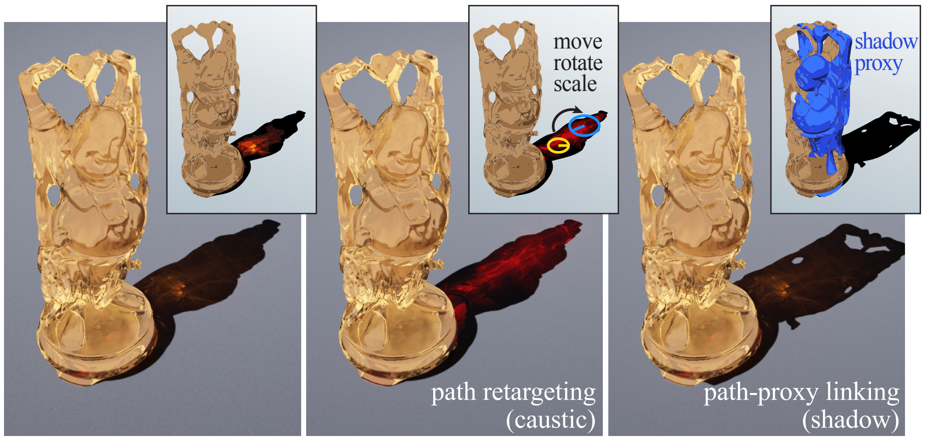

Figure 9.3: Left to right: original render; using path retargeting to displace light paths forming the caustic; path–proxy linking creates a proxy instance of the Buddha that affects only the shadow. Image taken from [200].

- *Path–Proxy Linking* edits path space *indirectly* according to edits of the scene.

These two approaches enable editing of complex shading effects such as multiply-refracted caustics, indirect lighting, reflection, and shadows. They are purposefully redundant, as some editing tasks can be completed with both path retargeting or path–proxy linking, although, depending on the task, one approach may be more intuitive or efficient.

Considering the entire path space poses several technical challenges, as we make no prior assumptions on the form of transport. We discuss how to render manipulated path space with bidirectional, importance-sampled global illumination (GI) algorithms.

We make the following *selection*-related contributions:

- path selection on surfaces with stroke- and region-sketching,
- automatic clustering, classification and ranking of transport effects based on light paths and their neighborhoods.

We make the following *manipulation*-related contributions:

- two complementary editing concepts for artistic light transport manipulation: *path retargeting* and *path–proxy linking*,
- consistent rendering of edited transport with PBR algorithms,
- a more general path-space editing solution, in which many existing techniques become special cases of our approach.

9.2 Previous Work

Having already extensively discussed previous work on appearance design and, specifically, light editing (Sec. 7.4), we will briefly contrast it with our work here.

Prior art uses sketch-, click-and-drag, and paint-based editing concepts; our system couples sketch-based selection with path-space filtering in order to facilitate identification of light features. As direct and indirect interaction paradigms are better suited to typical editing operations (cf. Sec. 7.3), we devise direct and indirect approaches to manipulate light transport, while goal-based painting interfaces are only used for selection.

Lee et al. [132] edit shading independently per object. Similarly, our selection and filtering of path space according to scene object IDs allows users to constrain edits to specific objects, but also to isolate effects *between* a subset of objects.

Similar to previous work, we also use a customized UI to intuitively expose transport manipulation operations for directional and spatial selection/editing. We do so while respecting the UI and interaction concepts of the DCC application we chose to build our system atop. This shrinks the learning curve and builds on artists' training and skills.

As summarized in Sec. 8.1, Reiner and colleagues [189] provide selective inspection and visualization tools for light transport; we extend these inspection approaches to a more complex path space analysis, as well as expose several approaches to now *manipulate* the visualized quantities.

Interactive *On-Surface Signal Deformation* (OSSD) [194] requires surface parameterizations (computed on the fly), complicating its applicability to animated/deforming scenes and restricting edits to signals *on surfaces*. In contrast, our approach manipulates path (space) segments, naturally supporting complex transport effects and animations without the need for any surface parameterizations.

All prior work either supports limited types of transport effects or relies on specialized rendering engines or precomputation, impacting their adoption in large-scale production pipelines.

Since the original dissemination of our method [200], to our knowledge, two other publications have built on a very similar approach. Ray portals [218] allow a larger class of edits than path retargeting, however have to introduce a search for valid configurations when connecting light path vertices in (bidirectional) path tracing. The search is linear in the number of connections *and* edits, and it is not discussed whether it terminates in the presence of multiple portals. The method by Günther and colleagues [63] performs smooth interpolation of caustic photons, thus allowing to directly specify target light distributions in a goal-based fashion, without having to manually retarget photon paths as in our work. These techniques nicely complement our more direct interaction approach by adding new tools to our framework, while methods such as *BendyLights* [114] or *Interactive Reflection Editing* [193] can be more seen as precursors subsumed by our method.

9.3 Goals and Overview

We will identify the goals of our system and outline how our method integrates into standard modeling and lighting workflows. We target six design requirements, while simultaneously balancing between artistic control and plausibility:

- **WYSIWYG Editing:** an interactive preview of the progressive GI solution (i.e., including caustics and indirect illumination) is needed to better inform artists at edit time.

- **Consistency:** edits must respect the PBR behavior by default, e.g., moving a shadow must also affect indirect lighting (unless explicitly disabled). Layered edits must also be supported.
- **Animation:** key-framing of editing operations should be supported, with edits extending naturally to animated scenes.
- **Usability:** edits should be easy to learn and intuitive, building on widely adopted user interfaces and workflows.
- **Generality:** all light transport manipulations should operate in a unified manner, e.g. caustic and diffuse indirect light selection and editing should be exposed under the same user interface.
- **Rendering:** the edited light transport solution should be computable using robust (ray tracing-based) GI methods. This is made possible by (implicitly) modifying path space, which is the underlying concept of those methods.

We implemented our method as a plug-in for a professional DCC system and employ stochastic progressive photon mapping (SPPM) [64] as our rendering back-end during editing. SPPM balances quality with immediate interactive and progressive feedback and, as with any ray tracing-based back-end, allows us to trivially gather path sample information. We also overlay a visualization of light transport paths (Sec. 8.3 and Fig. 9.2) to facilitate the inspection and editing of selected phenomena.

A typical editing session starts with selection and filtering of a transport effect. The user defines a region of interest by sketching, placing a bounding volume around the target area, or selecting objects that interact with the transport. Our system then automatically ranks incident light paths within the selection by their contribution and the surrounding transport, to assist in finding relevant light transport features quickly. At any point, the user may override our tool's suggested rankings and either manually select from an automatically generated list of transport paths or specify paths using our parameterized regular expression notation (Sec. 9.4.1).

We present two complementary editing concepts, each of which manipulates transport according to a different rationale (Fig. 9.3): *path retargeting* operates on the space of transport paths, allowing the user to transform (e.g. translate, rotate, or scale) path vertices; *path–proxy linking* indirectly edits light transport for a particular transport phenomenon according to external scene adjustments, e.g. artists can "displace" a shadow-casting object and our tool automatically creates an "invisible" proxy that only affects its shadow(s).

9.4 Filtering and Selection of Light Transport

Targeted and deliberate transport editing requires a precise and intuitive way to select and filter light transport effects in the scene. Internally, we employ an extension of Heckbert's regular expression-based path notation [75] that includes object IDs, providing more detailed differentiation of individual path interactions (Sec. 9.4.1).

As of this writing, similar classification schemes are supported in many commercial renderers, under the term "light path expression" (LPE) [22, 161, 178, 212, 214]. LPEs, however, are in most cases used to render a handful of different lighting features into separate image layers, for later use in a (non-physical) compositing stage, and as such require careful pre-selection of LPEs for the lighting effects of interest.

Moreover, beyond image location, all spatial information about light transport is lost. This is contrasted by our artist-friendly *interactive* exploration of light transport structure in the three-dimensional space defined by the given scene.

Expert users can select transport according to our notation, but we also expose more intuitive semi-automatic selection techniques (Sec. 9.4.2). In either case, our system analyzes and ranks the most significant phenomena in a selected region, all while visualizing the selected light transport according to our informative visualization scheme, which we covered in Sec. 8.3.

Fig. 9.2 illustrates our selection tools (and light path visualization). We use standard DCC gizmos (widgets) and overlay additional UI elements atop modeling and progressive PBR preview viewports. All options and parameter input fields are integrated into the DCC's standard menus and dialogs.

9.4.1 Extended Path Classification

Our parameterized regular expression syntax for path classification distinguishes between diffuse (D), glossy (G), and specular (S) interactions. We additionally classify interactions as reflections (superscript $\square^R$) or transmissions ($\square^T$), and store the ID of the object at the interaction. Extensions to multi-lobe BRDFs and object groups are conceivable, but largely depend on the underlying renderer and modeling system.

An important addition compared to other variants of Heckbert's notation or LPEs is that we define a token X_P corresponding to an *arbitrary* surface interaction (D, G, or S) that (optionally) lies in a user-selected sub-region of $P \subseteq \mathbb{R}^3$.

This extended notation is powerful enough to classify all surface-based editable transport phenomena. For example, $L_i S_j^T S_j^T X_P E$ corresponds to a visible transmissive caustic through object $\#j$, lit by light source $\#i$, where the final interaction (before the eye vertex) lies in a (user-specified) region P. To simplify user interaction, these expressions can easily be converted into a textual description for common use cases (and vice versa), such as, "caustic through <name of object>,"

9.4.2 Smart Selection

We believe an intuitive way of selecting a shading effect is to define a region on a surface where it is visible (i.e. $L(D|G|S)^* X_P E$). As such, we allow users to sketch or place bounding volumes (Fig. 9.2) to delimit regions of interest. We then collect transport paths whose vertices (i.e., photons) fall into this region, and compute a flux-weighted histogram according to the paths' classifications.

We subsequently offer two ways of ranking these paths (both exposed in the UI): *contribution ranking* orders histogram bins by decreasing flux and presents the first entries of this sorted list to the artist (the first element is chosen as default), while *discrimination ranking* compares the transport in the selected region with the transport in its surrounding neighborhood, using the rationale that something about the selected light transport is special or notable compared to its surrounding. To this end, we compute a second, outer histogram in an enlarged (50% by default) region around the original region of interest and compare it to the first histogram: we order path classifications by their (decreasing) relative flux in the histograms, i.e., for every classification, we divide its accumulated flux in the enlarged region's histogram by that in the inner one (as we use convex bounding volumes, paths in the inner region are guaranteed to be included in the outer region as well). We similarly select and manipulate eye subpaths, e.g., a mirror reflection X_PSE, by shooting importons (particles carrying importance) from the camera, which amounts to naive path tracing. Light transport effects can also be selected according to the order in which scene objects interact in the desired path, permitting a more visual approach to building the desired path specification. Here, we are limited to contribution ranking, since no region is specified.

Many photons must be collected (in the region of interest) to form meaningful statistics in complex scenes with many objects; we accumulate photons over many frames. Alternatively, an importance-driven metric can be used during accumulation. In complex scenes, histogram size can be reduced by merging similar bins according to high-level descriptors (as in Ritschel et al. [194]).

9.5 Manipulation of Light Transport

We now focus on our interactive manipulation tools. As mentioned earlier, our design goals include consistency and generality: the manipulated shading should remain as physically plausible as possible, without restricting artistic freedom. To this end, we start from a physically-based GI solution and always provide interactive feedback.

In the following, we will detail our *path retargeting* technique (explicitly altering path segments), and then describe *path–proxy linking*, which enables manipulation by linking transformed scene objects to appropriate light transport phenomena. Both of these strategies additionally support brightness scaling/offsetting and hue editing (in the spirit of [163]).

9.5.1 Path Retargeting

Path retargeting provides focused control over distinct lighting features, e.g., dragging a caustic or moving a reflection. The underlying concept is to choose a light (e.g. diffuse indirect light LD^RX_P) or an eye subpath (e.g. mirror reflection X_PS^RE), and then retarget (move) the subpath's endpoints. Retargeting alters path segments and thus implicitly affects secondary effects, such as indirect occlusion and interreflection. The following discussion assumes manipulation of a light path; eye paths are handled analogously.

Retargeting operates as follows: first, the transport phenomenon is selected (Sec. 9.4.2), defining the light subpaths in our extended path notation that will be affected during retargeting. If the user chose to specify a region of interest using a UI gizmo, then its center will act as a *source anchor*, serving as the origin of the retargeting transformation. The transformation is an affine mapping defined by placing a *target gizmo* (Fig. 9.2) in the scene. Note that the user can choose either to retarget only path vertices within the region of interest, or all vertices matching the classification filter. We illustrate the usefulness of this latter option, for example, when a glass sphere that is causing a caustic is retargeted. If the sphere is moved, the caustic may migrate outside the source region. Only when matching the path classification will the caustic also be retargeted. Both source and target regions are independent of the object causing the phenomenon, however their transformations can also be key-framed, and optionally specified relative to the object's location.

We give an example of retargeting in the context of photons: after selecting a caustic path $L_iS_j^TS_j^T(X|X_P)$, displacing the target node translates only the path vertices (photons) in the selected region P whose path matches this filter. Our

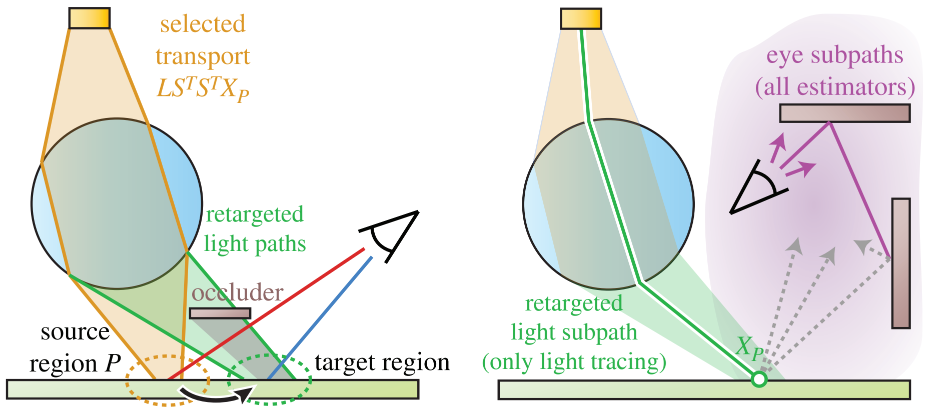

Figure 9.4: Left: Retargeted paths (shown here for light subpaths) interact with scene geometry. If we apply the inverse transformation to the blue camera ray (yielding the red ray) and continue tracing the path, we would miss the occlusion. Right: In bidirectional path tracing, we only allow manipulated path segments to be created with unidirectional sampling. Here, the light path must be constructed with light tracing up to X_P and only afterwards can we combine estimators using multiple importance sampling. Image taken from [200].

preview provides interactive feedback to the user, and any new path can of course undergo further manipulations. Note that we move path vertices but maintain the energy throughput of the original path; otherwise, for glossy and specular BSDFs, the throughput of manipulated paths would be (close to) zero. Such a manipulation can also be seen as a local tangent space transformation at the interaction before X such that the throughput (if recomputed) equals the original path throughout (see Sec. 9.9).

Integration with Light Transport Algorithms

Retargeting is defined to operate in the direction in which either light or importance propagate and can thus be trivially applied to light or eye subpaths, respectively. However, a fixed-length path can be constructed from several pairings of light and eye subpaths, each of different length. Therefore we have to ensure that all potential bidirectional Monte Carlo estimators consistently interpret the manipulations. Unfortunately, applying a manipulator in the opposite direction of its definition is non-trivial (Fig. 9.4, left). We proceed to describe our practical solution to this problem that ensures consistency and easily integrates into SPPM or bidirectional path tracing (BDPT) [128,230] at little extra computation cost. In Sec. 9.9 we discuss a more robust and theoretical approach that might perform better in some cases, but its integration into existing renderers is non-trivial.

We again provide an example for manipulated light paths, noting that eye path manipulation behaves analogously. BDPT constructs light and eye subpaths and then connects them. We can readily apply retargeting to light paths, but inconsistencies may occur when we trace the same path from the camera (i.e., when we use an estimator that attempts to construct the manipulated segment as an eye subpath; Fig. 9.4, left). Since it is not clear how to properly retarget eye paths, we must ensure that we employ only Monte Carlo estimators that construct the manipulated

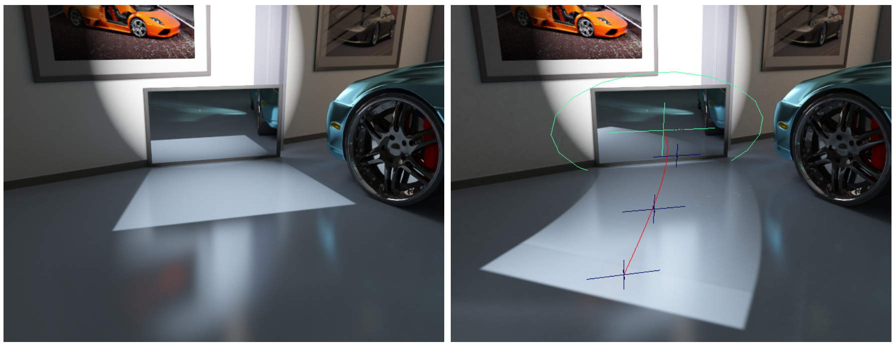

Figure 9.5: Bending paths to form "BendyCaustics" off the mirror. Image taken from [200].

segment(s) in the same direction as the manipulator's operation (Fig. 9.4, right).

For example, if we displace a caustic with a manipulator, we must ensure that only the (light-traced) unidirectional estimators up to the manipulated vertex are used. Once we reach the manipulated vertex, we have a subpath that can connect to the eye using all estimators. This ensures consistent results but reduces the set of available estimators for some subpaths, potentially locally decreasing the efficiency of multiple importance sampling in BDPT. SPPM is a special case where each path can be constructed using only a single estimator, avoiding any inconsistencies. For instance, caustics are manipulated by transforming light paths, affecting only photon tracing, and then density estimation connects to the eye paths. Analogously, reflection editing is an example of manipulating eye paths connected to light paths using density estimation.

Increasing Artistic Flexibility

Path retargeting can also be generalized: in our prototype we deform path segments to become curved trajectories (see Fig. 9.5 for an example) which generalizes *BendyLights* [114] to arbitrary transport phenomena.

On-surface signal deformation [194] "displaces" surface shading effects, however, topology constraints limit editing flexibility, e.g., caustics cannot move from an object to another disconnected one. In contrast, we work in path space where edits transparently transfer across objects (surface signals are just slices of the light field), including objects undergoing topological changes by animation/deformation/fracturing. We also automatically update secondary illumination effects, e.g., indirect shadows, to maintain PBR consistency.

9.5.2 Path–Proxy Linking

Our second manipulation concept is based on the idea that certain shading edits can be best conceived and achieved through manipulation of the scene elements, instead of explicitly editing light paths. For example, an object's shadow silhouette can be intuitively edited by rotating or scaling the object. The idea of *path–proxy linking* is to offer the possibility to specify different object transformations for individual components of the light transport, and can thus be seen as a generalized light linking

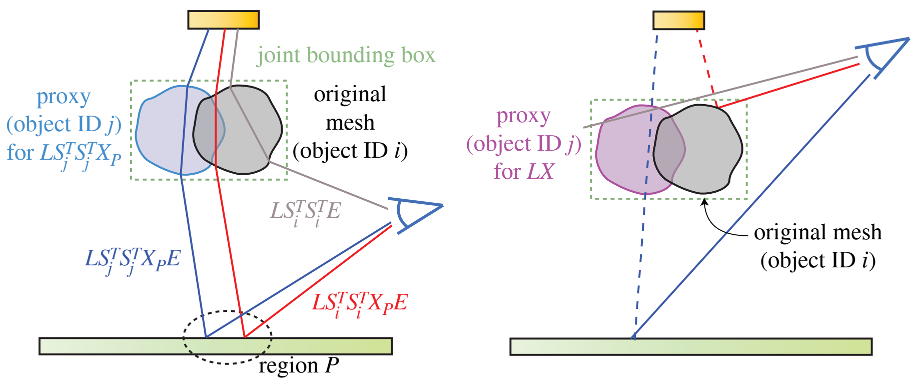
joint bounding box
proxy (object ID j) for $LS_j^T S_j^T X_P$
original mesh (object ID i)
$LS_i^T S_i^T E$
$LS_j^T S_j^T X_P E$
$LS_i^T S_i^T X_P E$
region P
proxy (object ID j) for LX
original mesh (object ID i)

Figure 9.6: Left: path–proxy linking for refractive caustics; paths are discarded if they match the filter but have not been created with the proper proxy. Right: next event estimation for eye subpaths is also discarded when the wrong instance is used; primary rays are always tested against the original mesh. Image taken from [200].

technique (but not a super-set).[2] Path–proxy linking can also be used for *subtractive* shading phenomena, such as shadows, which are best described by a lack of light transport rather than with a type of path; such edits would not be possible using path retargeting.[3] We also believe that manipulating a shadow using path–proxy linking is more intuitive: users select direct light on a surface, pick the shadow casting object, and are then presented with a proxy object used only for shadow computation (Fig. 9.3, right), that can be transformed using standard workflows.

As before, an edit always starts by selecting and filtering the desired transport phenomenon. After potentially using the visualization to help understand which scene objects are involved in the generation of the selected shading effect, the user selects one or more objects influencing the selected transport to be manipulated. Our method automatically creates a proxy object (virtual instance), for every selected object, that can undergo affine transformations while affecting only light transport of the selection filter. Every proxy is also linked to the selection gizmo, and once the selection is modified, the manipulated transport updates as well. We also permit multiple proxies per object, e.g., to separately manipulate shadows and indirect illumination. The geometry itself is not duplicated; when the object geometry is modified, all proxies are updated accordingly. Every proxy has a unique object ID, which is required to distinguish the different light paths during GI computation.

Efficient integration path–proxy linking requires two modest changes to acceleration structures and GI algorithms, which we detail below.

Acceleration Structures We use a bounding volume hierarchy (BVH) with two hierarchical levels.

First we build a BVH over the polygons of each object in the scene. These BVHs form the lower level of the hierarchy. Then we compute the *joint bounding box* of each

[2]The term *path–proxy linking* was chosen to explicitly indicate that light paths become linked to geometric proxies in the scene.

[3]Photon mapping can simulate shadows with “shadow photons” carrying negative energy, which would facilitate path retargeting of subtractive effects. We avoid this to remain compatible with more GI methods.

object and all its proxies. Finally, we construct the upper level of the hierarchy as a BVH over the joint bounding boxes, where each leaf contains the path classifications and affine transformations for the proxies of the corresponding object. This two-level hierarchy enables efficient affine transformations, as only the upper level needs to be updated when manipulating the proxies.

Global Illumination Computation At the core of every ray tracing based GI method, paths are constructed between light sources and sensor points. No special treatment is required for paths that interact only with non-manipulated objects.

For a path interacting with a manipulated object, consider the example in Fig. 9.6 (left): the user selects a refractive caustic $LS_i^T S_i^T X_P$ and creates a proxy for object i with the new ID j. All refractive caustic paths will now be computed using only the proxy, not the original object (splitting path space into disjoint sub-spaces). To achieve this, we need to slightly modify the tracing of paths. When a segment intersects a joint bounding box containing the original object and N proxies, the path should interact with only one of the $N+1$ representations. Since we do not yet know the complete path's classification, we probabilistically pick one representation and continue with construction. Once we can determine whether the path's classification matches the linked filter, we re-weight the path's contribution by $N+1$. Otherwise, we discard the path. In our example, the prefix of the red path $LS_i^T S_i^T X_P E$ matches the type of interaction, but uses the original mesh instead of the proxy and so will be discarded; the prefix of the blue path $LS_j^T S_j^T X_P E$ matches and uses the proxy for this type of light transport; thus this path contributes to the image. The gray path $LS_i^T S_i^T E$ used the original object and since it does not match the proxy's filter, it also contributes to the image.

Fig. 9.6 (right) demonstrates how path-proxy linking works with path tracing and next event estimation. Here, the user selects direct light manipulation, i.e., the filter is LX, and a proxy (ID j) has been created for this filter. When the original mesh (ID i) is chosen for the blue path, then the path is discarded: the prefix of the path LDE matches that filter LX, but does not use the proxy linked to it. When a primary ray intersects a joint bounding box, we always use the original mesh to construct paths, i.e., the red path is created as expected, while the gray ray will not intersect geometry.

To summarize, if a path uses the original mesh then the intersecting segment is a primary ray or the path remains valid only if it does not match any filter of the proxies; if a path intersects one of the proxies, it is valid only if its prefix matches the proxy's filter.

9.6 Extensions

In this section, we describe how our edits generalize to multiple layered edits, animated scenes, as well as participating media.

9.6.1 Multiple Edits and Animated Scenes

Two important requirements for artistic transport manipulation are that edits can be applied to animations, and also be animated (e.g., key-framed) themselves. The first is implicitly met since our method works on transport paths and manipulation is both affected by, and transferred to, static and dynamic objects. Moreover, all

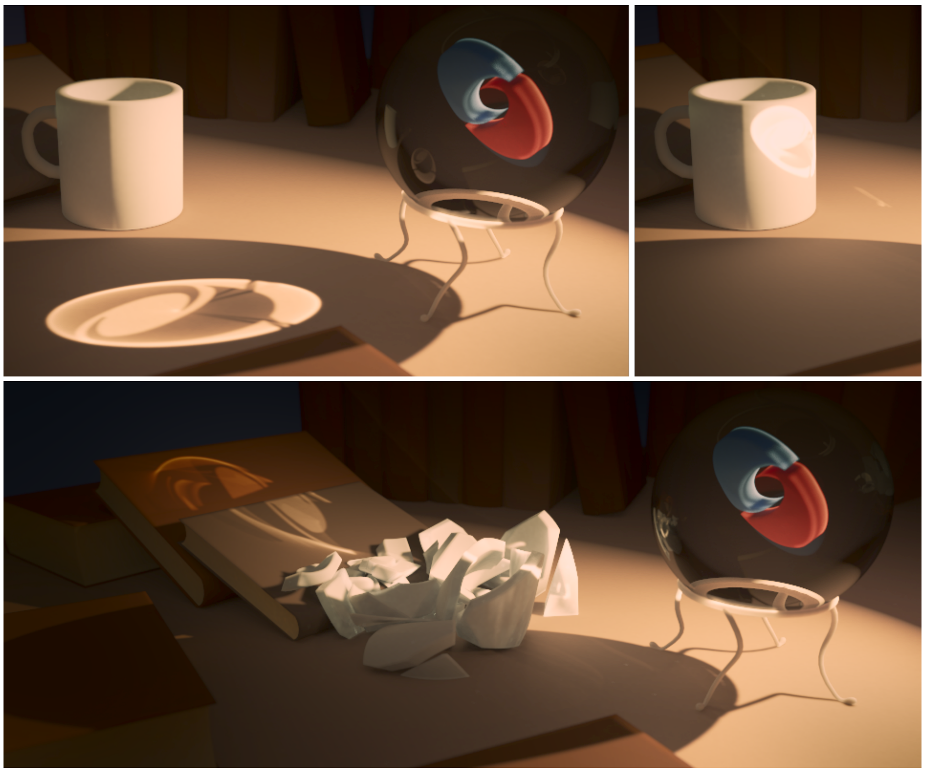

Figure 9.7: In reading order: the first frame of the *Mug* animation sequence, without any manipulation. The caustic is manipulated with path retargeting, causing indirect lighting in front of the mug. The mug has been shattered, and the manipulated caustic now appears on a stack of books in the background. Image taken from [200].

selection and manipulation gizmos can be animated in the same way as objects in the scene. All attributes can be key-framed or bound to simulations (e.g., a target gizmo can be bound to a rigid-body simulation). The video accompanying our original publication [200] illustrates transport manipulation in dynamic scenes with changing topology.[4] Multiple edits are supported in both path retargeting and path–proxy linking and can also be combined. Path retargeting manipulations do not commute, i.e., the order they are applied in matters.

9.6.2 Participating Media

The concepts we have elaborated on are straightforward to extend to light transport in participating media, see Fig. 9.8. While similar effects to the example could be added using image-based-techniques, doing so consistently for every frame of an animation sequence would be very tedious and error-prone.

We note, however, that path retargeting in dense media is inherently challenging, due to the short length of path segments. In such situations, a model based on continuous space deformation would be a better choice. Path–proxy linking is not affected by this issue.

[4]As of this writing, available at `http://cg.ivd.kit.edu/PSMPBLT.php`.

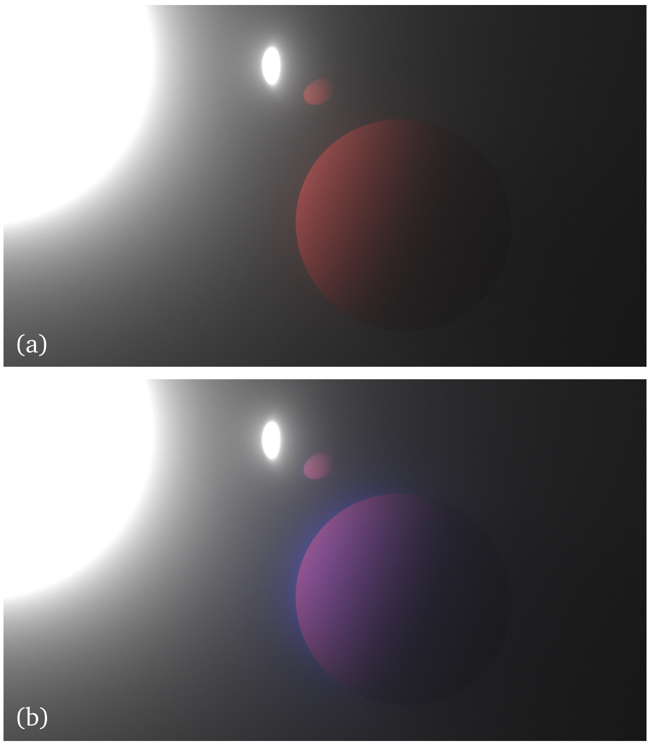

Figure 9.8: Volumetric path editing example. (a) Unmodified rendering of mirror and diffuse spheres in a homogeneous, strongly forward-scattering participating medium (Henyey-Greenstein phase function, anisotropy $g = 0.9$), illuminated by a disc light. (b) Tinting the indirect illumination of the central sphere, i.e., paths of type $LD_{\text{sphere}}V^{+}S?E$, from red to blue creates an effect reminiscent of a planet's atmosphere, while maintaining the appearance of the remaining light field. The run-time overhead for analyzing, matching and editing light paths was roughly 19%.

9.7 Results and Example Edits

We implemented our prototype editing system as an Autodesk Maya 2012 plug-in. Videos and timings were recorded on a PC with a 3.20GHz Intel Core i7 CPU, 8GB of RAM and an Nvidia GeForce GTX 580 card with 3GB of VRAM. Interactive previews progressively update at 5 to 40 frames per second, depending on the scene complexity and the viewport resolution.

Fig. 9.1 illustrates sequential edits, here on $LS^{T}G^{R}X_{P}$ and $LS^{T}X_{P}$ paths (glossy reflection off the car and sunlight through the window), a rotation and scaling of a glossy reflection on the floor, as well as reflection editing ($X_{P}S^{R}E$ paths). Reflection editing is not possible with light linking; the glossy reflection on the floor can of course be re-positioned with light linking, however, this would lack precise control over its shape and orientation, increasing artist effort (see the video for our edits). Converged renderings took several hours in our instrumented SPPM implementation (which is slower than pure SPPM due to the additional path bookkeeping).

Another scene from our video is shown in Fig. 9.7, where we retargeted a refractive caustic onto a mug that shatters. Notice how the changing indirect illumination

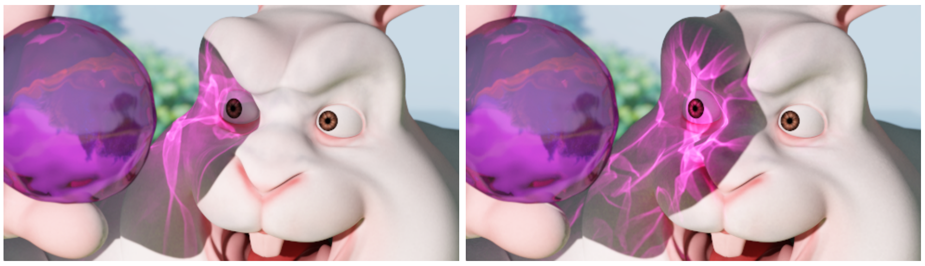

Figure 9.9: An example of path–proxy linking with shadow and caustic proxies in the *Bunny* scene. Left: original rendering without manipulation. Right: the shadow is displaced, rotated, and enlarged to cover more of the face, and the caustic is moved, squished, and rotated to highlight the bunny's expression of "evil intent." Image taken from [200].

is plausibly computed from the manipulated light transport. Fig. 9.10 demonstrates light manipulation using path retargeting in a complex architectural scene. We applied two edits to LD^RX_P paths, i.e. indirect diffuse illumination, "stretching" the color bleeding across the floor and down the stairs. Similar effects can be obtained with light linking, but would entail a significant effort. We note that lighting design becomes increasingly complex and cluttered when light linking is overused.

In Fig. 9.9, path–proxy linking is used to position and scale the shadow (LX_P paths) and refractive caustic ($LS^TS^TX_P$) independently, i.e. using two proxy objects.

Fig. 9.10 shows edits of diffuse global illumination in an architectural scene. While dramatic edits of lighting are possible with our system, more subtle edits like in this example are another viable use case for light transport manipulation.

Fig. 9.11 shows several sequential manipulations in a simple box scene, combining

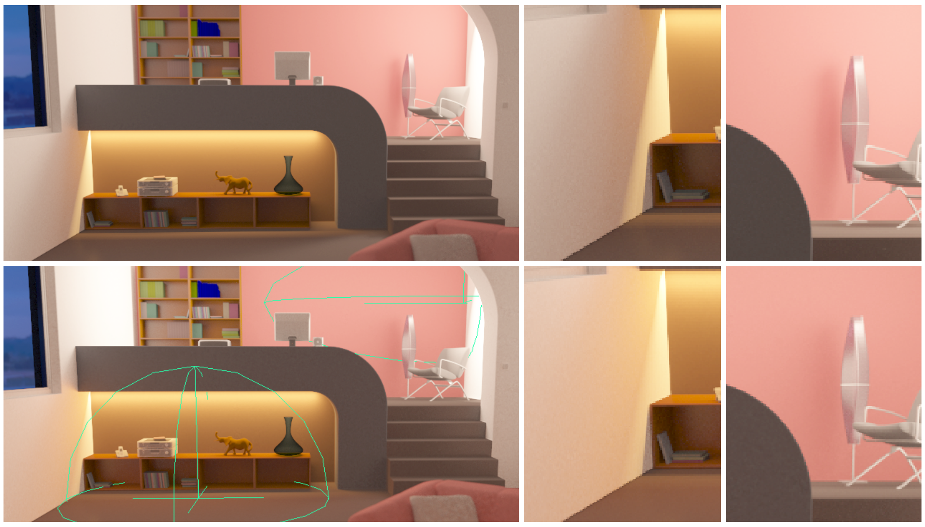

Figure 9.10: Manipulating diffuse global illumination in the *Living Room* scene. Top to bottom: original global illumination solution; indirect diffuse light above the shelf and on the back wall are manipulated with path retargeting. Image taken from [200].

both retargeting and linking and illustrating secondary effects, which would be tedious to edit with light linking techniques (where multiple lights would need to be generated, some in order to induce fake indirect light).

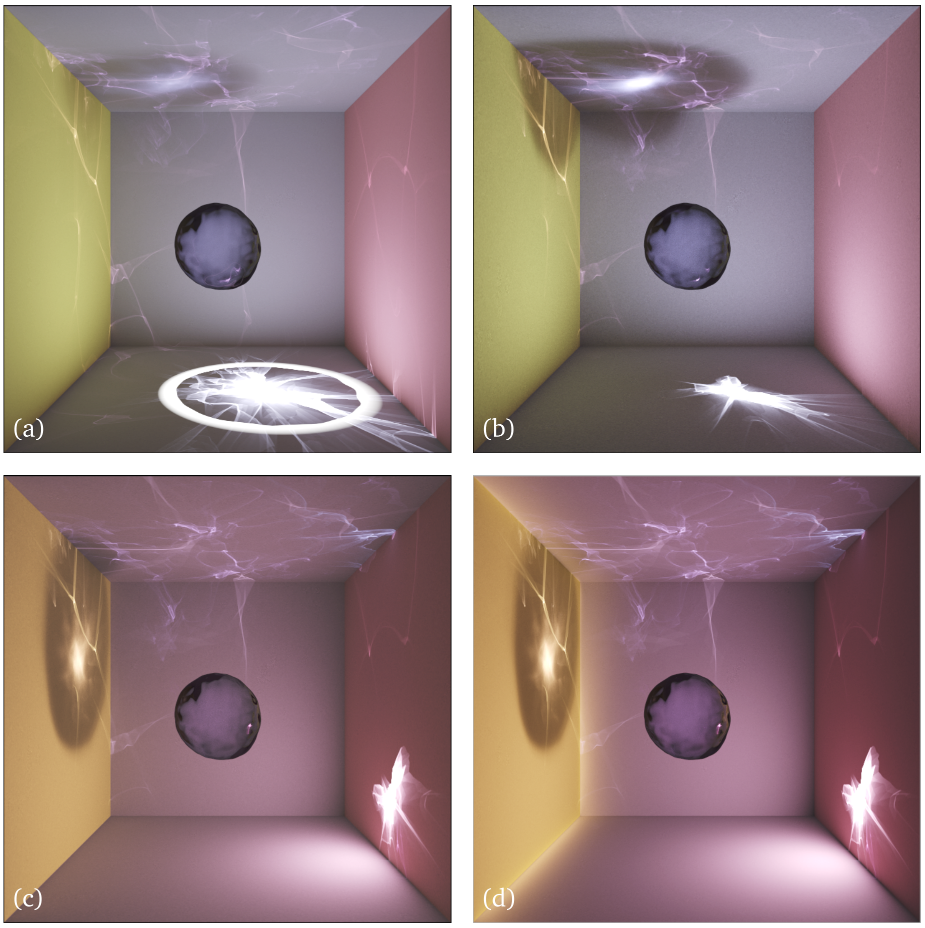

Figure 9.11: Multiple light edits of a bumpy glass sphere lit by a static spotlight. (a) Original render, (b) path–proxy linking enlarges the shadow, (c) retargeting moves the caustic to the right wall, and (d) another retargeting stretches the diffuse indirect lighting from the left wall ($L(.)^{*}D_{\text{leftwall}}$) *Note how edits (b) and (c) not only move lighting features, but automatically create consistent and plausible secondary effects.* Image taken from [200].

9.8 Domain Expert Feedback

In early 2013, we conducted a survey to assess the usefulness and potential of our manipulation tools and our selection and visualization approaches. Our survey included five technical directors (TDs) at *Filmakademie Baden-Württemberg*, a local film academy, all specialized in lighting and shading. They attested to being very familiar with PBR techniques and having significant experience with industry-standard graphics packages such as Chaos Group's V-Ray, Autodesk's Maya and Max, Side Effects' Houdini, and Pixar's RenderMan. Each TD confirmed that PBR is rapidly being adopted in production, making existing tools obsolete.[5] For fine-tuning and light manipulation they traditionally use light linking, but they also pointed out that it can be tedious and time-consuming in complex scenes, yet often the only available effective tool.

During TD interviews, we demonstrated the use of our tools in the *Bunny*, *Jewelry* (Fig. 9.2), *Garage*, and *Buddha* (Fig. 9.3) scenes. Afterwards, we let the TDs experiment with our tools for a few minutes and then presented them with several editing tasks: we asked them to use manipulators to retarget the caustic in the *Bunny* scene, i.e., to focus it, change its color, and move it to a more visually pleasing location. All TDs were able to perform these actions in less than a minute. In the *Buddha* scene we asked them to reshape the caustic with path retargeting and rotate the shadow with path–proxy linking. These editing tasks were accomplished in less than two minutes. The TDs were pleased with the secondary effects (e.g., inter-reflected caustics) being consistent with the edits.

TDs also identified several instances where our tools simplified lighting design, such as being able to freely manipulate caustics without iteratively tweaking material properties like the index of refraction. They additionally identified potential extensions: for instance, film production may employ and reuse fixed lighting rigs across several scenes, and these rigs could also be adapted on a per-scene basis according to artistic requirements using our tools.

Interactive feedback (selection, visualization and manipulation) coupled with instantaneous GI preview was positively received by every TD. Sketch-based selection was generally preferred for gizmo placement. TDs also stated that our visualization served as a powerful tool—even on its own—for increasing scene understanding, and that bundling path lines reduces visual clutter in complex lighting scenarios. They reported that they do not know of any visualization tools that provide more than a photon density map on surfaces, and hence could imagine using our visualization as a stand-alone tool to identify where light is coming from and "debug" scenes. Finally, we asked them to provide feedback on the main components, namely visualization, path retargeting, and path–proxy linking.

TDs discussed the importance of matching even the smallest details set by art direction and stated that our tool's support for multiple and independent local edits would significantly simplify these tasks. Furthermore, the ability to apply these edits while having access to a progressive GI preview facilitates the artistic iteration process. TDs also emphasized that our approach propagates indirect PBR shading effects after editing operations, avoiding approximations of segmenting these effects and simply pre-baking them into textures.

All TDs emphasized the importance of animation support, and confirmed that the ability to key-frame our editing operations was an essential design decision. Workflow-wise, they suggested that artists should be permitted to pick a set of the most important key-frames, which could be cached to speed up previews between them.

[5] Since the survey was taken, personal communication with researchers and professionals has confirmed this trend further.

9.9 Robust Bidirectional Transport

Our tools can also be formulated in Veach's transport framework [229], enabling the seamless use of our approach in light transport methods such as forward path tracing, (full) bidirectional path tracing and Metropolis Light Transport [231].

Path–Proxy Linking Here, different scene representations are used for different types of paths. For sampling, when path length is unknown prior to path construction (as in BDPT), Russian roulette should be applied to decide which representation to use: once a path is formed, it is rejected if it does not exist for the chosen representation. If, after subpath identification, we know that a representation will not match, it is not selected. Our probabilistic proxy/path selection (see Sec. 9.5.2) increases noise since new paths can be discarded; this increase of noise can be eliminated if *all* $N+1$ proxies are traced against and only the path generated by the one valid proxy is retained. This scheme would however increase computation cost, such that overall efficiency may be worse.

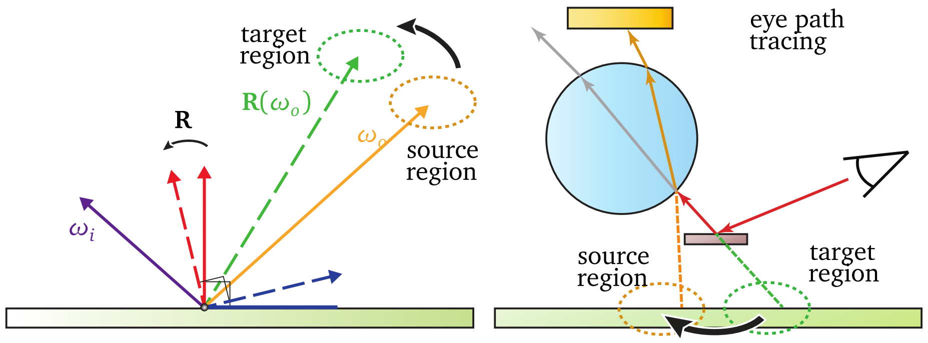

Figure 9.12: Left: path retargeting can be viewed as a rotation of either the incident or exitant tangent frames, depending on the manipulation (whether light or eye subpaths are modified) and the “direction” of the sampling method (either importance or radiance). Right: when we trace opposite to the manipulation direction, the inverse transformation has to be applied at each interaction to check for potential manipulations from another side. Image taken from [200].

Path Retargeting Retargeting effectively rotates *a surface's outgoing tangent frame* such that an outgoing segment points towards the user-specified target (Fig. 9.12, left). Similarly to vertex-interpolated and/or bump-mapped shading normals, this introduces a non-symmetric BSDF [229] for bidirectional transport. Given a surface point with normal N_x, BSDF f_s, and a path vertex with incident and outgoing directions ω_i and ω_o, path retargeting defines a rotation $\mathbf{R}$ of the tangent frame; the modified BSDF is:

$$f_s'(x,\omega_o,\omega_i) = f_s(x,\mathbf{R}(\omega_o),\omega_i),$$

and its adjoint is derived similarly to [229, Sec. 5.3.2] as:

$$f_s'^{*}(x,\omega_o,\omega_i) = f_s(x,\omega_i,\mathbf{R}^T(\omega_o))\frac{\left|\mathbf{R}^T(\omega_o)\cdot N_x\right|}{\left|\omega_o\cdot N_x\right|}.$$

The adjoint BSDF $f_s'^{*}$ should be used when the sampling and manipulation directions are inverted. Retargeting constrained to a source region must be properly treated using rejection sampling for paths from the direction opposite the manipulation direction. If, e.g., an eye subpath is constructed for a path that has been modified from the light direction, the inverse source–target transformation must be applied to ensure that the original light subpath hits the source region (Fig. 9.12, right); otherwise such a path is rejected.

9.10 Conclusion

In this chapter, we have presented a general approach to understanding and artistically editing the process of physically based rendering (PBR), via operation in the space of all possible light paths. Within this framework, we have developed a specific interactive shading editing system for intuitive visualization, selection, and manipulation of PBR. Path-space processing, semi-automatic selection/ranking approaches, visualization of selected light transport components, and direct and indirect manipulation techniques combine to form the core of the system.

Since we interactively process path space while it is being constructed, i.e., in a progressive rendering context, we can utilize any PBR back-end that can generate paths, avoiding the simulation or precomputation of any metadata, while our manipulations utilize hooks easily exposed by the renderer, such as ray re-shooting and adjustment of local shading parameters.

By integrating into an existing DCC system, artists can easily familiarize themselves with our tools and quickly complete nontrivial lighting design tasks in PBR contexts. Many of our ideas complement or extend existing editing metaphors we have seen in Ch. 7, such as light linking. Of the use cases we cooperatively identified with TDs, we primarily focused on those where intuitive PBR manipulation seemed most viable. Smart selection and retargeting/path–proxy linking handle these (primarily directional) effects quite well.[6] Given the feedback from our small survey of experienced users, we believe that our tool can be of great help in the current industrial-strength PBR art pipelines.

[6]We note that path retargeting can be used—albeit less intuitively—to manipulate smooth diffuse light as well (see Fig. 9.10).

Chapter 10

Conclusions

An ideal world is left as an exercise to the reader.

Paul Graham, *On Lisp*

In this thesis, we have covered a range of topics, beginning with improving general aspects of computer graphics, to specific physically based rendering methods, to our end goal of enabling artists to selectively take control over the image formation process, in a wider context beyond just simulating light transport accurately. Granted, that does not mean that we consider light transport simulation to be a solved problem, both in terms of efficiency and in what can be even rendered at all. As always, research is an ongoing process.

Starting with the basics of image synthesis, we have discovered that taking into account a given display's subpixel structure yields better filtering methods that enhance perceived display resolution at a very low cost. Our work extends previous approaches to arbitrary subpixel patterns of diverse displays, can be applied to common subproblems of computer graphics, such as anti-aliasing in rasterization or texture filtering, and has been perceptually verified in a user study.

Moving from the general realm of creating images to explicitly rendering *realistic* images, we extensively covered the process of physically based rendering, which refers to the accurate numerical simulation of light transport along the paths prescribed by the model of geometric optics. In practice, this model is sufficient to achieve photorealism for many scenes, and has recently seen wide adoption across fields.

After covering important aspects such as the expression of light transport as a sampling process in the space of all light paths, we noticed, among other things, the difficulty of efficiently rendering scenes with general participating media. Subsequently, we discovered a novel, scalable rendering method, based on the process of empirically analyzing and approximating the bias compensation process suggested by earlier work. Our approximations are such that the method still creates visually accurate results, but warrants an efficient implementation on modern parallel throughput processors.

Finally, we mentioned that the adoption of the simulation-centered methodology of physically based rendering has led to a loss of intuitive, fine-grained, and local control over the resulting image, which was present in earlier, less rigorous paradigms. With this premise, we moved our focus to the high-level purpose of enabling artistic control of physically based rendering, exploring the ground between physically accurate light transport simulation on one side, and artistic freedom on the other.

We saw that the artistic editing and design of material and lighting has been widely considered in research, but not much has been published about artistically editing global illumination in a general manner. Thus, we proposed our central idea that visualization, selection, and editing operations should be performed directly in the space encompassing all possible light paths, as opposed to current methods, which either work in image space or are tailored to specific, isolated lighting effects such as mirror reflections, shadows, or caustics. We realized our idea through a novel technique for direct visualization of light paths, as a visual understanding of light is required before effectively manipulating it.

We then introduced a specific system for the artistic editing of physically based rendering, exposing to artists intuitive direct and indirect manipulation approaches to edit complex lighting effects such as (multiply-refracted) caustics, diffuse and glossy indirect bounces, and direct or indirect shadows. Our sketch- and object-space selection allows artists to effectively isolate shading effects to inspect and edit. We classify and filter light paths on the fly, as they are generated by a (progressive) rendering method, and visualize the selected transport phenomena at the same time. We presented two light path manipulation approaches, *path retargeting* and *path–proxy linking*, which are examples of editing lighting effects in path space. Finally, we surveyed artists who used our tool to manipulate complex phenomena in both static and animated scenes.

10.1 Limitations

We can imagine situations in which our approach does not provide a suitable answer to artistic editing requirements.

As we always begin manipulation from an initial global illumination solution (to promote a physically plausible final result), our tool cannot easily create completely new lighting scenarios, which light linking can achieve.

As summarized in Ch. 8, Reiner et al. [189] show that transport visualization is challenging, with no single technique that works well in all cases. Indeed, we found visualization of diffuse indirect light to be difficult, as this phenomenon is less localized than, say, caustics. However, bundling helps even for diffuse light when multiple manipulators are used, as paths are contracted and different transport bundles are isolated.

In contrast to Ritschel et al.'s approach [193], the shape of, say, a caustic may change during manipulation, since we re-trace manipulated photons. Exaggerated manipulation may also cause completely unlit (source) regions and detachment of shadows.

10.2 Future Work

Several interesting directions for future work can stem from our investigation.

Path–proxy linking can be more closely integrated into existing light linking tools, or extended with more flexible proxy generation operations, e.g., using a completely different proxy object.

As mentioned above, strong displacements from path retargeting can result in self-intersections, which we could imagine avoiding with back-propagation (akin to inverse kinematics) of the manipulated segment's deformation, or using manifold walks [87] to solve for valid retargeting directions in a goal-based manner. Second, our retargeting manipulator is unidirectional, reducing the performance of bidirectional rendering methods. Investigating how to robustly adapt retargeting to estimators sampling paths in the opposite directions remains for future work.

As we have hinted in Sec. 9.6.2, extending our approach to volumetric scattering in participating media is straightforward, but also poses several interesting challenges:

For light transport visualization, dense media, where path segments are very short, are one scenario where even bundling does not help reduce clutter, as the bundles themselves become too short. We have started experimenting with interactive *time-of-flight* methods for visualization, which only show a small portion of photon trajectories at a time, akin to transient rendering [88]. These methods complement our bundling approach, but a thorough investigation remains for future work.

Along the same lines, volumetric lighting can be easily identified—using extended path notation and spatial selection—and edited with either simple gain/hue controls or path-proxy linking, but *path retargeting* does not pose any useful manipulation due to the short length of path segments in dense media. This is inherent in the technique's approach to explicitly modify path segments.

10.3 Final Thoughts

Veach and Guibas [229–231] pioneered the path-space formulation of light transport, together with algorithms which more effectively explore important paths in this infinite-dimensional space. Jakob and Marschner [85,87], for example, later proposed a technique to better explore specular and near-specular paths, thus exploring path space through simulation. Our system, on the other hand, allows users to visualize and edit path-space representations of light transport in a scene, and can thus be viewed as exposing an artist-driven exploration and manipulation of path space.

Acknowledgments

A work like this dissertation is not created in isolation and I would like to express my gratitude towards a few people.

Carsten Dachsbacher, for supervising my work at KIT, and ultimately helping transform what once started as dabbling in graphics, into what now is a career.

Derek Nowrouzezahrai, for mentoring my extended stay in Montréal, where I also met various other cool people such as Gilles-Phillippe Paillé, Eric Heitz, Jonathan Dupuy, Arnaud Emilien, and Pierre Poulin.

All my numerous coauthors, without whom this thesis would not exist. I also would have traveled a lot less.

All my former colleagues at the computer graphics group at KIT. Notably, Boris Neubert, for taking interest in co-supervising my thesis. Diana Kheil for being around and helpful in general. And especially those of the very early days: Jan Novák, Thomas Engelhardt, Gábor Liktor and Tim Reiner.

I am especially thankful to Hauke Rehfeld, Max-Gerd Retzlaff, and Florian Simon, who reviewed early drafts of this thesis.

My fellow co-founders at Lightrig, for helping move this idea from research into the industry. Christophe Hery, for taking interest in our work early on.

Various friends and acquaintances, for offering ample distraction from writing this thesis.

And my family, for encouraging me to finish it after all.

Bibliography

[1] Tomas Akenine-Möller, Eric Haines, and Naty Hoffman. *Real-Time Rendering, 3rd Edition*. A. K. Peters, Ltd, 2008.

[2] Oskar Åkerlund, Mattias Unger, and Rui Wang. Precomputed visibility cuts for interactive relighting with dynamic BRDFs. In *IEEE Computer Graphics and Applications*, pages 161–170. IEEE Computer Society, 2007.

[3] Xiaobo An and Fabio Pellacini. AppProp: All-pairs appearance-space edit propagation. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 27(3):40:1–40:9, August 2008.

[4] Xiaobo An, Xin Tong, Jonathan D. Denning, and Fabio Pellacini. AppWarp: Retargeting measured materials by appearance-space warping. *ACM Transactions on Graphics (Proc. of SIGGRAPH Asia)*, 30(6):147:1–147:10, December 2011.

[5] Bradley Atcheson. Subpixel rendering of Bayer-patterned images. `http://www.cs.ubc.ca/~atcheson/projects.html`, 2005. Last retrieved on 2012-07-03.

[6] David C. Banks. Illumination in diverse codimensions. In *Proc. SIGGRAPH*, pages 327–334, 1994.

[7] Ilya Baran, Jiawen Chen, Jonathan Ragan-Kelley, Frédo Durand, and Jaakko Lehtinen. A hierarchical volumetric shadow algorithm for single scattering. In *ACM Transactions on Graphics (Proc. of SIGGRAPH Asia)*, pages 178:1–178:10, 2010.

[8] Ronen Barzel. Lighting controls for computer cinematography. *Journal of Graphics Tools*, 2(1):1–20, 1997.

[9] Sean Bell, Paul Upchurch, Noah Snavely, and Kavita Bala. OpenSurfaces: a richly annotated catalog of surface appearance. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 32(4):111:1–111:17, August 2013.

[10] Aner Ben-Artzi, Kevin Egan, Ravi Ramamoorthi, and Frédo Durand. A precomputed polynomial representation for interactive BRDF editing with global illumination. *ACM Transactions on Graphics*, 27(2):13:1–13:13, May 2008.

[11] Aner Ben-Artzi, Ryan Overbeck, and Ravi Ramamoorthi. Real-time BRDF editing in complex lighting. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 25(3):945–954, July 2006.

[12] Benedikt Bitterli. The secret life of photons - simulating 2D light transport. `https://benedikt-bitterli.me/tantalum/`, 2015. Last retrieved on 2017-05-18.

[13] Adrien Bousseau, Emmanuelle Chapoulie, Ravi Ramamoorthi, and Maneesh Agrawala. Optimizing environment maps for material depiction. *Computer Graphics Forum (Proceedings of EG Symposium on Rendering)*, 30(4):1171–1180, 2011.

[14] Normand Brière and Pierre Poulin. Hierarchical view-dependent structures for interactive scene manipulation. In *Proc. SIGGRAPH*, pages 83–90, 1996.

[15] Brent Burley. Physically-based shading at Disney. In *Practical physically-based shading in film and game production*, Proc. SIGGRAPH (Courses), pages 10:1–10:7, 2012.

[16] F. W. Campbell and J. G. Robson. Application of Fourier analysis to the visibility of gratings. *Journal of Physiology*, 197(3):551–566, 1968.

[17] Eva Cerezo, Frederic Perez-Cazorla, Xavier Pueyo, Francisco Seron, and François X. Sillion. A survey on participating media rendering techniques. *The Visual Computer*, 21:303–328, 2005.

[18] Matthäus G. Chajdas, Andreas Weis, and Rüdiger Westermann. Assisted environment map probe placement. In *Proc. SIGRAD*, 2011.

[19] Subrahmanyan Chandrasekhar. *Radiative Transfer*. Dover Books on Intermediate and Advanced Mathematics. Dover Publications, 1960.

[20] Ewen Cheslack-Postava, Rui Wang, Oskar Åkerlund, and Fabio Pellacini. Fast, realistic lighting and material design using nonlinear cut approximation. *ACM Transactions on Graphics (Proc. of SIGGRAPH Asia)*, 27(5), 2008.

[21] Per H. Christensen, Julian Fong, David M. Laur, and Dana Batali. Ray tracing for the movie 'cars'. In *Proc. of IEEE Symposium on Interactive Ray Tracing*, pages 1–6, 2006.

[22] Per H. Christensen and Wojciech Jarosz. The path to path-traced movies. *Foundation and Trends in Computer Graphics and Vision*, 10(2):103–175, 2016.

[23] David Cline and Parris Egbert. A practical introduction to Metropolis light transport. Brigham Young University Technical Report, 2005.

[24] David Cline, Justin Talbot, and Parris Egbert. Energy redistribution path tracing. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 24(3):1186–1195, 2005.

[25] Michael F. Cohen and John R. Wallace. *Radiosity and Realistic Image Synthesis*. Morgan Kaufmann Publishers Inc., Natick, MA, USA, 1993.

[26] Mark Colbert. *Appearance-driven Material Design*. PhD thesis, School of Electrical Engineering and Computer Science, University of Central Florida, 2008.

[27] Mark Colbert, Sumanta Pattanaik, and Jaroslav Křivánek. BRDF-Shop: Creating physically correct bidirectional reflectance distribution functions. *IEEE Computer Graphics and Applications*, 26(1):30–36, 2006.

[28] Mark Colbert, Erik Reinhard, and Charles Edward Hughes. Painting in high dynamic range. *Journal of Visual Communication and Image Representation*, 18(5):387–396, 2007.

[29] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. *Computer Graphics (Proc. SIGGRAPH)*, 18(3):137–145, July 1984.

[30] António Cardoso Costa, António Augusto de Sousa, and Fernando Nunes Ferreira. Lighting design: A goal based approach using optimization. In *Proc. EG Workshop on Rendering*, June 1999.

[31] Antonio Criminisi, Martin Kemp, and Sing Bing Kang. Reflections of reality in Jan van Eyck and Robert Campin. *Historical Methods*, 37, January 2004.

[32] Franklin C. Crow. Shadow algorithms for computer graphics. *Computer Graphics (Proc. SIGGRAPH)*, 1(2):242–248, 1977.

[33] Carsten Dachsbacher, Jaroslav Křivánek, Miloš Hašan, Adam Arbree, Bruce Walter, and Jan Novák. Scalable realistic rendering with many-light methods. *Computer Graphics Forum*, 33(1):88–104, 2014.

[34] Carsten Dachsbacher and Marc Stamminger. Reflective shadow maps. In *Proc. SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 203–213, 2005.

[35] Carsten Dachsbacher, Marc Stamminger, George Drettakis, and Frédo Durand. Implicit visibility and antiradiance for interactive global illumination. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 26, 2007.

[36] Tomáš Davidovič, Jaroslav Křivánek, Miloš Hašan, Philipp Slusallek, and Kavita Bala. Combining global and local virtual lights for detailed glossy illumination. In *ACM Transactions on Graphics (Proc. of SIGGRAPH Asia)*, 2010.

[37] Eugene d'Eon and Geoffrey Irving. A quantized-diffusion model for rendering translucent materials. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 30(4), July 2011.

[38] Piotr Didyk, Elmar Eisemann, Tobias Ritschel, Karol Myszkowski, and Hans-Peter Seidel. Apparent display resolution enhancement for moving images. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 29(4), 2010.

[39] Yue Dong, Xin Tong, Fabio Pellacini, and Baining Guo. AppGen: Interactive material modeling from a single image. *ACM Transactions on Graphics (Proc. of SIGGRAPH Asia)*, 30(6):146:1–146:10, December 2011.

[40] Julie Dorsey, Holly E. Rushmeier, and François X. Sillion. *Digital Modeling of Material Appearance*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.

[41] Tom Duff. Polygon scan conversion by exact convolution. *Proceedings International Conference on Raster Imaging and Digital Typography*, pages 151–168, 1989.

[42] Philip Dutré, Philippe Bekaert, and Kavita Bala. *Advanced Global Illumination, 2nd Edition*. A. K. Peters, Ltd, 2006.

[43] Candice H. Brown Elliott, Thomas L. Credelle, and Michael F. Higgins. Adding a white subpixel. *Information Display*, 21(5), 2005.

[44] Eric Enderton and Daniel Wexler. The workflow scale: Why 5x faster might not be enough. In *Computer Graphics International*, June 2011.

[45] Thomas Engelhardt and Carsten Dachsbacher. Epipolar sampling for shadows and crepuscular rays in participating media with single scattering. In *Proc. SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 119–125, 2010.

[46] Thomas Engelhardt, Jan Novák, Thorsten-Walther Schmidt, and Carsten Dachsbacher. Approximate bias compensation for rendering scenes with heterogeneous participating media. *Computer Graphics Forum (Proceedings of Pacific Graphics)*, 31(7):2145–2154, September 2012.

[47] Thomas Engelhardt, Thorsten-Walther Schmidt, Jan Kautz, and Carsten Dachsbacher. Low-cost subpixel rendering for diverse displays. *Computer Graphics Forum*, 33(1):199–209, February 2013.

[48] Christer Ericson. *Real-Time Collision Detection*. The Morgan Kaufmann Series in Interactive 3-D Technology. Morgan Kaufmann Publishers Inc., 2005.

[49] Lu Fang, Oscar C. Au, Yi Yang, Weiran Tang, and Xing Wen. A new adaptive subpixel-based downsampling scheme using edge detection. In *Circuits and Systems (Proceedings of ISCAS)*, pages 3194–3197, 2009.

[50] Raanan Fattal. *ACM Transactions on Graphics*, 28(1):1–11, 2009.

[51] Tony Finch. Incremental calculation of weighted mean and variance. University of Cambridge Computing Service Technical Report, February 2009.

[52] R.A. Fisher. *Statistical methods for research workers*. Oliver & Boyd, Edinburgh, Scotland, UK, 1925.

[53] E. W. Forgy. Cluster analysis of multivariate data: efficiency versus interpretability of classifications. *Biometrics*, 21(3):768—-769, September 1965.

[54] Jean-Baptiste Joseph Fourier. *Théorie analytique de la chaleur.* Firmin Didot, père et fils, Paris, 1822.

[55] Iliyan Georgiev, Jaroslav Křivánek, Tomáš Davidovič, and Philipp Slusallek. Light transport simulation with vertex connection and merging. *ACM Transactions on Graphics (Proc. of SIGGRAPH Asia)*, 31(6):192:1–192:10, 2012.

[56] Iliyan Georgiev and Philipp Slusallek. Simple and robust iterative importance sampling of virtual point lights. In *Proc. Eurographics (Short Papers)*, pages 57–60, 2010.

[57] Reid Gershbein and Patrick M. Hanrahan. A fast relighting engine for interactive cinematic lighting design. In *Proc. SIGGRAPH*, pages 353–358, July 2000.

[58] Ioannis Gkioulekas, Bei Xiao, Shuang Zhao, Edward H. Adelson, Todd Zickler, and Kavita Bala. Understanding the role of phase function in translucent appearance. *ACM Transactions on Graphics*, 32(5):147:1–147:19, 2013.

[59] Andrew Glassner. *Principles of Digital Image Synthesis*. Morgan Kaufmann Publishers Inc., Natick, MA, USA, 1995.

[60] Bruce Gooch and Amy Gooch. *Non-Photorealistic Rendering*. AK Peters/CRC Press, Natick, MA, USA, 2001.

[61] Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, and Bennett Battaile. Modeling the interaction of light between diffuse surfaces. *Computer Graphics (Proc. SIGGRAPH)*, 18(3):213–222, July 1984.

[62] Ned Greene and Paul S. Heckbert. Creating raster omnimax images from multiple perspective views using the elliptical weighted average filter. *IEEE Computer Graphics and Applications*, 6(6):21–27, 1986.

[63] Tobias Günther, Kai Rohmer, Christian Rössl, Thorsten Grosch, and Holger Theisel. Stylized caustics: Progressive rendering of animated caustics. *Computer Graphics Forum (Proceedings of EG Symposium on Rendering)*, 35(2), 2016.

[64] Toshiya Hachisuka and Henrik Wann Jensen. Stochastic progressive photon mapping. *ACM Transactions on Graphics*, 28:141:1–141:8, 2009.

[65] Toshiya Hachisuka, Anton S. Kaplanyan, and Carsten Dachsbacher. Multiplexed Metropolis light transport. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 33(4), August 2014.

[66] Toshiya Hachisuka, Shinji Ogaki, and Henrik Wann Jensen. Progressive photon mapping. *ACM Transactions on Graphics (Proc. of SIGGRAPH Asia)*, 27(5):130:1–130:8, 2008.

[67] Toshiya Hachisuka, Jacopo Pantaleoni, and Henrik Wann Jensen. A path space extension for robust light transport simulation. *ACM Transactions on Graphics (Proc. of SIGGRAPH Asia)*, 31(6):191:1–191:10, 2012.

[68] Fabian Hahn, Sebastian Martin, Bernhard Thomaszewski, Bob Sumner, Stelian Coros, and Markus Gross. Rig-space physics. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 31(4), July 2012.

[69] Johannes Hanika and Carsten Dachsbacher. Efficient Monte Carlo rendering with realistic lenses. In *Computer Graphics Forum (Proceedings Eurographics)*, volume 33, 2014.

[70] Johannes Hanika, Anton S. Kaplanyan, and Carsten Dachsbacher. Improved half vector space light transport. *Computer Graphics Forum (Proceedings of EG Symposium on Rendering)*, 34(4), 2015.

[71] Miloš Hašan, Jaroslav Křivánek, Bruce Walter, and Kavita Bala. Virtual spherical lights for many-light rendering of glossy scenes. *ACM Transactions on Graphics (Proc. of SIGGRAPH Asia)*, 28(5):143:1–143:6, 2009.

[72] Miloš Hašan, Fabio Pellacini, and Kavita Bala. Direct-to-indirect transfer for cinematic relighting. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 25(3):1089–1097, July 2006.

[73] Miloš Hašan, Fabio Pellacini, and Kavita Bala. Matrix row-column sampling for the many-light problem. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 26, 2007.

[74] Miloš Hašan and Ravi Ramamoorthi. Interactive albedo editing in path-traced volumetric materials. *ACM Transactions on Graphics*, 32(2):11:1–11:11, April 2013.

[75] Paul S. Heckbert. Adaptive radiosity textures for bidirectional ray tracing. *Computer Graphics (Proc. SIGGRAPH)*, 24(4):145–154, 1990.

[76] Danny Holten and Jarke J. van Wijk. Force-directed edge bundling for graph visualization. *Computer Graphics Forum*, 28(3):983–990, 2009.

[77] Lukas Hošek and Alexander Wilkie. An analytic model for full spectral skydome radiance. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 31(4), July 2012.

[78] Lukas Hošek and Alexander Wilkie. Adding a solar radiance function to the Hošek-Wilkie skylight model. *IEEE Computer Graphics and Applications*, 33(3):44–52, May 2013.

[79] John F. Hughes, Andries van Dam, Morgan McGuire, David F. Sklar, James D. Foley, Steven K. Feiner, and Kurt Akeley. *Computer Graphics: Principles and Practice, 3rd Edition*. Addison-Wesley Professional, Boston, MA, USA, 2013.

[80] Matthias B. Hullin, Johannes Hanika, and Wolfgang Heidrich. Polynomial optics: A construction kit for efficient ray-tracing of lens systems. *Computer Graphics Forum (Proceedings of EG Symposium on Rendering)*, 31(4), 2012.

[81] Homan Igehy. Tracing ray differentials. In *Computer Graphics (Proc. SIGGRAPH)*, pages 179–186, 1999.

[82] David S. Immel, Michael F. Cohen, and Donald P. Greenberg. A radiosity method for non-diffuse environments. In *Proc. SIGGRAPH*, pages 133–142, New York, NY, USA, 1986.

[83] Kei Iwasaki, Yoshinori Dobashi, and Tomoyuki Nishita. Interactive bi-scale editing of highly glossy materials. *ACM Transactions on Graphics*, 31(6):144:1–144:7, 2012.

[84] Wenzel Jakob. Mitsuba renderer. `http://www.mitsuba-renderer.org/`, 2010. Last retrieved on 2017-02-02.

[85] Wenzel Jakob. *Light Transport Methods on Path-Space Manifolds*. PhD thesis, Cornell University, August 2013.

[86] Wenzel Jakob, Adam Arbree, Jonathan T. Moon, Kavita Bala, and Stephen R. Marschner. A radiative transfer framework for rendering materials with anisotropic structure. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 29(4):53:1–53:13, July 2010.

[87] Wenzel Jakob and Steve Marschner. Manifold exploration: A Markov chain Monte Carlo technique for rendering scenes with difficult specular transport. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 31(4), July 2012.

[88] Adrian Jarabo, Julio Marco, Adolfo Muñoz, Raul Buisan, Wojciech Jarosz, and Diego Gutierrez. A framework for transient rendering. *ACM Transactions on Graphics (Proc. of SIGGRAPH Asia)*, 33(6), 2014.

[89] Adrian Jarabo, Belen Masia, Adrien Bousseau, Fabio Pellacini, and Diego Gutierrez. How do people edit light fields? *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 33(4), 2014.

[90] Wojciech Jarosz, Craig Donner, Matthias Zwicker, and Henrik Wann Jensen. Radiance caching for participating media. Proc. SIGGRAPH (Sketches), 2008.

[91] Wojciech Jarosz, Derek Nowrouzezahrai, Iman Sadeghi, and Henrik Wann Jensen. A comprehensive theory of volumetric radiance estimation using photon points and beams. *ACM Transactions on Graphics (Proc. of SIGGRAPH Asia)*, 30(1):5:1–5:19, 2011.

[92] Wojciech Jarosz, Derek Nowrouzezahrai, Robert Thomas, Peter-Pike Sloan, and Matthias Zwicker. Progressive photon beams. *ACM Transactions on Graphics (Proc. of SIGGRAPH Asia)*, 30(6):181:1–181:12, 2011.

[93] Wojciech Jarosz, Volker Schönefeld, Leif Kobbelt, and Henrik Wann Jensen. Theory, analysis and applications of 2D global illumination. *ACM Transactions on Graphics*, 31(5):125:1–125:21, September 2012.

[94] Wojciech Jarosz, Matthias Zwicker, and Henrik Wann Jensen. The beam radiance estimate for volumetric photon mapping. *Computer Graphics Forum (Proceedings Eurographics)*, 27(2):557–566, 2008.

[95] Henrik Wann Jensen. *Realistic image synthesis using photon mapping*. A. K. Peters, 2001.

[96] Henrik Wann Jensen and Per H. Christensen. Efficient simulation of light transport in scences with participating media using photon maps. In *Proc. SIGGRAPH*, pages 311–320, 1998.

[97] Henrik Wann Jensen, Stephen R. Marschner, Marc Levoy, and Patrick M. Hanrahan. A practical model for subsurface light transport. In *Proc. SIGGRAPH*, pages 511–518, 2001.

[98] Jorge Jimenez, Károly Zsolnai, Adrian Jarabo, Christian Freude, Thomas Auzinger, Xian-Chun Wu, Javier der Pahlen, Michael Wimmer, and Diego Gutierrez. Separable subsurface scattering. *Computer Graphics Forum*, 34(6):188–197, 2015.

[99] Garrett M. Johnson and Mark D. Fairchild. The effect of opponent noise on image quality. In *Proceedings of SPIE*, volume 5668, pages 82–89, 2005.

[100] James T. Kajiya. The rendering equation. In *Proc. SIGGRAPH*, pages 143–150, New York, NY, USA, 1986.

[101] James T. Kajiya and Michael Karl Ullner. Filtering high quality text for display on raster scan devices. In *Proc. SIGGRAPH*, pages 7–15, 1981.

[102] James T. Kajiya and Brian P. Von Herzen. Ray tracing volume densities. *Computer Graphics (Proc. SIGGRAPH)*, 18(3):165–174, July 1984.

[103] Anton S. Kaplanyan and Carsten Dachsbacher. Adaptive progressive photon mapping. *ACM Transactions on Graphics*, 32(2):16:1–16:19, 2013.

[104] Anton S. Kaplanyan and Carsten Dachsbacher. Path space regularization for holistic and robust light transport. *Computer Graphics Forum (Proceedings Eurographics)*, 32(2):63–72, 2013.

[105] Anton S. Kaplanyan, Johannes Hanika, and Carsten Dachsbacher. The natural-constraint representation of the path space for efficient light transport simulation. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 33(4):129:1–129:11, August 2014.

[106] Ondřej Karlík, Martin Růžička, Václav Gassenbauer, Fabio Pellacini, and Jaroslav Křivánek. Toward evaluating the usefulness of global illumination for novices in lighting design tasks. *IEEE Trans. on Visualization and Computer Graphics*, 20(6):944–954, June 2013.

[107] Jan Kautz, Solomon Boulos, and Frédo Durand. Interactive editing and modeling of bidirectional texture functions. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 26(3):53:1–53:10, July 2007.

[108] Jan Kautz, Peter-Pike Sloan, and John Snyder. Fast, arbitrary BRDF shading for low-frequency lighting using spherical harmonics. In *Proc. EG Workshop on Rendering*, pages 291–296, June 2002.

[109] John K. Kawai, James S Painter, and Michael F. Cohen. Radioptimization: Goal based rendering. In *Proc. SIGGRAPH*, pages 147–154, New York, NY, USA, 1993. ACM.

[110] Csaba Kelemen and László Szirmay-Kalos. Simple and robust mutation strategy for Metropolis light transport algorithm. TU Vienna Technical Report, 2001.

[111] Alexander Keller. Instant radiosity. In *Computer Graphics (Proc. SIGGRAPH)*, pages 49–56, 1997.

[112] William Brandon Kerr and Fabio Pellacini. Toward evaluating lighting design interface paradigms for novice users. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 28(3):26:1–26:9, August 2009.

[113] William Brandon Kerr and Fabio Pellacini. Toward evaluating material design interface paradigms for novice users. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 29(4):35:1–35:10, July 2010.

[114] William Brandon Kerr, Fabio Pellacini, and Jonathan D. Denning. BendyLights: Artistic control of direct illumination by curving light rays. *Computer Graphics Forum (Proceedings of EG Symposium on Rendering)*, 29(4):1467–1459, 2010.

[115] Erum Arif Khan, Erik Reinhard, Roland W. Fleming, and Heinrich H. Bülthoff. Image-based material editing. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 25(3):654–663, July 2006.

[116] Alan King, Christopher Kulla, Alejandro Conty, and Marcos Fajardo. BSSRDF importance sampling. Proc. SIGGRAPH (Tech Talks), 2013.

[117] Oliver Klehm, Ivo Ihrke, Hans-Peter Seidel, and Elmar Eisemann. Volume stylizer: Tomography-based volume painting. In *Proc. SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 161–168, 2013.

[118] Michiel A. Klompenhouwer and Gerard de Haan. Subpixel image scaling for color matrix displays. *Journal of the Society for Information Display*, 11(1):99–108, 2003.

[119] Claude Knaus and Matthias Zwicker. Progressive photon mapping: A probabilistic approach. *ACM Transactions on Graphics*, 30:25:1–25:13, May 2011.

[120] Craig Kolb, Don P. Mitchell, and Patrick M. Hanrahan. A realistic camera model for computer graphics. In *Proc. SIGGRAPH*, pages 317–324, New York, NY, USA, 1995.

[121] Thomas Kollig and Alexander Keller. Illumination in the presence of weak singularities. *Monte Carlo and Quasi-Monte Carlo methods*, pages 245–257, 2004.

[122] Anders Wang Kristensen, Tomas Akenine-Möller, and Henrik Wann Jensen. Precomputed local radiance transfer for real-time lighting design. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 24(3):1208–1215, August 2005.

[123] Jaroslav Křivánek, Marcos Fajardo, Per H. Christensen, Eric Tabellion, Michael Bunnell, David Larsson, and Anton S. Kaplanyan. Global illumination across industries. In *Proc. SIGGRAPH (Courses)*, 2010.

[124] Jaroslav Křivánek, James A. Ferwerda, and Kavita Bala. Effects of global illumination approximations on material appearance. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 29(4):1–10, 2010.

[125] Jaroslav Křivánek, Iliyan Georgiev, Toshiya Hachisuka, Martin Šik, Petr Vévoda, Derek Nowrouzezahrai, and Wojciech Jarosz. Unifying points, beams, and paths in volumetric light transport simulation. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 33(4):1–13, 2014.

[126] Jaroslav Křivánek, Iliyan Georgiev, Anton S. Kaplanyan, and Juan Cañada. Recent advances in light transport simulation: Theory and practice. In *Proc. SIGGRAPH*, New York, NY, USA, 2013. ACM.

[127] Thomas Kroes, Frits H. Post, and Charl P. Botha. Exposure render: an interactive photo-realistic volume rendering framework. *PLoS ONE*, 7(7), July 2012.

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

[129] Eric P. Lafortune and Yves D. Willems. Rendering participating media with bidirectional path tracing. In *Proc. EG Workshop on Rendering*, pages 91–100, 1996.

[130] Samuli Laine, Hannu Saransaari, Janne Kontkanen, Jaakko Lehtinen, and Timo Aila. Incremental instant radiosity for real-time indirect illumination. In *Proc. EG Symposium on Rendering*, pages 277–286, 2007.

[131] Robert S. Laramee, Helwig Hauser, Helmut Doleisch, Benjamin Vrolijk, Frits H. Post, and Daniel Weiskopf. The state of the art in flow visualization: Dense and texture-based techniques. *Computer Graphics Forum*, 23, 2004.

[132] Chang Ha Lee, Xuejin Hao, and Anitabh Varshney. Geometry-dependent lighting. *IEEE Trans. on Visualization and Computer Graphics*, 12(2):197–207, 2006.

[133] Jaakko Lehtinen, Matthias Zwicker, Emmanuel Turquin, Janne Kontkanen, Frédo Durand, François X. Sillion, and Timo Aila. A meshless hierarchical representation for light transport. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 27(3):37:1–37:9, August 2008.

[134] Stuart P. Lloyd. Least square quantization in PCM. Bell Telephone Laboratories Paper, 1957.

[135] Bradford J. Loos, Lakulish Antani, Kenny Mitchell, Derek Nowrouzezahrai, Wojciech Jarosz, and Peter-Pike Sloan. Modular radiance transfer. *ACM Transactions on Graphics (Proc. of SIGGRAPH Asia)*, 30(6):178:1–178:10, December 2011.

[136] Ovidio Mallo, Ronald Peikert, Christian Sigg, and Filip Sadlo. Illuminated lines revisited. In *IEEE Visualization*, pages 19–26, 2005.

[137] Oliver Mattausch, Takeo Igarashi, and Michael Wimmer. Freeform shadow boundary editing. *Computer Graphics Forum (Proceedings Eurographics)*, 32(2):175–184, May 2013.

[138] Pavlos Mavridis and Georgios Papaioannou. High quality elliptical texture filtering on GPU. In *Proc. SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 23–30, 2011.

[139] Stephen McAuley, Stephen Hill, Adam Martinez, Ryusuke Villemin, Matt Pettineo, Dimitar Lazarov, David Neubelt, Brian Karis, Christophe Hery, Naty Hoffman, and Håkan Zap Andersson. Practical physically-based shading in film and game production. In *Proc. SIGGRAPH (Courses)*. ACM, 2012.

[140] Stephen McAuley, Stephen Hill, Adam Martinez, Ryusuke Villemin, Matt Pettineo, Dimitar Lazarov, David Neubelt, Brian Karis, Christophe Hery, Naty Hoffman, and Håkan Zap Andersson. Physically based shading in theory and practice. In *Proc. SIGGRAPH (Courses)*, pages 22:1–22:8, New York, NY, USA, 2013. ACM.

[141] Scott McCloud. *Understanding Comics: The Invisible Art*. William Morrow Paperbacks, 1993.

[142] Joel McCormack, Ronald Perry, Keith I. Farkas, and Norman P. Jouppi. Feline: fast elliptical lines for anisotropic texture mapping. In *Proc. SIGGRAPH*, pages 243–250, 1999.

[143] Johannes Meng, Florian Simon, Johannes Hanika, and Carsten Dachsbacher. Physically meaningful rendering using tristimulus colours. *Computer Graphics Forum (Proceedings of EG Symposium on Rendering)*, 34(4):31–40, 2015.

[144] Dean S. Messing and Scott Daly. Improved display resolution of subsampled colour images using subpixel addressing. In *International Conference on Image Processing*, volume 1, pages I–625–I–628, 2002.

[145] Dean S. Messing and Louis J. Kerofsky. Using optimal rendering to visually mask defective subpixels. In *Proceedings of SPIE,* volume 6057, pages 60570O–1–60570O–12, 2006.

[146] Dean S. Messing, Louis J. Kerofsky, and Scott Daly. Subpixel rendering on non-striped colour matrix displays. In *Image Processing (Proceedings of ICIP)*, volume 2, pages 949–952, 2003.

[147] Don P. Mitchell and Arun N. Netravali. Reconstruction filters in computer graphics. *Computer Graphics (Proc. SIGGRAPH)*, 22:221–228, 1988.

[148] A. Muñoz, J. I. Echevarria, F. J. Seron, J. Lopez-Moreno, M. Glencross, and Diego Gutierrez. BSSRDF estimation from single images. *Computer Graphics Forum (Proceedings Eurographics)*, 30(2):445–464, 2011.

[149] Kathy T. Mullen. The contrast sensitivity of human colour vision to red-green and blue-yellow chromatic gratings. *Journal of Physiology*, 359:381–400, 1985.

[150] Ren Ng, Ravi Ramamoorthi, and Patrick M. Hanrahan. All-frequency shadows using non-linear wavelet lighting approximation. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 22(3):376–381, July 2003.

[151] Ren Ng, Ravi Ramamoorthi, and Patrick M. Hanrahan. Triple product wavelet integrals for all-frequency relighting. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 23(3):477–487, August 2004.

[152] Duc Quang Nguyen, Ronald Fedkiw, and Henrik Wann Jensen. Physically based modeling and animation of fire. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 21(3):721–728, July 2002.

[153] Fred Edwin Nicodemus. *Self-Study Manual on Optical Radiation Measurements: Part I*. U.S. National Bureau of Standards, March 1976.

[154] Fred Edwin Nicodemus, J.C. Richmond, J.J. Hsia, I.W. Ginsberg, and T. Limperis. *Geometrical Considerations and Nomenclature for Reflectance*. U.S. National Bureau of Standards, October 1977.

[155] Jeffry S. Nimeroff, Eero Simoncelli, and Julie Dorsey. Efficient re-rendering of naturally illuminated environments. In *Proc. EG Workshop on Rendering*, pages 359–373, June 1994.

[156] Tomoyuki Nishita, Takao Sirai, Katsumi Tadamura, and Eihachiro Nakamae. Display of the earth taking into account atmospheric scattering. In *Proc. SIGGRAPH*, pages 175–182, New York, NY, USA, 1993. ACM.

[157] Jan Novák, Thomas Engelhardt, and Carsten Dachsbacher. Screen-space bias compensation for interactive high-quality global illumination with virtual point lights. In *Proc. SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 119–124, 2011.

[158] Jan Novák, Derek Nowrouzezahrai, Carsten Dachsbacher, and Wojciech Jarosz. Progressive virtual beam lights. *Computer Graphics Forum (Proceedings of EG Symposium on Rendering)*, 31(4), 2012.

[159] Jan Novák, Derek Nowrouzezahrai, Carsten Dachsbacher, and Wojciech Jarosz. Virtual ray lights for rendering scenes with participating media. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 31(4), July 2012.

[160] Derek Nowrouzezahrai, Jared Johnson, Andrew Selle, Dylan Lacewell, Michael Kaschalk, and Wojciech Jarosz. A programmable system for artistic volumetric lighting. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 30(4):29:1–29:8, 2011.

[161] NVIDIA ARC GmbH. NVIDIA Iray technical documentation. `http://nvidia-arc.com/iray/`, 2017. Last retrieved on 2017-05-11.

[162] Juraj Obert. *Real-time cinematic design of visual aspects in computer generated images*. PhD thesis, School of Electrical Engineering and Computer Science, University of Central Florida, 2010.

[163] Juraj Obert, Jaroslav Křivánek, Fabio Pellacini, Daniel Sýkora, and Sumanta Pattanaik. iCheat: A representation for artistic control of indirect cinematic lighting. *Computer Graphics Forum (Proceedings of EG Symposium on Rendering)*, 27(4):1217–1223, June 2008.

[164] Juraj Obert, Fabio Pellacini, and Sumanta Pattanaik. Visual editing for all-frequency shadow design. *Computer Graphics Forum (Proceedings of EG Symposium on Rendering)*, 29(4):1441–1449, 2010.

[165] Makoto Okabe, Yasuyuki Matsushita, Li Shen, and Takeo Igarashi. Illumination brush: Interactive design of all-frequency lighting. In *Proc. Pacific Graphics*, pages 171–180, 2007.

[166] Hisanari Otsu, Anton S. Kaplanyan, Johannes Hanika, Carsten Dachsbacher, and Toshiya Hachisuka. Fusing state spaces for Markov chain Monte Carlo rendering. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 35(4), July 2017.

[167] Hisanari Otsu, Yonghao Yue, Qiming Hou, Kei Iwasaki, Yoshinori Dobashi, and Tomoyuki Nishita. Replica exchange light transport on relaxed distributions. Proc. SIGGRAPH (Sketches), 2013.

[168] Jiawei Ou, Ondřej Karlík, Jaroslav Křivánek, and Fabio Pellacini. Evaluating progressive-rendering algorithms in appearance design tasks. *IEEE Computer Graphics and Applications*, 33(6):58–68, 2013.

[169] Mark Pauly, Thomas Kollig, and Alexander Keller. Metropolis light transport for participating media. In *Proc. EG Workshop on Rendering*, pages 11–22, 2000.

[170] Vincent Pegoraro. *Efficient Physically-Based Simulation of Light Transport in Participating Media*. PhD thesis, School of Computing, University of Utah, 2009.

[171] Vincent Pegoraro, Mathias Schott, and Steven G. Parker. An analytical approach to single scattering for anisotropic media and light distributions. In *Proc. of Graphics Interface*, pages 71–77, 2009.

[172] Fabio Pellacini. EnvyLight: An interface for editing natural illumination. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 29(4):34:1–34:8, August 2010.

[173] Fabio Pellacini, Frank Battaglia, R. Keith Morley, and Adam Finkelstein. Lighting with paint. *ACM Transactions on Graphics*, 26(2):9:1–9:14, June 2007.

[174] Fabio Pellacini and Jason Lawrence. AppWand: Editing measured materials using appearance-driven optimization. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 26(3):54:1–54:9, July 2007.

[175] Fabio Pellacini, Parag Tole, and Donald P. Greenberg. A user interface for interactive cinematic shadow design. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 21(3):563–566, July 2002.

[176] Fabio Pellacini, Kiril Vidimče, Aaron Lefohn, Alex Mohr, Mark Leone, and John Warren. Lpics: a hybrid hardware-accelerated relighting engine for computer cinematography. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 24(3):464–470, August 2005.

[177] Matt Pharr and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation, 2nd Edition*. Morgan Kaufmann Publishers Inc., 2010.

[178] Pixar Animation Studios. RenderMan 21 documentation. `https://rmanwiki.pixar.com/display/REN/RenderMan+Documentation`, 2017. Last retrieved on 2017-02-02.

[179] John C. Platt. Optimal filtering for patterned displays. *IEEE Signal Processing Letters*, 7(7):179–180, July 2000.

[180] John C. Platt, Bert Keely, Bill Hill, Bodin Dresevic, Claude Betrisey, Don P. Mitchell, Greg Hitchcock, James J. Blinn, and Turner Whitted. Displaced filtering for patterned displays. In *Proceedings Society for Information Display Symposium*, pages 296–299, May 2000.

[181] Pierre Poulin and Alain Fournier. Lights from highlights and shadows. In *Proc. SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 31–38, 1992.

[182] A.J. Preetham, Peter Shirley, and Brian Smits. A practical analytic model for daylight. In *Proc. SIGGRAPH*, pages 243–250, 1999.

[183] Simon Premože, Michael Ashikhmin, Ravi Ramamoorthi, and Shree K. Nayar. Practical rendering of multiple scattering effects in participating media. In *Proc. EG Symposium on Rendering*, 2004.

[184] William Henry Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes, 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, 2007.

[185] Claude Puech and François X. Sillion. *Radiosity and Global Illumination*. Morgan Kaufmann Publishers Inc., Natick, MA, USA, 1994.

[186] Matthias Raab, Daniel Seibert, and Alexander Keller. Unbiased global illumination with participating media. In *Monte Carlo and Quasi-Monte Carlo Methods 2006*, pages 591–606, 2008.

[187] Jonathan Ragan-Kelley, Charlie Kilpatrick, Brian W. Smith, Doug Epps, Paul Green, Christophe Hery, and Frédo Durand. The lightspeed automatic interactive lighting preview system. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 26(3), July 2007.

[188] Ravi Ramamoorthi. Precomputation-based rendering. *Foundations and Trends in Computer Graphics and Vision*, 3(4):281–369, April 2009.

[189] Tim Reiner, Anton S. Kaplanyan, Marcel Reinhard, and Carsten Dachsbacher. Selective inspection and interactive visualization of light transport in virtual scenes. *Computer Graphics Forum (Proceedings Eurographics)*, 31(2):711–718, 2012.

[190] Erik Reinhard, Wolfgang Heidrich, Paul Debevec, Sumanta Pattanaik, Greg Ward, and Karol Myszkowski. *High Dynamic Range Imaging, 2nd Edition - Acquisition, Display, and Image-Based Lighting*. Morgan Kaufmann Publishers Inc., 2010.

[191] Erik Reinhard, Erum Arif Khan, Ahmet Oguz Akyüz, and Garrett M. Johnson. *Color Imaging: Fundamentals and Applications*. A. K. Peters, Ltd., 2008.

[192] Tobias Ritschel, Thorsten Grosch, Min H. Kim, Hans-Peter Seidel, Carsten Dachsbacher, and Jan Kautz. Imperfect shadow maps for efficient computation of indirect illumination. In *ACM Transactions on Graphics (Proc. of SIGGRAPH Asia)*, pages 129:1–129:8, 2008.

[193] Tobias Ritschel, Makoto Okabe, Thorsten Thormählen, and Hans-Peter Seidel. Interactive reflection editing. *ACM Transactions on Graphics (Proc. of SIGGRAPH Asia)*, 28(5):129:1–129:7, 2009.

[194] Tobias Ritschel, Thorsten Thormählen, Carsten Dachsbacher, Jan Kautz, and Hans-Peter Seidel. Interactive on-surface signal deformation. *ACM Transactions on Graphics*, 29(4):36:1–36:8, 2010.

[195] Holly E. Rushmeier and Kenneth E. Torrance. The zonal method for calculating light intensities in the presence of a participating medium. *Computer Graphics (Proc. SIGGRAPH)*, 21(4):293–302, 1987.

[196] Iman Sadeghi, Heather Pritchett, Henrik Wann Jensen, and Rasmus Tamstorf. An artist-friendly hair shading system. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 29(4):56:1–56:10, July 2010.

[197] Takafumi Saito and Tokiichiro Takahashi. Comprehensible rendering of 3-D shapes. In *Proc. SIGGRAPH*, pages 197–206, August 1990.

[198] Marco Salvi, Kiril Vidimče, Andrew Lauritzen, and Aaron Lefohn. Adaptive volumetric shadow maps. In *Proc. EG Symposium on Rendering*, pages 1289–1296, June 2010.

[199] Andreas Schilling, Günter Knittel, and Wolfgang Strasser. Texram: A smart memory for texturing. *IEEE Computer Graphics and Applications*, 16:32–41, 1996.

[200] Thorsten-Walther Schmidt, Jan Novák, Johannes Meng, Anton S. Kaplanyan, Tim Reiner, Derek Nowrouzezahrai, and Carsten Dachsbacher. Path-space manipulation of physically-based light transport. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 32(4):129:1–129:11, August 2013.

[201] Thorsten-Walther Schmidt, Fabio Pellacini, Derek Nowrouzezahrai, Wojciech Jarosz, and Carsten Dachsbacher. State of the art in artistic editing of appearance, lighting, and material. In *Eurographics 2014 - State of the Art Reports*. Eurographics Association, April 2014.

[202] Thorsten-Walther Schmidt, Fabio Pellacini, Derek Nowrouzezahrai, Wojciech Jarosz, and Carsten Dachsbacher. State of the art in artistic editing of appearance, lighting, and material. *Computer Graphics Forum*, 35(1):216–233, February 2016.

[203] Chris Schoeneman, Julie Dorsey, Brian Smits, James Arvo, and Donald P. Greenberg. Painting with light. In *Proceedings of SIGGRAPH 93*, ACM Transactions on Graphics, pages 143–146, August 1993.

[204] Emanuel Schrade, Johannes Hanika, and Carsten Dachsbacher. Sparse high-degree polynomials for wide-angle lenses. *Computer Graphics Forum (Proceedings of EG Symposium on Rendering)*, 35(4):89–97, 2016.

[205] Robert Sedgewick and Kevin Wayne. *Algorithms, 4th Edition*. Addison-Wesley Professional, Boston, MA, USA, 1993.

[206] Carlo H. Séquin and Eliot K. Smyrl. Parameterized ray tracing. In *Computer Graphics (Proc. SIGGRAPH)*, pages 307–314, July 1989.

[207] Hyun-Chul Shin, Jin-Aeon Lee, and Lee-Sup Kim. SPAF: Sub-texel precision anisotropic filtering. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pages 99–108, 2001.

[208] S. Silva, B Sousa Santos, and J. Madeira. Using color in visualization: A survey. *Computer and Graphics*, 35(2):320–333, 2011.

[209] Florian Simon, Johannes Hanika, and Carsten Dachsbacher. Rich-VPLs for improving the versatility of many-light methods. *Computer Graphics Forum (Proceedings Eurographics)*, 34(2):575–584, May 2015.

[210] Peter-Pike Sloan, Jan Kautz, and John Snyder. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 21(3):527–536, July 2002.

[211] Solid Angle Ltd. Arnold. `https://www.solidangle.com/arnold/`, 2017. Last retrieved on 2017-05-17.

[212] Solid Angle Ltd. Arnoldpedia for Arnold 5 - light path expression AOVs. `https://support.solidangle.com/display/A5ARP/Light+Path+Expression+AOVs`, 2017. Last retrieved on 2017-05-11.

[213] Ying Song, Xin Tong, Fabio Pellacini, and Pieter Peers. SubEdit: A representation for editing measured heterogenous subsurface scattering. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 28(3):31:1–31:10, August 2009.

[214] Sony Pictures Imageworks. Open shading language documentation - OSL light path expressions. `https://github.com/imageworks/OpenShadingLanguage/wiki/OSL-Light-Path-Expressions`, 2017. Last retrieved on 2017-05-11.

[215] Nicolai Spassov and Todor Stoytchev. The presence of cave hyaena (crocuta crocuta spelaea) in the upper palaeolithic rock art of Europe. *Historia naturalis bulgarica*, 16:159–166, 2004.

[216] Jos Stam. Multiple scattering as a diffusion process. In *Proc. EG Workshop on Rendering*, pages 41–50, 1995.

[217] B. Steinert, Holger Dammertz, Johannes Hanika, and Hendrik P.A. Lensch. General spectral camera lens simulation. *Computer Graphics Forum*, 30(6):1643–1654, 2011.

[218] Thomas Subileau, Nicolas Mellado, David Vanderhaege, and Matthias Paulin. Light transport editing with ray portals. In *Proc. Computer Graphics International*, 2015.

[219] Kevin Suffern. *Ray Tracing from the Ground Up*. A. K. Peters, Ltd, 2007.

[220] Bo Sun, Ravi Ramamoorthi, Srinivasa G. Narasimhan, and Shree K. Nayar. A practical analytic single scattering model for real-time rendering. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 24(3):1040–1049, July 2005.

[221] Weifeng Sun and Amar Mukherjee. Generalized wavelet product integral for rendering dynamic glossy objects. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 25(3):955–966, July 2006.

[222] Xin Sun, Kun Zhou, Yanyun Chen, Stephen Lin, Jiaoying Shi, and Baining Guo. Interactive relighting with dynamic BRDFs. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 26(3):27:1–27:10, July 2007.

[223] Xin Sun, Kun Zhou, Stephen Lin, and Baining Guo. Line space gathering for single scattering in large scenes. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 29(4):54:1–54:8, July 2010.

[224] Ivan Edward Sutherland. *Sketchpad: A Man-Machine Graphical Communication System*. PhD thesis, Massachusetts Institute of Technology, 1963.

[225] Eric Tabellion and Arnauld Lamorlette. An approximate global illumination system for computer-generated films. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 23(3):469–476, 2004.

[226] Marco Tarini, Paolo Cignoni, and Claudio Montani. Ambient occlusion and edge cueing to enhance real-time molecular visualization. *IEEE Trans. on Visualization and Computer Graphics*, 12(5):1237–1244, 2006.

[227] Krzysztof Templin, Piotr Didyk, Tobias Ritschel, Elmar Eisemann, Karol Myszkowski, and Hans-Peter Seidel. Apparent resolution enhancement for animations. In *Proceedings of the 27th Spring Conference on Computer Graphics*, pages 85–92, 2011.

[228] Peter Vangorp, Jurgen Laurijssen, and Philip Dutré. The influence of shape on the perception of material reflectance. *ACM Transactions on Graphics*, 26(3), 2007.

[229] Eric Veach. *Robust Monte Carlo Methods for Light Transport Simulation*. PhD thesis, Stanford University, 1997.

[230] Eric Veach and Leonidas J. Guibas. Bidirectional estimators for light transport. In *Proc. EG Workshop on Rendering*, pages 147–162, 1994.

[231] Eric Veach and Leonidas J. Guibas. Metropolis light transport. In *Proc. SIGGRAPH*, pages 65–76, 1997.

[232] Ryusuke Villemin and Christophe Hery. Practical illumination from flames. *Journal of Computer Graphics Techniques*, 2(2):142–155, 2013.

[233] Jeffret Scott Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software*, 11(1):37–57, March 1985.

[234] Bruce Walter, Adam Arbree, Kavita Bala, and Donald P. Greenberg. Multidimensional lightcuts. *ACM Transactions on Graphics*, 25(3):1081–1088, July 2006.

[235] Bruce Walter, Sebastian Fernandez, Adam Arbree, Kavita Bala, Michael Donikian, and Donald P. Greenberg. Lightcuts: a scalable approach to illumination. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 24(3):1098–1107, 2005.

[236] Bruce Walter, Pramook Khungurn, and Kavita Bala. Bidirectional lightcuts. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 31(4):59:1–59:11, 2012.

[237] Bruce Walter, Stephen R. Marschner, Hongsong Li, and Kenneth E. Torrance. Microfacet models for refraction through rough surfaces. In *Proc. EG Symposium on Rendering*, pages 195–206, 2007.

[238] Jiaping Wang, Peiran Ren, Minmin Gong, John Snyder, and Baining Guo. All-frequency rendering of dynamic, spatially-varying reflectance. *ACM Transactions on Graphics (Proc. of SIGGRAPH Asia)*, 28(5):133:1–133:10, December 2009.

[239] Rui Wang, John Tran, and David Luebke. All-frequency relighting of non-diffuse objects using separable BRDF approximation. In *Proc. EG Workshop on Rendering*, pages 345–354, June 2004.

[240] Christoph Wetzel, editor. *Belser Stilgeschichte*. Belser-Verlag, Stuttgart, 1999.

[241] Alexander Wilkie and Lukas Hošek. Predicting sky dome appearance on earth-like extrasolar worlds. In *Proc. 29th Spring conference on Computer Graphics (SCCG 2013)*, May 2013.

[242] Lance Williams. Casting curved shadows on curved surfaces. *Computer Graphics (Proc. SIGGRAPH)*, 12(3):270–274, 1978.

[243] Ed R. Woodcock, T. Murphy, P. J. Hemmings, and Longworth T. C. Techniques used in the GEM code for Monte Carlo neutronics calculations in reactors and other systems of complex geometry. In *Applications of Computing Methods to Reactor Problems*, pages 557–579. Argonne National Laboratory, 1965.

[244] Hongzhi Wu, Julie Dorsey, and Holly E. Rushmeier. Physically-based interactive bi-scale material design. *ACM Transactions on Graphics (Proc. of SIGGRAPH Asia)*, 30(6):145:1–145:10, December 2011.

[245] Hongzhi Wu, Julie Dorsey, and Holly E. Rushmeier. Inverse bi-scale material design. *ACM Transactions on Graphics (Proc. of SIGGRAPH Asia)*, 32(6):163:1–163:10, 2013.

[246] Jiajing Xu, Joyce Farrell, Tanya Matskewich, and Brian Wandell. Prediction of preferred ClearType filters using the S-CIELAB metric. In *Image Processing (Proc. of ICIP)*, pages 361–364, 2008.

[247] Shuang Zhao, Miloš Hašan, Ravi Ramamoorthi, and Kavita Bala. Modular flux transfer: efficient rendering of high-resolution volumes with repeated structures. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 32(4):131:1–131:12, July 2013.

[248] Tobias Zirr, Marco Ament, and Carsten Dachsbacher. Visualization of coherent structures of light transport. *Computer Graphics Forum (Proceedings of EuroVis)*, 34(3):491–500, May 2015.

Index