Efficient Illumination Algorithms for Global Illumination In Interactive and Real-Time Rendering

A DISSERTATION
SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY AT THE
DEPARTMENT OF INFORMATICS OF THE
ATHENS UNIVERSITY OF ECONOMICS & BUSINESS

Konstantinos Vardis
December 2016
Department of Informatics
Athens University of Economics & Business
Greece

Abstract

The synthesis of photorealistic digital imagery has long been considered as one of the most fascinating domains in the field of computer graphics. Its main goal is the generation of visually stunning images, mimicking as close as possible the appearance of objects in the physical world. The endeavor for visual realism has directed a large amount of research interest in the investigation of the interactions of light and matter, resulting in an established mathematical framework and the striking beauty of the generated images on today's production level renderers. While the theoretical concepts of light transport are well understood and applied in offline rendering, the interactive reproduction of the underlying physical processes remains a highly challenging topic due to the various constraints involved in the process. Furthermore, the increased need for the delivery of highly dynamic interactive content in today's vast virtual environments, that can potentially change in every frame, has undoubtedly increased the necessity for highly efficient, interactive illumination algorithms.

In this thesis, we investigate such methods, in the field of photorealistic image synthesis. Our contributions focus exclusively on the development of robust algorithms for real-time and interactive global illumination, under the considerations of fully dynamic content. Regarding real-time rendering, the majority of algorithms are based on the rasterization pipeline, where the support of dynamic content is inherently provided. However, the strict time restrictions of real-time applications pose significant constraints in the operation of the computationally demanding global illumination algorithms, severely impacting the resulting quality of the rendered images. There, the most common approximation is imposed on the algorithmic input where, typically, the highly-detailed geometric information is replaced by either (i) a partial (layered), view-dependent and discretized representation, or a (ii) view-independent but, crude, regular subdivision of the environment in image- and volume-space methods, respectively. We contribute to the domain of real-time rendering by proposing two novel techniques, focusing particularly on the improvement of the visual instability of prior approaches as well as on the efficiency of the underlying representations. First, we propose a generic method to efficiently address the view-dependent inconsistencies of image-domain methods, demonstrated on screen-space ambient occlusion. This is accomplished by taking advantage of buffers containing geometric information from other view points, already generated as part of the rendering process, such as the shadow maps. Second, we improve the efficiency as well as on the visual stability of volume-based global illumination methods, by introducing the idea of directional chrominance subsampling for radiance field compression, an optimized cache point population scheme and a view-independent approximate indirect shadowing technique.

In order to support dynamic content in interactive applications, the effort of the research community has been heavily focused on the improvement of the efficiency of the ray tracing pipeline, which has been traditionally employed for production rendering. However, the computational overhead of the required complex acceleration structures is still restricting these approaches to partially static content. Alternatively, a recent category of techniques have attempted to exploit the rasterization pipeline, which inherently supports dynamic environments, to achieve quality identical to the ray tracing pipeline. Still, the proposed solutions are not yet able to support a full global illumination algorithm without posing any restrictions on the geometric representation or on the effects that can be reproduced. In the domain of interactive rendering, we present two methods that investigate the ability of the modern rasterization pipeline to provide a viable alternative to the costly construction stages of spatial acceleration structures. The proposed methods are able to perform high-quality interactive ray tracing in arbitrarily complex and dynamic environments, thus lifting the limitations of prior rasterization-based methods. Our first method

employs multifragment rendering techniques to effectively capture, for the first time, a highly-detailed representation of the entire environment where a full global illumination algorithm, such as path tracing, can be elegantly supported. Ray tracing is efficiently achieved by exploiting image-space empty space skipping and approximate ray-fragment intersection tests. The presented solution advances the field of image-space ray tracing and provides small construction times as well as scalable traversal performance. However, the resulting quality is approximate and can suffer from high memory overhead due to its fragment-based data structure. Our second approach, which completes our investigation, applies the deferred nature of the traditional ray-tracing pipeline in a rasterization-based framework. Thus, we are able to exploit a primitive-based acceleration data structure and support three, conflicting in prior work, objectives: (i) dynamic environments through fast construction times, (ii) quality identical to the ray tracing pipeline via primitive-based intersection tests, and (iii) reduced memory requirements. Additionally, the presented method further generalizes on the field of image-space ray tracing by exploiting various empty space skipping optimization strategies in order to efficiently support accurate ray-primitive intersection queries.

Περίληψη στα Ελληνικά

Η ψηφιακή σύνθεση φωτορεαλιστικών εικόνων θεωρείται ένα από τα πιο συναρπαστικά πεδία στον τομέα των γραφικών του υπολογιστή. Κύριος στόχος της είναι η παραγωγή εικόνων που προσομοιώνουν όσο το δυνατόν πιο πειστικά τα μοντελοποιημένα αντικείμενα, όπως αυτά απεικονίζονται στον πραγματικό κόσμο. Η προσπάθεια βελτίωσης του οπτικού ρεαλισμού έχει κατευθύνει ένα μεγάλο κομμάτι της έρευνας στην διερεύνηση των αλληλεπιδράσεων του φωτός και της ύλης, με αποτέλεσμα την θεμελίωση ενός εμπεριστατωμένου μαθηματικού πλαισίου καθώς και την εντυπωσιακή ποιότητα των παραγόμενων εικόνων από τις σημερινές εφαρμογές σύνθεσης εικόνας σε γραφικά κινηματογραφικής ποιότητας. Ενώ οι θεωρητικές έννοιες της μετάδοσης του φωτός έχουν κατανοηθεί σε μεγάλο βαθμό και εφαρμόζονται στα γραφικά μη πραγματικού χρόνου, η σωστή αναπαραγωγή των διαφόρων φυσικών φαινομένων σε διαδραστικά και πραγματικού χρόνου περιβάλλοντα είναι ένα εξαιρετικά δύσκολο πρόβλημα, λόγω των χρονικών περιορισμών που επιβάλλονται στις συγκεκριμένες διεργασίες. Η ανάγκη για αλγορίθμους ικανούς να αποδώσουν φωτορεαλιστικές εικόνες σε δυναμικά περιβάλλοντα είναι ακόμα πιο απαραίτητη στην σημερινή εποχή όπου το μέγεθος καθώς και η πολυπλοκότητα των ψηφιακών κόσμων, οι οποίοι μπορούν να μεταβληθούν δραστικά σε κάθε καρέ, έχει αυξηθεί δραματικά.

Ο σκοπός της παρούσας διατριβής είναι να διερευνήσει τέτοιες μεθόδους που εμπίπτουν στον τομέα της φωτορεαλιστικής σύνθεσης εικόνας. Οι συνεισφορές μας επικεντρώνονται αποκλειστικά στην ανάπτυξη αποτελεσματικών αλγορίθμων ολικού φωτισμού σε γραφικά πραγματικού χρόνου καθώς και διαδραστικά, λαμβάνοντας υπόψη τους περιορισμούς που προκύπτουν σε δυναμικά περιβάλλοντα. Όσον αφορά την πρώτη κατηγορία (γραφικά πραγματικού χρόνου), η πλειονότητα των αλγορίθμων βασίζεται στη μέθοδο της σχεδίασης (rasterization), όπου η υποστήριξη δυναμικού περιεχομένου παρέχεται εγγενώς. Όμως, τα αυστηρά χρονικά όρια των εφαρμογών αυτών θέτουν και σημαντικούς περιορισμούς στην ακρίβεια και λειτουργία των υπολογιστικά απαιτητικών αλγορίθμων ολικού φωτισμού, επηρεάζοντας σοβαρά την ποιότητα των τελικών εικόνων. Οι πιο σημαντικές απλοποιήσεις επιβάλλονται από πολύ νωρίς· συνήθως, η γεωμετρική πληροφορία της σκηνής αντικαθίσταται είτε από (i) μία μερική και εξαρτώμενη από τη θέση του παρατηρητή, διακριτοποιημένη αναπαράσταση της σκηνής, ή από (ii) μία ολική μεν, αλλά αδρή, κανονική υποδιαίρεση της σκηνής, στις τεχνικές που δουλεύουν στον χώρο της εικόνας και ογκομετρικές μεθόδους, αντίστοιχα. Συμβάλλουμε στον τομέα αυτό προτείνοντας δύο νέες τεχνικές, οι οποίες επικεντρώνονται στη βελτίωση της οπτικής αστάθειας που προκύπτει από τις προσεγγίσεις που γίνονται στους αλγορίθμους αυτούς, καθώς και στην αποτελεσματικότητα των δομών που χρησιμοποιούνται. Η πρώτη μας μέθοδος, η οποία εφαρμόζεται στις κλασσικές μεθόδους εικόνας για παρεμπόδιση του έμμεσου φωτισμού, επικεντρώνεται στην γενική αντιμετώπιση του προβλήματος των οπτικών ατελειών που προκύπτουν από την εξάρτηση στην οπτική γωνία του παρατηρητή. Αυτό επιτυγχάνεται με την αξιοποίηση γεωμετρικής πληροφορίας από άλλες οπτικές γωνίες, χωρίς να υπάρχει περιορισμός στον αριθμό ή την τοποθεσία αυτών που είναι ήδη διαθέσιμες σε μία μηχανή γραφικών, όπως είναι οι εικόνες για την παραγωγή σκιών. Η δεύτερη μας μέθοδος ασχολείται με την βελτίωση της αποτελεσματικότητας της δομής καθώς και την οπτική σταθερότητα των ογκομετρικών μεθόδων ολικού φωτισμού, εισάγοντας την ιδέα της υποδειγματοληψίας της χρωματικότητας για την κατευθυντική συμπίεση της λαμπρότητας στο σφαιρικό πεδίο, ένα βελτιστοποιημένο σύστημα τοποθέτησης σημείων λαμπρότητας καθώς και μια προσεγγιστική τεχνική απόδοσης έμμεσων σκιάσεων ανεξαρτήτων από την εκάστοτε όψη.

Στην κατηγορία των διαδραστικών γραφικών, η υποστήριξη δυναμικού περιεχόμενου από την ερευνητική κοινότητα έχει επικεντρωθεί σε μεγάλο βαθμό στη βελτίωση της αποτελεσματικότητας αλγορίθμων βασιζόμενοι στην παρακολούθηση ακτινών (ray tracing), το οποίο χρησιμοποιείται σχεδόν αποκλειστικά σε γραφικά μη πραγματικού χρόνου. Ωστόσο, το υπολογιστικό κόστος κατασκευής των απαιτούμενων επιταχυντικών δομών εξακολουθεί να περιορίζει αυτές τις μεθόδους σε εν μέρει στατικό περιεχόμενο. Εναλλακτικά, πρόσφατες ερευνητικές προσπάθειες έχουν προσπαθήσει να εκμεταλλευτούν τη μέθοδο της σχεδίασης γραφικών, το οποίο υποστηρίζει εγγενώς δυναμικά περιβάλλοντα, για να επιτύχουν ποιότητα ταυτόσημη με την παρακολούθηση ακτινών. Παρόλα αυτά, οι υπάρχουσες λύσεις δεν είναι ακόμη σε θέση να υποστηρίξουν πλήρως έναν αλγόριθμο ολικού φωτισμού χωρίς να θέτουν περιορισμούς όσον αφορά είτε τη γεωμετρική αναπαράσταση ή τις πολλαπλές αλληλεπιδράσεις μεταξύ του φωτός και των διαφόρων οπτικών μέσων. Στην κατηγορία των διαδραστικών γραφικών παρουσιάζουμε δύο μεθόδους που διερευνούν τη δυνατότητα του σύγχρονου μοντέλου σχεδίασης γραφικών να παρέχει μια εναλλακτική υπολογιστικά λύση στα ακριβά στάδια κατασκευής των επιταχυντικών δομών. Οι προτεινόμενες μέθοδοι μπορούν να αποδώσουν υψηλής ποιότητας διαδραστική παρακολούθηση ακτινών σε αυθαίρετης πολυπλοκότητας δυναμικά περιβάλλοντα, αφαιρώντας έτσι τους περιορισμούς των προηγούμενων μεθόδων που βασίζονται στη σχεδίαση γραφικών. Η πρώτη μας μέθοδος αξιοποιεί τεχνικές πολυεπιπέδων για να αποθηκεύσει αποτελεσματικά και για πρώτη φορά, μία λεπτομερή αναπαράσταση όλου του περιβάλλοντος στην οποία ένας αλγόριθμος ολικού φωτισμού - όπως η παρακολούθηση μονοπατιού (path tracing) - μπορεί να υποστηριχθεί. Η παρακολούθηση ακτινών επιτυγχάνεται αποτελεσματικά αξιοποιώντας γρήγορη μεταπήδηση χώρων στο χώρο της εικόνας και προσεγγιστικούς ελέγχους τομών μεταξύ των ακτινών και των παραγόμενων τεμαχίων (fragments). Η προτεινόμενη λύση βελτιώνει δραματικά την ποιότητα αλλά και την ταχύτητα αλγορίθμων που βασίζονται στην παρακολούθηση ακτινών στο χώρο της εικόνας ενώ παράλληλα παρέχει μικρούς χρόνους κατασκευής, καθώς και κλιμακωτή απόδοση διάσχισης ακτινών. Ωστόσο, η συγκεκριμένη μέθοδος προσφέρει προσεγγιστική ποιότητα εικόνας, συγκριτικά με τις παραδοσιακές μεθόδους παρακολούθησης ακτινών, και μπορεί να έχει υψηλό κόστος μνήμης λόγω της αποθήκευσης δεδομένων για την φωτοσκίαση των τεμαχίων για όλη την σκηνή. Η δεύτερη μέθοδός μας, η οποία ολοκληρώνει την έρευνά μας, μεταφέρει την αναβαλλόμενη λογική της αποθήκευσης γεωμετρικής πληροφορίας από τους αλγορίθμους παρακολούθησης ακτινών στο μοντέλο σχεδίασης γραφικών. Με αυτόν τον τρόπο είμαστε σε θέση να δημιουργήσουμε μία δομή που βασίζεται σε γεωμετρία αντί για τεμάχια και να υποστηρίξουμε τρεις, αντικρουόμενους σε προηγούμενες ερευνητικές δουλειές, στόχους: (i) δυναμικά περιβάλλοντα με γρήγορους χρόνους κατασκευής, (ii) ποιότητα ταυτόσημη με τους αλγόριθμους παρακολούθησης ακτινών με ελέγχους τομών βασισμένους σε πολύγωνα και (iii) μειωμένες απαιτήσεις μνήμης. Επιπλέον, η προτεινόμενη μέθοδος βελτιώνει και γενικεύει περαιτέρω τον τομέα της παρακολούθησης ακτινών στο χώρο της εικόνας με την αξιοποίηση διαφόρων βελτιστοποιήσεων μεταπηδήσεων χώρου ώστε να υποστηρίξει αποτελεσματικά αναλυτικούς ελέγχους τομών μεταξύ ακτινών και πολυγώνων.

The work presented in this thesis has received funding from the European Union's Seventh Framework Programme under grant agreement no 600533 (EC FP7 STREP Project PRESIOUS) and has also been co-financed by the European Union (European Social Fund - ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: ARISTEIA II-GLIDE under grant agreement no 3712.

Co-financed by Greece and the European Union

Three-Member Supervisory Committee:

- Georgios Papaioannou, Assistant Professor at the Department of Informatics of the Athens University of Economics and Business (supervisor)
- George Xylomenos, Associate Professor at the Department of Informatics of the Athens University of Economics and Business
- Nikolaos Platis, Lecturer at the Department of Informatics and Telecommunications of the University of the Peloponnese

Seven-Member Examination Committee:

- Georgios Papaioannou, Assistant Professor at the Department of Informatics of the Athens University of Economics and Business (supervisor)
- Stavros Toumpis[1], Assistant Professor at the Department of Informatics of the Athens University of Economics and Business
- Nikolaos Platis, Lecturer at the Department of Informatics and Telecommunications of the University of the Peloponnese
- Theoharis Theoharis, Professor at the Department of Informatics and Telecommunications of the University of Athens
- Ioannis Fudos, Associate Professor at the Department of Computer Science & Engineering of the University of Ioannina
- Katerina Mania, Associate Professor at the School of Electrical and Computer Engineering of the Technical University of Crete
- Konstantinos Moustakas, Assistant Professor at the Department of Electrical and Computer Engineering of the University of Patras

[1] Prof. George Xylomenos was substituted by Prof. Stavros Toumpis in the Seven-Member Examination Committee

Acknowledgments

The work presented in this thesis could not be accomplished without the help and support from many people. First and foremost, I would like to thank my advisor, Georgios Papaioannou, for his guidance, knowledge as well as his inexhaustible thirst toward researching and exploring new ideas and directions. His support has simply been invaluable in all the years of my research.

I owe my deepest gratitude to my parents, Katerina and Andreas, for their unconditional support and for their continuous encouragement to follow my passion for computer science since I was very young. Thank you both, for everything.

Next, I am grateful to my former colleague Andreas Vasilakis, whose enthusiasm, support and confidence in my work have been essential to the completion of this thesis. I will never forget our long, and often late night, hours we spent collaborating, brain storming, coding and arguing.

I would also like to thank my former colleagues, Anthousis Andreadis and Pavlos Mavridis, who provided a friendly environment to work in as well as for our insightful and valuable conversations.

Finally, I would like to thank all the members of my committee, for their insightful comments and suggestions when reviewing this dissertation. I would to all the anonymous reviewers for their feedback on the work presented in this dissertation.

Dedicated to Dad

Dedicated to Mom

Dedicated to Charles

Contents

Chapter 1

Introduction

Over the last few years, the tremendous increase in graphics processing power in conjunction with the capabilities of parallel computing in rendering applications has increased the need for methods delivering highly realistic, interactive content.

In this thesis, we investigate such methods, in the field of photorealistic image synthesis. Our contributions focus exclusively on the development of robust algorithms for global illumination (GI), directly applicable to interactive and real-time rendering of dynamic environments.

This chapter is organized as follows: Section 1.1 provides a generic introduction to rendering and illumination. Section 1.2 discusses the open problems that were targeted during this research and Section 1.3 summarizes our contributions. Finally, Section 1.4 outlines and explains the structure of this thesis.

1.1 Rendering

In this section, we begin with an introduction to the areas of rendering and realistic image synthesis. We provide an explanation to these generic concepts in order to familiarize readers with the general terminology used throughout this thesis.

1.1.1 Classification of Rendering Algorithms

The generation of synthetic images based on a collection of data, e.g., the representation of a virtual environment, is accomplished through the process of *rendering*. This vast field of computer graphics has numerous applications in multiple areas such as architectural design, games, the films and TV industry and medical and scientific visualization, just to name a few. Consequently, a large amount of research effort has been focused on the development of generic, accurate and robust rendering algorithms throughout the years.

One way to classify rendering methods is according to the following broad criteria: (i) the realistic accuracy of the generated images and (ii) interactivity.

According to the first criterion, the distinction falls between *photorealistic* and *non-photorealistic* rendering. The main goal of photorealistic rendering is the generation of images indistinguishable from real photographs, by simulating the phenomena of the physical world as closely as possible. Conversely, in non-photorealistic rendering, the necessity to obey the physical laws is more relaxed. These algorithms, therefore, depend partially or wholly on stylized rendering. Typical uses include toon rendering and technical illustrations. Other non-photorealistic approaches emphasize the interpretation and intuitive association of data; i.e., conveying information in a more contextual and understandable way to the human brain. A common example of the latter is scientific visualization.

Regarding the second criterion, algorithms are categorized as *offline*, *interactive* or *real-time*, listed here in ascending order with respect to interactivity requirements. Offline algorithms, the prevailing approach for production rendering, have zero interactivity requirements as the final content is delivered to

the target audience as a finalized, fixed creation, e.g., in the form of a digital movie sequence. Therefore, their major priority is the generation of images with the highest possible degree of visual quality. To accomplish this, computer clusters (render farms) may work in parallel for minutes, hours or even longer in order to produce a single frame.

On the other side of the spectrum, real-time algorithms focus on interactivity, immersion and fully dynamic content, i.e., environments that can change drastically in every frame. Being directly applicable to games and executed on commodity hardware, the rendered image is generally computed in fractions of a second, e.g., 16ms or 60 frames per second (fps). This way, the latency between user-triggered actions and the final result is minimized. However, the need to deliver a smooth interactive experience on fully dynamic environments poses severe restrictions both on the data size and quality of the input as well as on the algorithmic complexity of the operations that can be performed.

Algorithms operating at interactive rates but with no strict timing requirements, lie in the middle and balance the constraints posed by interactivity and quality. Hence, they are aimed at applications where the final result depends on both. Architectural and lighting design are prominent examples of interactive rendering. There, physically correct visualization of the environment is not the only design goal; the various parameters that constitute the virtual environment, such as the camera, geometry or light sources can be manipulated, deformed, changed or removed at any time. In such applications, image updates of up to one minute may be tolerable.

At this point, we should clarify the use of the term “interactive” as it is used throughout this thesis. First, while an algorithm capable of handling dynamic content is inherently considered interactive, the opposite does not always hold true in the literature; an interactive algorithm does not necessarily imply its capability to support dynamic environments. In several methods, for example, interactive walkthroughs are possible as long as the environment remains static. If anything changes, however, such as the locations and physical properties of the objects and/or the light sources, a (non-interactive) computation stage must be performed again. When we refer to an interactive algorithm throughout this thesis, we indirectly assume dynamic content and, therefore, the time constraints relate to the entire process. Second, it can be reasonable to assume that an algorithm operating at real-time rates is an interactive algorithm as well. Here, they are treated separately; an interactive method lies between offline and real-time rendering and is distinguished from real-time image generation due to its more relaxed time constraints.

1.1.2 Global Illumination

An *illumination algorithm* receives as input a description of a virtual environment, containing information on the location of objects and their material properties, a set of light sources and a specification of a sensor; a camera representing an eye in the virtual world. The expected outcome is then an image, depicting the scene as if the user was viewing it in the real world from that particular camera location. The design of such efficient and robust algorithms forms the basis of realistic image synthesis.

Photorealistic rendering is, ideally, based on the theory of light transport. However, an accurate simulation of the quantum nature of light is practically infeasible and simplifications are deemed as necessary. In computer graphics, the macroscopic model of *geometric optics* is usually employed which, in most scenarios, is a sufficient approximation of energy transport in the visible range of the electromagnetic spectrum. In this model, the measured quantity (radiance) is assumed to propagate instantaneously (in vacuum), following infinitesimally thin straight paths (rays) in homogeneous media, diverging only on material boundaries (e.g., a change of the index of refraction). The *rendering equation*, proposed by Kajiya in 1986 [Kaj86], is the prevailing mathematical model used to date and has been extensively used in graphics research to describe the interaction of light and matter in a synthetic environment.

A full *global illumination* algorithm attempts to solve the rendering equation in its entirety, i.e., at every location within the environment. However, this involves the computation of high-dimensional integrals of discontinuous functions and is considered an intractable problem (a thorough discussion is provided in Chapter 2). As such, a large part of research in the graphics community has been devoted on the development of robust illumination algorithms by simplifying and approximating the rendering equation.

For example, the radiosity algorithm works on a discretized version of the scene and assumes all surfaces are pure diffuse reflectors (see Section 2.4.1). The simulation of caustics, which appear due to multiple reflective and refractive rays focusing on a diffuse surface, are best approached with photon mapping (see Section 2.4.3). Both of these methods introduce statistical bias. Approaches that attempt to solve the rendering equation without introducing any systematic error also exist, such as path tracing, which works by recursively tracing rays throughout the environment. Nevertheless, their probabilistic nature makes them rather inefficient with infinitesimally small light sources and highly specular surfaces (see Section 2.4.2).

1.2 Problem Description

As already discussed, the goal of any illumination algorithm is to generate synthetic images based on a set of parameters and conditions representing a virtual environment. The illumination problem is inherently a visibility problem. For each light-surface interaction within the environment, we need to either identify the surface locations reachable by photons emitted from a point on a luminary or gather the energy transported via other surfaces directly in the line of sight of a particular gathering point.

Even though this process can be rather complicated, it can be also decomposed into an ordered sequence of specific steps. Thus, the rendering problem can be expressed in the form of a pipeline; a sequence of smaller operations executed in a linear fashion, where the output of each step forms the input of the next one, possibly in a re-entrant manner. This is called the *rendering pipeline*. A useful property of pipelining is that large problems can be broken down into a sequence of subproblems, where each one can be executed efficiently by a combination of specialized algorithms and parallelization. Furthermore, improving the overall efficiency of the pipeline eventually boils down to identifying and refining the slowest stages of the pipeline [AMH08].

Historically, the divergent design considerations between offline and real-time applications resulted in different approaches to resolve direct visibility, resulting in the two most popular rendering pipelines employed today in computer graphics: rasterization and ray tracing. While they may seem conceptually different, they are fundamentally similar; both of them can be (algorithmically) expressed in the form of a double loop, where the order of operations in each one is switched. Due to this, and before we proceed onto a more in-depth analysis of the problems we investigated in this thesis, we start our discussion on a more abstract level, where we explain both the striking similarities between these two concepts as well as their differences.

1.2.1 Classification of Rendering Pipelines

Regardless of the underlying operations, rendering pipelines operate under the following two conditions: (i) given a set of three-dimensional objects (or sometimes, two-dimensional shapes) and a set of two-dimensional pixels on the image plane, the color of each pixel in the output image must be computed, and (ii) the accurate depiction of an image must involve the determination of visible surfaces from the particular viewpoint the scene is rendered. Algorithmically, the first condition can be expressed in the form of a double *for* loop and has led to two major algorithmic families: *object-order* and *image-order* algorithms.

An object-order algorithm, as its name indicates, iterates over all objects first. Each object, or primitive, undergoes a sequence of operations and is eventually projected onto the image plane. For each pixel overlapped by the primitive, its per-pixel color is computed but not necessarily stored in the image buffer. This is determined by a visible surface determination algorithm, such as the Z-buffer [Cat74], which updates the image buffer only if the primitive encountered is the closest one up to that point. Alternatively, a more advanced data structure may be employed, such as per-pixel linked lists, which stores a depth-ordered sequence of all per-pixel primitives. Object-order algorithms have the advantage of processing each primitive only once, while the per-pixel shading operation might occur more than once. The latter can result in wasted operations when only directly visible surfaces are computed.

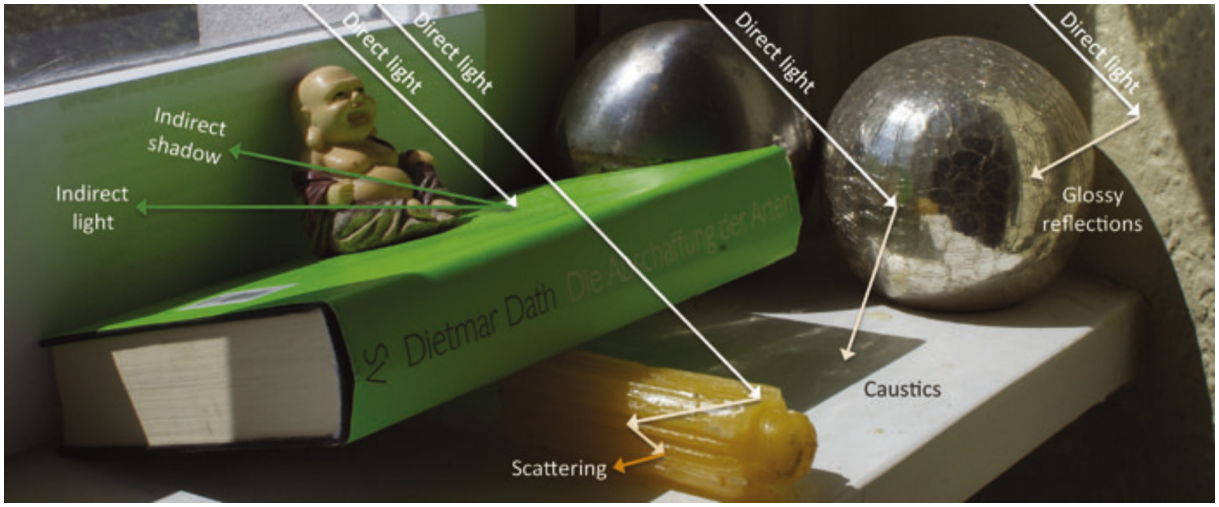


Figure 1.1: A real-world photograph demonstrating global illumination. Areas not directly visible to the light source would appear dark under a local illumination model. These media can be reached by photons through subsequent scattering events, producing phenomena such as caustics, glossy reflections and the "bleeding" of the color of an object into adjacent ones. (Image source: [Rit+12])

An image-order algorithm iterates over all pixels first. A ray is cast from each pixel location and checked for intersections against the entire set of primitives. The pixel color is then evaluated once for the closest intersected primitive. Image-order algorithms have the advantage of processing (shading) each pixel only once, i.e., there are no wasted overdraw operations. On the downside, the entire set of primitives must be traversed per pixel. This is a computationally expensive operation and requires advanced acceleration structures to drop the traversal cost, at the expense of construction overhead.

Both algorithms are well-suited to different scenarios. Object-order techniques are able to resolve direct visibility of opaque surfaces very fast. This is mainly for three reasons: (i) there is no need for the construction of complex data structures, e.g., only an image- and a depth buffer are required, (ii) in typical environments, primitives tend to overlap small regions of the image buffer, exploiting the shared resource data stored in local caches of the graphics processing unit (GPU), and (iii) the total number of primitives is usually lower than the total number of pixels. Hence, they have been highly preferred for real-time applications. There, a convincing amount of realism can easily be accomplished via direct illumination effects, which simulate the appearance of surfaces based on the energy emitted directly from the light sources, i.e., the primary bounce of light, ignoring the illumination received (indirectly) by photons interacting with all the other objects in the environment (see Figure 1.1). While this was traditionally deemed sufficient enough, the simulation of indirect illumination phenomena that is needed in today's fully complex and dynamic virtual worlds, is a difficult task. Consequently, developers end up employing various approximative techniques to simulate isolated effects, commonly implemented as a series of post-processing operations [And15; Mey16], resulting in an increase in the complexity of modern graphics engines, sacrificing both the workflow and productivity.

On the other hand, global illumination phenomena are easily and accurately provided by image-order techniques, where spawning and tracing rays throughout the environment is inherently compatible with the natural propagation of light. However, the computational overhead of the acceleration data structures, in conjunction with the incoherent memory patterns of secondary rays are still restricting these techniques to partially static content.

1.2.2 The Rasterization Pipeline

The *rasterization* pipeline has been the method of choice for real-time applications. Operating in *object-order*, direct visibility is determined by iterating through scene primitives, projecting them into an image plane, sampling their surface and storing these sampled surfaces in a *depth buffer* (see Figure 1.2).

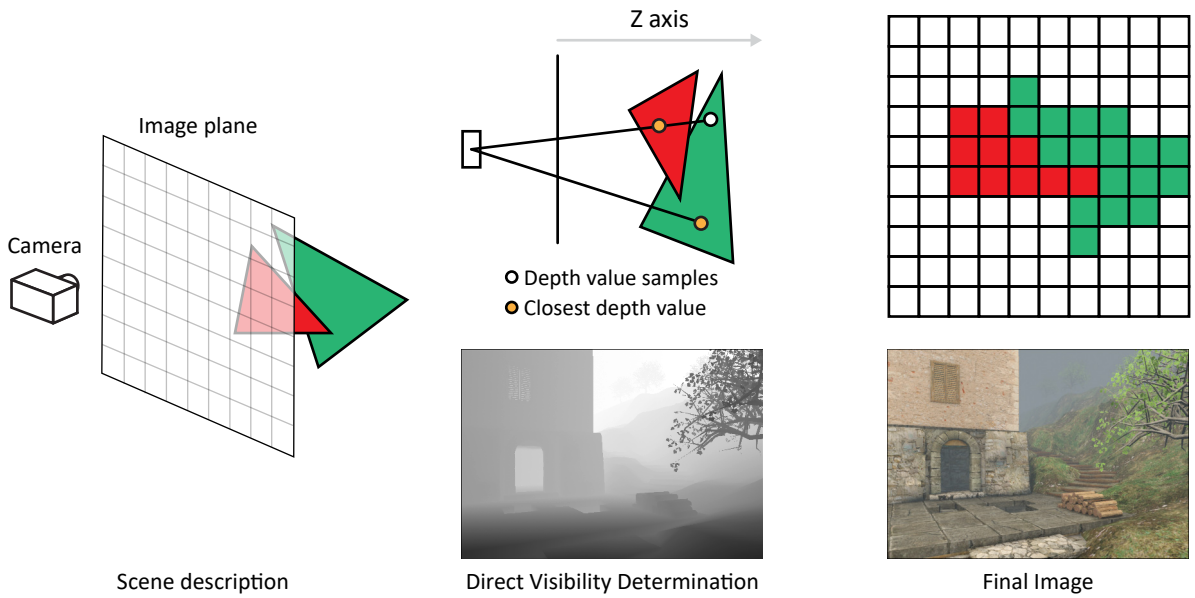


Figure 1.2: Simplistic example of the rasterization pipeline. Given as input a description of the scene (left), the final color is computed by projecting primitives onto the image plane and sampling them (top right). The correct depth order (with respect to the camera viewpoint) of the primitives is determined via the use of a depth buffer, which iteratively maintains the closest depth values for each pixel (top middle). A real-time example of a depth buffer, along with the final image, is shown at the bottom.

The advantage of rasterization is due to its practicality. It can be easily implemented in hardware and can therefore exploit the available rasterization units, allows for high local data cache-coherence since primitives tend to overlap neighboring pixels and is, in general, a more efficient algorithm in typical scenarios, where the number of primitives in a scene is lower than the number of pixels. Furthermore, primary visibility, i.e., the visible surface locations with respect to the camera viewpoint, is resolved very fast. These are the main reasons behind its popularity in real-time graphics. The major problem of rasterization lies in its difficulty to incorporate global effects efficiently which, in conjunction with the various approximations involved in the process, has a direct effect in the resulting quality. These issues have resulted in a large amount of research focusing on the improvement of the quality of these methods.

The two dominant approaches in rasterization operate either in the *image domain* (screen-space), or jointly, in a discrete *volume domain*. Screen-space techniques focus on the use of user-centered information, i.e., originating from a particular viewpoint and within the viewing frustum (the field of view of the camera). This is demonstrated in Figure 1.3. Despite their fast performance, all calculations are based on a subset of geometric information stored in a set of image buffers, representing only the part of the environment that is visible through the viewing frustum. This, along with the discrete sampling nature of rasterization, severely approximates the input to any illumination algorithm. In more detail, primitives are only captured if they are present at specific sampling locations inside the viewing frustum, usually the center of the pixel. As a result, geometry oblique to the view direction is sparsely sampled and objects either parallel to the view direction or residing outside the view frustum are skipped entirely. These approximations are manifested in the form of *view dependencies* and comprise the main source of visual artifacts in image-space methods. On the other hand, volume-based methods are able to capture the entire scene by discretizing it into regularly sized blocks, or *voxels*, where each voxel serves as a representative of the volume of the enclosed portion of the environment (see Figure 1.4). However, they assume a uniform granularity of the represented geometric detail, limiting the quality of the information that can be stored to the size of each voxel. Essentially, this assumption poses an upper-bound in the frequency of the simulated phenomena as contact details and small-scale surface information cannot be accurately represented. The rasterization pipeline, which is the basis of all methods developed in this thesis, is discussed in more detail in Section 2.6.

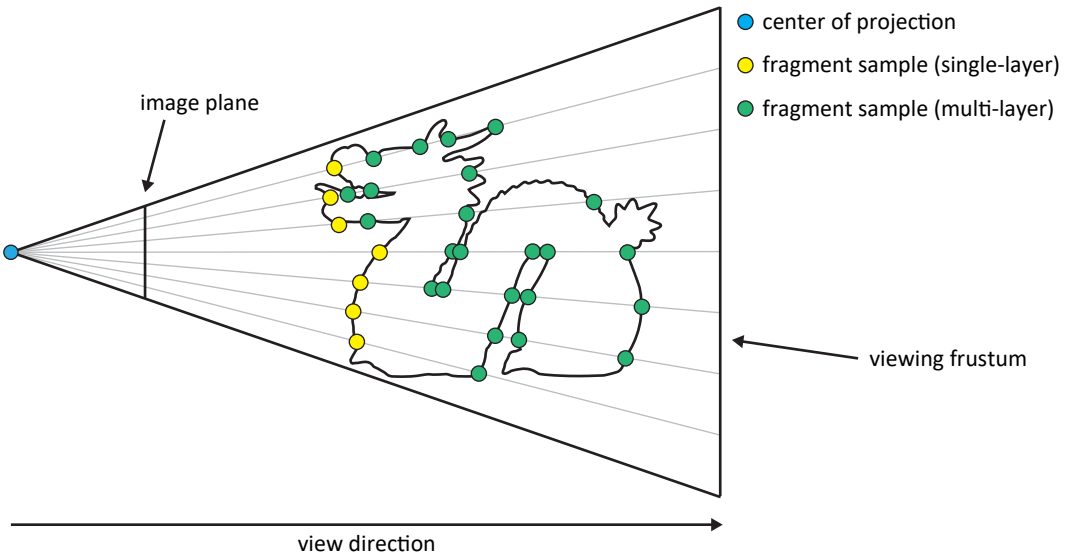


Figure 1.3: Image-based methods capture the scene with respect to a particular viewpoint (blue circle). For each pixel on the image plane, a series of samples (fragments) are generated and stored. Depending on the technique, the stored samples can represent only the first visible surface (single-layer - yellow circles), or its subsequent ones as well (multi-layer - green circles).

1.2.3 The Ray Tracing Pipeline

Ray tracing is considered the most widely accepted approach for physically correct simulation of global illumination phenomena. Algorithms based on ray tracing provide more direct means (compared to rasterization) to determine visible surfaces and perform illumination computations. In general, the process involves the construction of an object-based spatial data structure and the tracing of rays throughout the environment to identify the closest ray-objects intersections, without involving projections or scene approximations. On the downside, ray tracing requires that for every ray, the entire environment must be traversed. Clearly, this requires the use of complex acceleration structures, as naive traversal can become prohibitively slow even for simple environments.

A common ray tracing pipeline consist of two simple stages: a construction stage and a ray traversal stage. Typically, the acceleration structures employed in ray tracing are usually based on some form of spatial partitioning scheme, such as kd-trees and octrees or on bounding volume representations. The major benefit is of these structures is that they are organized hierarchically, which reduces the traversal complexity from $O(n)$ to $O(\log n)$. However, they are computationally expensive to construct and this process has been historically considered more of a preprocess operation. In order to support some form of interactive content, the data structure must be either partially updated or entirely rebuilt from

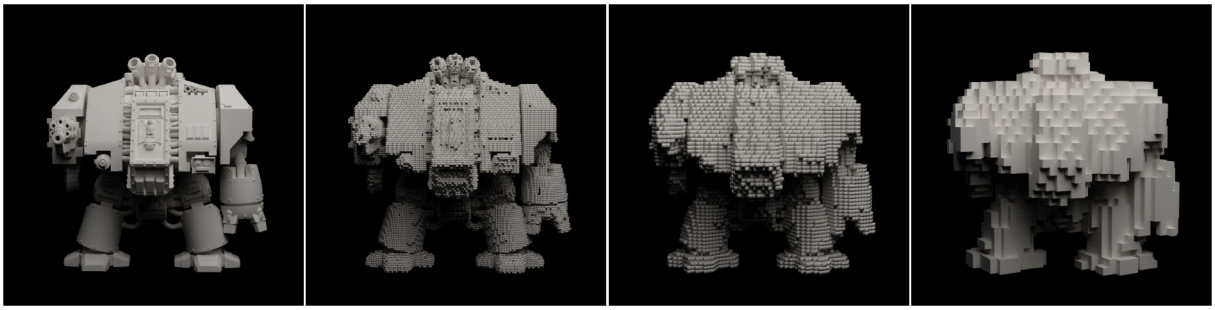


Figure 1.4: Volume-based methods discretize the scene into regularly sized voxels of uniform granularity. Left: The high-detailed model. Middle-left to right: Successive voxelizations of the original version.

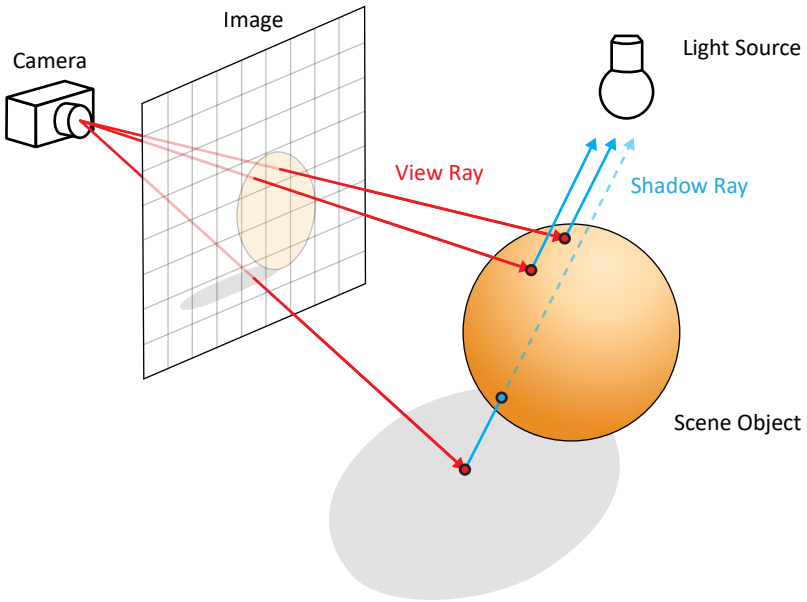


Figure 1.5: Illustrative example of an image-driven ray tracing pipeline. Rays are generated from the virtual camera, crossing the image plane, and traced within the environment. The final color of each pixel is decided based on the intersections of these rays with the scene objects. (Image adapted from: [Wika])

scratch. Alternatively, simpler data structures can be employed, such as regular grids, which offer much faster construction times. However, this comes at the expense of efficient ray traversal as empty space skipping is unavailable. A brief discussion on the research conducted in spatial data structures is offered in Section 3.1.2.

During traversal, rays can be shot from any location in the environment to simulate light transport for various purposes. For example, one can trace the trajectories of photons by sending out rays from the surface of luminaries and following the paths that are formed after the photons are deflected on surface boundaries. In image generation, a very common approach is to generate a set of "primary" rays from the camera that cross each pixel and are directed towards the environment, e.g., as in an image-order algorithm. By following the rays' trajectory as they intersect and get scattered off or through objects, we can estimate the light energy that contributes to each pixel, and shade it accordingly (see Figure 1.5). There are various ways in which ray traversal can be exploited, ranging from direct visibility calculations [App68] and simple reflection/refraction effects [Whi80] to probabilistic [Kaj86] and mutation-driven approaches [Vea98]. These methods are discussed in more detail in Section 2.4.2.

The advantage of these approaches lies in their simplicity. By simulating the flow of light propagation common physical effects are indirectly provided, allowing the production of visually stunning images. However, the resource-demanding construction stage that is required frequently in interactive content, the irregular memory access patterns observed mainly during the tracing of incoherent rays and its hardware implementation difficulty have, traditionally, restricted ray tracing to static content and production rendering.

1.2.4 Problems in Real-time Global Illumination

The technical challenges posed by the strict time restrictions of real-time applications require developers to implement and integrate highly efficient algorithms with very controllable and bounded performance into their applications, making it an intriguing field of research in computer graphics. As already discussed, the majority of these techniques are based on the rasterization pipeline. There, real-time illumination algorithms have to employ a series of assumptions and approximations in order to ensure the highest possible interactivity and minimization of latency.

The most crucial simplification is induced on the environment where a full geometric representation,

that is, a primitive-based structure, is replaced by either the directly visible surfaces from the camera viewpoint in screen-space methods or by a voxel grid in volume-based ones. This modification is important as it decouples the scene complexity from the algorithmic input, making possible the development of robust and efficient algorithms applicable to dynamic environments with important gains in terms of rendering times, memory size and bandwidth requirements.

A second important simplification is performed on the algorithmic operations, e.g., the rendering equation. The most common practice involves the computation of direct and indirect illumination separately, since direct effects can be computed very efficiently in the rasterization pipeline. The indirect part, however, is computationally expensive even for non real-time applications and, historically, it was replaced by a constant ambient term. In the last few years, numerous illumination techniques have been proposed that are able to achieve sufficiently convincing results for the purposes of real-time rendering. Despite being quite diverse in the way they operate, the simplifications imposed on the rendering equation are usually similar: the generic scattering distribution function is replaced by specific distributions modeling ideally diffuse or glossy surfaces, the number of light paths is restricted to one or two, participating media are often omitted and, finally, occlusion between indirect light paths is either approximated or entirely ignored. Empirical methods are also used, such as ambient occlusion. The numerous ways the rendering equation is approximated and a thorough categorization of the proposed techniques in the literature are discussed in Sections 2.3 and 2.4 respectively.

It becomes evident that illumination algorithms can pose some very interesting problems in the domain of real-time rendering making in an exciting field of research. In this thesis, we focus on and investigate two of the most popular approaches, screen-space ambient occlusion and voxel-based global illumination, which we describe next.

1.2.4.1 Screen-space Ambient Occlusion

Ambient occlusion (AO) attempts to estimate the "openness" of the environment around a point of interest and is the most extensively used technique for contact shadows, corner darkening and ambient light attenuation (see Section 2.3.2). By exploiting the rasterization pipeline, its screen-space variants can be executed at very low rendering times, support fully dynamic environments and greatly enhance the resulting realism in almost all scenarios. Therefore, a large amount of research interest has been devoted to the improvements of these methods.

In the literature, a multitude of techniques have been proposed in order to improve sampling quality, memory access patterns and the convergence of the occlusion estimator. However, the view-dependent approximation of the algorithmic input causes severe deviations on the occlusion estimator, resulting in visual instability of the rendered image. Surprisingly, this issue has received very little interest from the research community. So far, the proposed methods attempt to mitigate the problem by either combining several depth-peeled layers and storing the highest occlusion or operating via a fixed view configuration relative to the camera position. The limitations of previous methods are: (i) the generation of multiple views imposing an almost linear increase in the rendering times, (ii) a camera-based configuration is not able to capture geometric information parallel to the view direction and, (iii) the view with the highest occlusion is favored rather than the view with better visibility properties between both the shaded point and the samples drawn in its neighborhood.

Our first work, presented in Chapter 4, investigates *all* of the aforementioned issues and also proposes a multiview configuration to compute the occlusion estimator with three major differences: (i) the location of the views is based on arbitrary viewpoints, allowing the use of the readily available shadow maps which imposes no additional cost in the construction stage, (ii) the estimator is computed via an importance-based weighting scheme, favoring occlusion from views with higher importance in screen-space, and (iii) an adaptive sample selection technique ensures that the estimation of ambient occlusion comes in sub-linear computational cost with respect to the number of views.

1.2.4.2 Volume-based Global Illumination

A more accurate and complete computation of indirect illumination requires knowledge of the entire environment, since the energy emitted from the light sources is propagated throughout the entire scene before reaching the camera sensor. For real-time purposes, the ray tracing procedure against the costly spatial acceleration structure is usually replaced by a final gathering operation against a coarse discretization of the scene.

A typical process in these voxel-based methods is as follows: Initially, the scene is discretized spatially into a uniform lattice. Second, the light emitted from the light sources is captured in some form of image-based representation, e.g., cubemaps or some instant radiosity variant. Third, the incoming radiance is encoded as a radial function, whose coefficients are rapidly computed through projection to a truncated series of orthonormal basis functions such as spherical or hemi-spherical harmonics, and injected into the cells of the three dimensional lattice. This process is called *caching*, where each cell serves as a *cache point*, sparsely storing a representative of radiance at that location in space, called the *radiance field*. Energy is then distributed within the volume by exchanging information between cache points through either a sampling procedure or an iterative propagation scheme. In a final step, the indirect illumination is computed by performing a final gathering step in the high-resolution frame buffer. The final color is obtained by selecting a finite set of sample positions, retrieving the (interpolated) coefficients and reconstructing the reflected radiance at each position.

The benefits of voxel-based representations combined with the local data access and interpolation mechanisms of graphics hardware allow for very efficient, but approximate, computations of indirect illumination in scenes of arbitrary complexity and dynamic lighting conditions. In this thesis, we identify three problems existing in prior techniques, affecting both the performance and quality of the indirect illumination. First, the standard practice of expressing the radiance field as a truncated series of spherical harmonics demands a large number of coefficients (for the entire scene) in order to sufficiently approximate the original signal. Subsequently, this increases the bandwidth requirements for the texture fetches during the gathering operation. Second, if in-scattering due to transport of indirect lighting through participating media is absent, storing radiance for all grid points in the volume results in wasted computations. This is mainly evident in large and open virtual environments, a typical scenario in modern real-time applications. Finally, indirect occlusion is very costly to compute (via a cubemap representation), view-dependent (through the depth buffer) or computed via a probabilistic scheme. To this end, we introduced the idea of directional chrominance subsampling for radiance field compression, an optimized positioning scheme of radiance cache points only near surface locations and a view-independent approximate indirect shadowing technique based on a binary geometry volume. These techniques are demonstrated as part of a fast and stable method for indirect diffuse illumination which is presented in Chapter 5. Note that we consider both the chrominance compression and the optimized positioning of cache points as generic contributions that can be applied to other radiance caching techniques as well.

1.2.5 Problems in Interactive Global Illumination

As noted previously in this chapter, interactive applications are dependent both on visually correct results as well as on the ability to manipulate the various parameters that constitute a virtual environment. Therefore, the implementation of techniques able to support global illumination algorithms at interactive rates, are ideal candidates for this rendering category.

In general, any robust interactive illumination algorithm needs to satisfy the following criteria: fast construction, efficient traversal times, bounded memory requirements and support for arbitrarily complex and animated environments. So far, the conflicting nature between ray tracing and rasterization methods has resulted to the proposal of numerous techniques that improve either the interactivity in ray tracing, the quality in rasterization, or both, by combining the two approaches.

In ray tracing, interactive rates require fast construction of the employed acceleration data structure. Throughout the years, researchers have investigated various refitting and updating strategies in order to inject new geometry to the previous frame's acceleration structure. However, the support of real-

time animations, streamed from a shared distributed virtual environment or dynamically generated from interactive manipulation tools, simulations, etc., eventually results to the simple, but costly, solution of completely rebuilding the acceleration structure in every frame.

Rasterization-based methods, on the other hand, can elegantly support dynamic environments due to their very fast construction stage. However, little investigation has been made so far in the literature on their applicability to support a full global illumination algorithm. Here, the major limitations of the previous methods are two: (i) the approximate geometric representation of rasterization (discussed in Section 1.2.2) poses severe limitations for the purposes of interactive rendering and (ii) their forward strategy to fetch the shading properties of each incoming fragment (or voxel) during the construction stage, regardless of whether they participate in the illumination computations or not, results in significant waste of computational resources. Throughout this thesis, we refer to the second issue as *over-fetching*.

Finally, a recent category of methods follow a hybrid approach that attempts to bridge the gap between rasterization-based and ray-tracing approaches. They can provide efficient results but require multiple data representations and are limited either in bounded environments or in the type of effects they are able to compute.

In this thesis, we propose two methods for a robust solution to the problem of interactive rendering by lifting the limitations of prior rasterization-based methods. The first method, presented in Chapter 6, investigates the ability of rasterization to capture the entire environment and effectively perform ray-fragment intersection tests in order to fully support a global illumination algorithm. The presented solution augments the field of image-space ray tracing and provides efficient construction times as well as scalable traversal performance. However, the resulting quality is approximate due to its fragment-based data structure and still prone to over-fetching issues. The second approach, presented in Chapter 7, further generalizes the field of image-space ray tracing by exploring a primitive-based acceleration data structure, providing accurate ray-object intersections and a complete solution to view dependencies and the over-fetching problem.

1.3 Summary of Contributions

In this section, we provide a detailed summary of our contributions which were discussed in the previous sections, a list of our publications as well as a brief discussion of our research framework. In total, we investigated four distinct techniques directly related to global illumination problems, equally allocated to the domains of real-time and interactive rendering.

In Chapter 4, we propose a generic method to address the view-dependent inconsistencies of the screen-space ambient occlusion algorithms. We accomplish this by taking advantage of buffers containing geometric information from other view points, already generated as part of the rendering process. Furthermore, our method introduces an importance sampling scheme for effectively fusing ambient occlusion from arbitrary viewpoints, without overestimating the result or requiring special passes such as depth peeling. The proposed method is not limited to a particular image-based ambient occlusion algorithm and is thus orthogonal to previous work. Our technique maintains the advantages of screen-space methods for real-time rendering, while significantly reducing view-dependent artifacts, in general.

This work, titled “Multi-view Ambient Occlusion with Importance Sampling” has been presented at the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (i3D) in 2013 and has been published in the ACM Digital Library.

In Chapter 5, we introduce a generic approach to improve the efficiency as well as the visual instability of volume-based radiance caching methods in real-time rendering. First, we propose a perceptually-based radiance field compression scheme based on the idea of chrominance subsampling, by expressing the radiance field in luminance/chrominance values and encoding the directional chrominance component in low-order spherical harmonics. This way, we allow the storage of luminance in higher-order spherical harmonics in a smaller (total) memory budget while maintaining the finer representation of intensity transitions. Second, our compressed scheme is combined with an optimized cache population method, by generating cache points only at locations that are guaranteed to contribute to the reconstructed surface

irradiance. Finally, we present a view-independent indirect occlusion technique, which is computed very efficiently based on a binary geometry volume. We integrate our contributions in a very fast and stable diffuse indirect lighting method, with support for arbitrary light bounces and the ability to simulate complex and dynamic environments, as demonstrated in our extensive experimental study. Last but not least, our general qualitative evaluation indicates benefits for offline rendering applications as well.

This method, titled "Real-time Radiance Caching using Chrominance Compression" has been published in the Journal of Computer Graphics Techniques (JCGT) in 2014 and was invited for presentation at the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (i3D) conference in 2015.

In Chapter 6, we start our investigation on rasterization-based ray tracing for the interactive rendering of global illumination phenomena in fully dynamic environments. We propose an image-space ray-tracing method which, for the first time, can effectively capture in high detail the entire environment by combining the advantages of voxel-based approaches (complete geometry) and image-based methods (highly detailed geometric data). Compared to prior work, the proposed method does not require any precomputations or the maintenance of additional spatial data structures, is applicable to arbitrarily complex and dynamic environments and can support a full global illumination solution. We initially rasterize the scene by performing multifragment rendering in a cubemap configuration. During traversal, near and distant geometric information is detected by generalizing screen-space ray tracing on our multiview/layered structure. To retain fast construction times in conjunction with efficient traversal, our data structure is based on a novel A-buffer implementation based on double-linked lists, depth subdivisions and decoupled storage, allowing fast layer sorting, bi-directional traversal and efficient empty-space skipping. Furthermore, an experimental study is provided that demonstrates that our method can fully support global illumination algorithms, such as path tracing as well as ambient occlusion. We conclude with the first complete analysis between multifragment variants in terms of performance and memory usage for ray-fragment intersections. This evaluation showcases that our method is generic and any multifragment variant can be used instead as a trade-off between performance and memory consumption. While the presented method is able to reduce view-dependencies significantly and support the implementation of a full global illumination algorithm, the fragment-based nature of the underlying data structure can only perform approximate ray-object intersection queries, which restricts the accuracy of the final image, and is susceptible to memory over-fetching issues.

The presented work, titled "A Multiview and Multilayer approach to Interactive Ray Tracing" has been presented at the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (i3D) in 2016 and has been published in the ACM Digital Library.

In Chapter 7, we continue our investigation on the field of image-space ray tracing by proposing a novel method which explores a primitive-based acceleration data structure and supports fast construction times, analytic intersection tests and reduced memory requirements. Initially, we build a compact data structure containing the scene geometry based on primitive linked-lists instead of fragment-based information. During traversal and for each light path segment, we trace our acceleration structure for any ray-primitive collisions. Then, for each identified hit, we fetch the required properties and store them in a memory-bounded auxiliary buffer used for illumination computations. This way, we are able to effectively resolve the over-fetching issues of prior approaches. For efficient traversal, we exploit several culling optimizations in image- and depth-space, such as screen-space hierarchical traversal, pixel frustum clipping, depth subdivision and lossless buffer downscaling. Finally, our experimental study demonstrates that our method generalizes the area of image-based ray tracing under the constraints posed by arbitrarily complex and animated scenarios and effectively resolves the two major remaining issues of the technique discussed in Chapter 6.

The presented work, titled "DIRT: Deferred Image-based Ray Tracing" has been presented at High-Performance Graphics (HPG) in 2016 and has been published in the The Eurographics Association Digital Library.

Publication List

The complete publication list is also provided below:

- Kostas Vardis, Georgios Papaioannou, and Athanasios Gaitatzes. *Multi-view Ambient Occlusion with Importance Sampling*. In: *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. I3D '13. Orlando, Florida: ACM, 2013, pp. 111–118. ISBN: 978-1-4503-1956-0. DOI: 10.1145/2448196.2448214.

  URL: http://doi.acm.org/10.1145/2448196.2448214

- Kostas Vardis, Georgios Papaioannou, and Anastasios Gkaravelis. *Real-time Radiance Caching using Chrominance Compression*. In: *Journal of Computer Graphics Techniques (JCGT)* 3.4 (Dec. 2014), pp. 111–131. ISSN: 2331-7418.

  URL: http://jcgt.org/published/0003/04/06/

- Kostas Vardis, Andreas A. Vasilakis, and Georgios Papaioannou. *A Multiview and Multilayer Approach for Interactive Ray Tracing*. In: *Proceedings of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. I3D '16. Redmond,Washington: ACM, 2016, pp. 171–178. ISBN: 978-1-4503-4043-4. DOI: 10.1145/2856400.2856401.

  URL: http://doi.acm.org/10.1145/2856400.2856401

- Konstantinos Vardis, Andreas-Alexandros Vasilakis, and Georgios Papaioannou. *DIRT:Deferred Image-based Ray Tracing*. In: *Eurographics/ ACM SIGGRAPH Symposium on High Performance Graphics*. Ed. by Ulf Assarsson and Warren Hunt. The Eurographics Association, 2016. ISBN: 978-3-03868-008-6. DOI: 10.2312/hpg.20161193.

  URL: https://diglib.eg.org/handle/10.2312/hpg20161193

Research Platform

It should be noted that a custom graphics engine, named XEngine [VP], was developed and served as a research platform for the purposes of this thesis. XEngine was employed for the implementation of various real-time and interactive algorithms and has provided a consistent testing framework for the experimental study of all the methods that were investigated throughout our research. An earlier version of XEngine is provided for free under the terms of the MIT license (MIT) through the AUEB Computer Graphics Group.

1.4 Thesis Structure

This dissertation is divided into 8 chapters. The remaining chapters are structured as follows: In Chapter 2, the necessary background in the field of interactive rendering is laid out, as it is related to this thesis. In Chapter 3, we provide an overview of relevant prior art. In Chapter 4, we present our generic framework for efficiently addressing the limitations of screen-space ambient occlusion techniques in real-time rendering, in terms of their geometric representation. In Chapter 5, we propose our very fast volume-based global illumination technique for diffuse surfaces, based on radiance caching and reflective shadow maps. We explain the concept of radiance field compression and how we support multiple bounces and indirect visibility. In Chapter 6, we present our generic method for fragment-based image-space ray tracing for complex and dynamic environments and a full path-tracing implementation capable of fast construction times and scalable traversal cost through approximate intersection queries. In Chapter 7, we further expand the field of image-space ray tracing by applying the deferred nature of spatial-based methods in the rasterization pipeline, achieving fast construction times and analytic intersection tests in a memory-friendly framework. Finally, in Chapter 8, we provide a review of our research and conclude this thesis.

Chapter 2

Theoretical Background

In this chapter, we provide the necessary theoretical background on the area of light transport and photorealistic image synthesis. We start with an overview of some important mathematical concepts in Section 2.1. Then, we discuss the elements of light transport, their formulations in computer graphics as well as the most common ways to solve the rendering equation, in Sections 2.2, 2.3 and 2.4, respectively. We continue with a brief discussion on color spaces in Section 2.5, and finally, we conclude with the rasterization pipeline, in Section 2.6.

2.1 Mathematical Preliminaries

The development of robust illumination algorithms is fundamentally based on the exploitation of powerful mathematical concepts and tools. For example, the estimation of high-dimensional integrals, such as the rendering equation, is based on numerical analysis and stochastic methods. Similarly, various approximation techniques rely on the properties of functions represented in the frequency domain. Consequently, the illumination algorithms presented in this thesis depend heavily on the understanding of such approaches.

The beginning of this chapter is devoted on two very important mathematical topics, as they are related to this thesis. Monte Carlo integration, along with the basics of probability theory is discussed in Section 2.1.1. Then, the Spherical Harmonic functions are presented in Section 2.1.2.

2.1.1 Monte Carlo Integration

In global illumination problems, we need to estimate the distribution of light in a virtual world, expressed through the rendering equation (see Section 2.2). In its general form, this involves the estimation of a definite high-dimensional integral of a potentially discontinuous and a priori unknown function of incoming light (radiance) for which no closed-form solution exists. In absence of an analytical form, the integral is approximated using numerical integration techniques.

Numerical methods are commonly classified as *deterministic* and *stochastic*. Deterministic techniques, such as quadrature integration, approximate the integral as a sum over n partitions over the domain of interest. They perform reasonably well and have high convergence rates as long as there are no discontinuities present and the number of dimensions is low.

Stochastic methods on the other hand, are better suited to model complex behaviors involving large datasets with a high degree of uncertainty such as weather forecasts, earthquake predictions and Brownian motion, i.e., the irregular movement of microscopic particles due to collisions of molecules with the encompassing medium. Besides that, they are also well-suited to simulate complicated problems expressed in the form of high-dimensional integrals. The most powerful tools for solving the rendering equation are based on Monte Carlo integration [Sob94]. The idea is trivially simple: random samples are chosen over the integration domain, the integrand is evaluated for each sample and, finally, the integral is approximated based on the statistical mean of these evaluations.

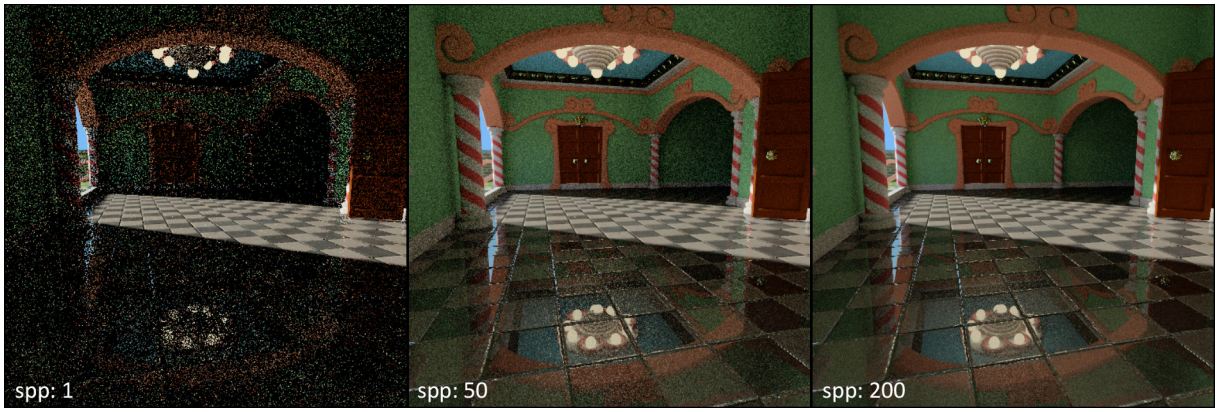


Figure 2.1: Path tracing example of the convergence of the Monte Carlo estimator. High variance, manifested as noise, is dominating the image when only 1 sample per pixel (spp) is used (left). This is gradually reduced by increasing the number of samples (middle and right). Variance reduction techniques have been employed in the generations of these images, such as importance sampling. Image rendered with XEngine [VP], using the method presented in Chapter 7.

Monte Carlo integration is very powerful due to its simplicity; the evaluation of any complex multi-dimensional integral is transformed into a simple expected value problem. Furthermore, its convergence rate is independent of the number of dimensions and, most importantly, it poses no restrictions on the continuity or form of the integrand. Its major drawback is that it suffers from high variance and a large number of samples are typically required to ensure convergence. In illumination algorithms, this is commonly manifested as high-frequency noise. Therefore, a large number of samples is required to reduce the variance and converge to the expected solution. To increase the convergence rate, variance reduction techniques are usually employed. For example, importance sampling is used in order to distribute samples towards areas that contribute more to the final integral value. Another source of variance is due to the use random numbers, which can produce clumping under a finite amount of trials. To reduce these errors, various techniques make use of methods based on the strategic distribution of samples, such as stratified sampling, and low-discrepancy sequences. An example of the convergence of the Monte Carlo Estimator is shown in Figure 2.1.

Next, we briefly review Monte Carlo integration and some aspects from probability theory as they are related to the research conducted on this thesis. For more information, the interested reader is directed to [PH04; Dut+06].

2.1.1.1 Basics of Probability Theory

Stochastic processes are based on the idea of random numbers. A *random variable* x is a variable generated by a random process, based on a set of probable outcomes. Each time an experiment is performed, the value of x is chosen from the given set of outcomes, where each outcome is associated with a certain probability.

As an example, a random variable might describe the outcome of flipping a coin, which would be heads or tails, such as $x_i = [\text{heads}, \text{tails}]$. If the coin is fair, the probability of each outcome p_i is $\frac{1}{2}$. It could also describe the result of rolling a die, which would be a value in the integer set $[1, 6]$. For a fair die, the probability of each outcome $p_i = \frac{1}{6}$. In this sense, a random variable is also understood as a function, a mapping of a set of outcomes to certain values. These are examples of *discrete* random variables, where the set of possible outcomes is finite, or countably infinite (such as the set of natural numbers $\mathbb{N}$). If x is defined over a continuous domain, it is referred to as a *continuous* random variable, e.g., the exact height of a random person, or a point on the surface of a sphere. Since the rendering equation takes values over the domain of real numbers (assuming geometric optics), we restrict our discussion to continuous random variables.

The *expected value* of a continuous random variable x, or the weighted mean, defined over a domain Ω, is:

$$E[x] = \int_{\Omega} xp(x)\,\mathrm{d}x, \tag{2.1}$$

where $p(x)$ is the *probability density function*, or PDF, describing the probability of occurrence of each outcome of the random variable in the range $[x, x + dx]$. In a similar fashion, given a function of random variables $f(x)$, which is also a random variable, its expected value for the same $p(x)$ is:

$$E[f(x)] = \int_{\Omega} f(x)p(x)\,\mathrm{d}x. \tag{2.2}$$

The PDF has the following properties: (i) it is non-negative, (ii) it always integrates to one over the domain of interest:

$$\int_{a}^{b} p(x)\,\mathrm{d}x = 1, \tag{2.3}$$

and, (iii) it is the derivative of the *cumulative distribution function*, or CDF, which describes the probability that a random variable is less or equal than some value y:

$$CDF(x \leq y) = \int_{-\infty}^{y} p(x)\,\mathrm{d}x. \tag{2.4}$$

2.1.1.2 Monte Carlo Estimator

Assume we need to evaluate the integral I of a function f over a domain Ω:

$$I = \int_{\Omega} f(x)\,\mathrm{d}x. \tag{2.5}$$

The idea of Monte Carlo integration is to approximate I by drawing N samples over the integration domain, where each sample is weighted by a probability density function:

$$\hat{I} = \frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)}{p(x_i)}, \tag{2.6}$$

where $\hat{I}$ is the estimator of the target function and $x_1, ..., x_N$ are *independent and identically distributed* random variables.

The Monte Carlo estimator is *unbiased*; the expected value of $\hat{I}$ is equal to the original integral, $E[\hat{I}] = I$. In other words, the estimator will equal the correct solution, *on average*. *Biased* estimators also exist, where the expected value of the estimator is not the correct one. However, a biased estimator might *converge* to the correct solution as the number of samples approaches infinity. In these cases, the estimator is considered *biased but consistent*. In various scenarios, these types of estimators are preferred as they are simpler and faster to compute. The variance σ^2 of the Monte Carlo estimator is:

$$\sigma^2[\hat{I}] = \frac{1}{N}\int_{\Omega}(\frac{f(x)}{p(x)} - I)^2 p(x)\,\mathrm{d}x, \tag{2.7}$$

which shows that the variance is proportional to $\frac{1}{N}$. Consequently, the error decreases at a rate of $O(\frac{1}{\sqrt{N}})$. In other words, the number of samples must be increased by 4 in order for the error to be reduced by 2.

Importance Sampling

As already explained, Monte Carlo approaches suffer from high variance. A major source of variance is due to the choice of the probability density function [Dut+06]. The simplest PDF is the uniform probability distribution, i.e., where each sample has equal probability of being selected over the integration

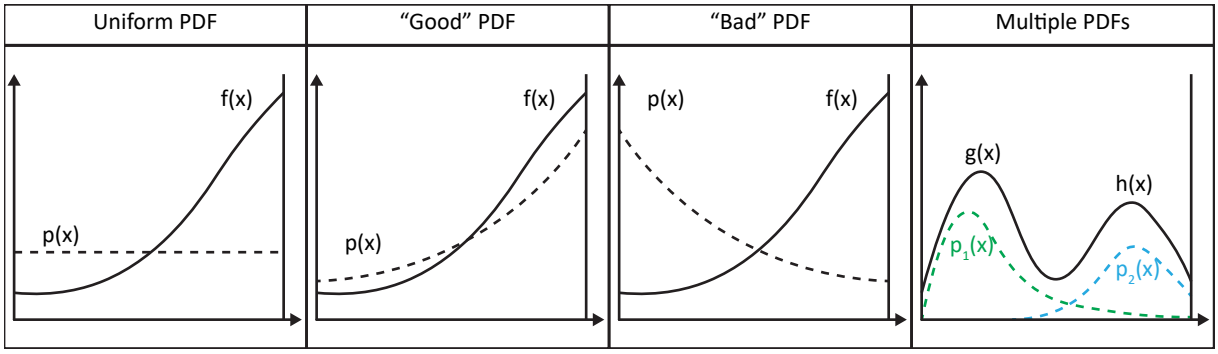


Figure 2.2: The choice of the probability density function is highly important for the reduction of variance. Left: The simplest PDF ($p(x)$), chooses samples uniformly over the domain of integration, which can lead to high variance. Middle left and right: The efficiency of importance sampling techniques depend on the choice of the PDF. A good PDF should match as closely as possible the shape of f, distributing more samples towards areas where the integrand has high values. Right: Multiple importance sampling can be employed in cases where $f(x)$ contains peaks originating from more than one functions, where samples are distributed based on more than one PDFs.

domain. Such a PDF can lead to high variance if a function contains high peaks in several regions, as an enormous amount of samples would be required in order to reach these areas and converge to the expected value of the integral (see left of Figure 2.2). Ideally, the optimal PDF is one that sets the variance to zero. Unfortunately, such a PDF requires knowledge of the integral we are trying to compute:

$$p(x) = \frac{|f(x)|}{\int_\Omega f(x)\,\mathrm{d}x}. \tag{2.8}$$

In other words, an optimal PDF can not be found. However, if we can generate a PDF that exhibits similar characteristics to f, we can achieve significant error reduction. Intuitively, this makes sense: distributing the samples towards regions where f has high (important) values, will increase the efficiency of the estimator. The technique of reducing the variance based on the choice of an alternative distribution function that (partially) matches the form of the integrand is called *importance sampling*. The choice of the PDF in these techniques is particularly important, as performing importance sampling based on a badly chosen distribution function, i.e., distributing samples towards regions where f has low values, can result in highly increased variance, even worse than the one resulting from a uniform PDF. This is demonstrated in the middle insets of Figure 2.2.

Important sampling based on a simple PDF might not always be beneficial. Consider the case where the integrand consists of the product of more than one functions, e.g., $\int g(x)h(x)\,\mathrm{d}x$, and a single PDF cannot be easily derived from them. Distributing samples based on, for example g, would result in slow convergence in areas where the shape of the integrand is dominated by h, as the PDF of g would give very low contribution there (right of Figure 2.2). In these cases, a very powerful strategy is the combination of multiple distribution functions, known as *multiple importance sampling*. In this technique, samples are drawn from multiple distributions and weighted accordingly against all of them, resulting in significant variance reduction.

A common practice in illumination algorithms is the distribution of samples based on one or more of the functions comprising the illumination integral (e.g., the rendering equation); a cosine weighted term, the incident light, or the material properties of the underlying surfaces. For example, in a scene consisting of large area lights and specular surfaces, the probability that a uniformly chosen light sample will contribute significantly to the highly narrow outgoing direction of a reflection event is significantly small. There, distributing samples based on the material's properties, i.e., the bidirectional scattering distribution function (see Section 2.2.3.1), ensures a faster convergence. Conversely, scenes containing small lights and materials with wide distribution lobes can benefit from sampling the light sources, as the probability that samples generated based on the surface properties will reach the light source is again,

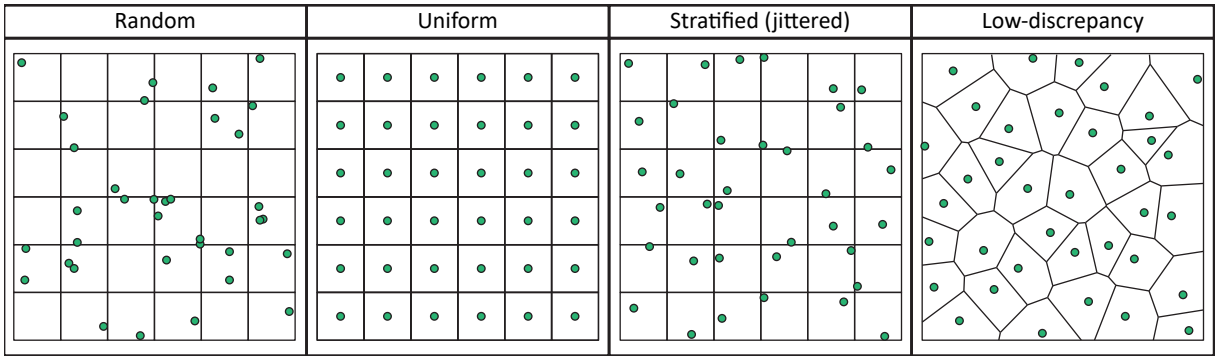


Figure 2.3: Examples of sampling strategies. Left: Employing (pseudo) random numbers can lead to bad distribution of samples, missing large and important areas of the integration domain. Middle-left: Dividing the domain into subregions and placing a sample at each center resolves this issue, but causes aliasing. Middle-right: Stratified sampling jitters the location of samples in each subregion, essentially replacing aliasing with noise, which is less objectionable (perceptually) to the human visual system. Right: Quasi-random sequences (e.g., using the Halton set) exchange the randomness with determinism in order to enforce a better equidistribution of the generated samples.

small. In the case, however, where scenes consist of both specular surfaces and large light emitters, multiple importance sampling is the optimal choice in order to distribute samples based on the both the BSDF as well as the light sources. For more details regarding the intrinsics of multiple importance sampling and its impact on the approximation of illumination integrals, the interested reader is referred to the PhD thesis of Eric Veach [Vea98].

Distribution Techniques

A finite number of samples can end up being poorly distributed over the integration domain, causing *clumping* in various regions of the integrand (left of Figure 2.3). This is a second source of variance, regardless of the PDF used. As such, various variance reduction techniques are focused on the careful placement of samples over the domain of interest.

Stratified sampling partitions the integration domain into n non-overlapping regions. In each region, or *stratum*, a sample is chosen based on a probability density function, usually a simple uniform distribution (middle-left and middle-right of Figure 2.3). This ensures that samples are well-distributed over the entire domain, reducing the estimator's variance more efficiently. However, the total number of samples is dimension-dependent, requiring n^d samples for a d-dimensional integral. Hence, this technique can quickly become prohibitive for a large number of dimensions.

N-rooks sampling [Shi91] retains the subdivision characteristics of the aforementioned techniques, but requires only n samples regardless of the domain dimensionality.

Finally, *quasi Monte Carlo* sampling strategies reduce the variance by entirely replacing the random number generation with deterministic low-discrepancy sequences, enforcing an equal distribution, or minimum separation, between the generated samples [Nie92] (right of Figure 2.3). The Halton and Hammersley sets are commonly used in these techniques, as they are trivial to compute and can easily extend to multiple dimensions. These sets are based on the radical inverse, i.e., expressing positive integers as a sequence of digits of a particular base and reflecting these digits around the decimal point. To generate numbers for the 1- and 2-dimensional case of the Halton and Hammersley sets, the base 2 of the radical inverse (also known as the Van der Corput sequence) is used. For multiple dimensions, numbers can be easily generated by using different bases, each one associated with a different prime number.

2.1.2 Spherical Harmonics

Spherical harmonics [CH65] (SH) are harmonic polynomial functions which satisfy Laplace's equation when restricted to a sphere. They have a direct correlation with Fourier analysis and the Fourier series, which is used to represent functions on the two-dimensional circle instead of the three-dimensional sphere. In Fourier series, a function can be approximated as a linear combination of (infinite) sinusoids, where each weight is essentially a projection of the target function on a particular basis function, representing a specific frequency on the frequency domain. Similarly, spherical harmonics can be employed to approximate functions defined on the surface of a sphere, such as functions included in the rendering equation.

The SH have been extensively used for the representation of BRDFs [CMS87; WAT92], BRDFs and intensity distributions[Sil+91], environment maps [RH01; SKS02] and radiance caching [NPG05; KD10; KED11; MP11; Pap11]. Other forms of representations also exist, such as hemispherical harmonics [Gau+04; Kri+05], Spherical Radial Basis Functions [Leu+06], Wavelets [Gor+93; NRH03; NRH04; Kon+06] and Spherical Wavelets [SS95].

Outside the field of computer graphics, they have a wide range of applications in problems involving spherical symmetry, ranging from the computation of atomic orbital electron configurations in quantum mechanics [Edm96] to the representation of the gravitational and magnetic fields of celestial bodies [Cam03].

In this section we provide a brief overview of spherical harmonics, as they are related to this thesis (see Chapter 5) and the field of computer graphics in general. For a more thorough analysis the readers are referred to [Gre03; Sch05; Ram02; Slo08].

Definition

Spherical harmonics are a set of orthonormal basis functions over the unit sphere. Using the conversion between Cartesian (x, y, z) and spherical coordinates (r, θ, ϕ):

$$(x, y, z) = (r \sin\theta \cos\phi, r \sin\theta \sin\phi, r \cos\phi), \tag{2.9}$$

where $r \in [0, \infty)$ is the radius, $\theta \in [0, \pi]$ is the inclination angle from the z direction, and $\phi \in [0, 2\pi)$ is the azimuth angle. The basis function set of spherical harmonics on the unit sphere are defined as functions of $(1, \theta, \phi)$ as:

$$Y_l^m(\theta, \phi) = N_l^m e^{im\phi} P_l^{|m|}(\cos\theta), \tag{2.10}$$

where $l, m \in \mathbb{Z}, -l \leq m \leq l$. The variable l is referred to as the *degree*, frequency *band*, or *band index* of the corresponding spherical harmonic function and the variable m as the *order*. The band is directly associated with the degree of the polynomials it represents, e.g., band 0 is a constant function, band 1 contains linear functions, band 2 contains quadratic functions, etc. From the inequality constraint of m, it follows that there are $2l + 1$ basis functions for each degree l. The variable N_l^m is simply a normalization constant:

$$N_l^m = \sqrt{\frac{2l+1}{4\pi} \frac{(l-|m|)!}{(l+|m|)!}}, \tag{2.11}$$

and, finally, P_l^m are the *associated Legendre polynomials*. The formula presented in Equation 2.10 contains complex values. However, since illumination computations require only values in the set of the real numbers, a real basis of spherical harmonics is also defined as:

$$Y_l^m(\theta, \phi) = \begin{cases} \sqrt{2} N_l^m P_l^{|m|}(\cos\theta) \sin|m|\,\phi, & \text{if } m < 0, \\ N_l^m P_l^m(\cos\theta), & \text{if } m = 0, \\ \sqrt{2} N_l^m P_l^m(\cos\theta) \cos m\phi, & \text{if } m > 0. \end{cases} \tag{2.12}$$

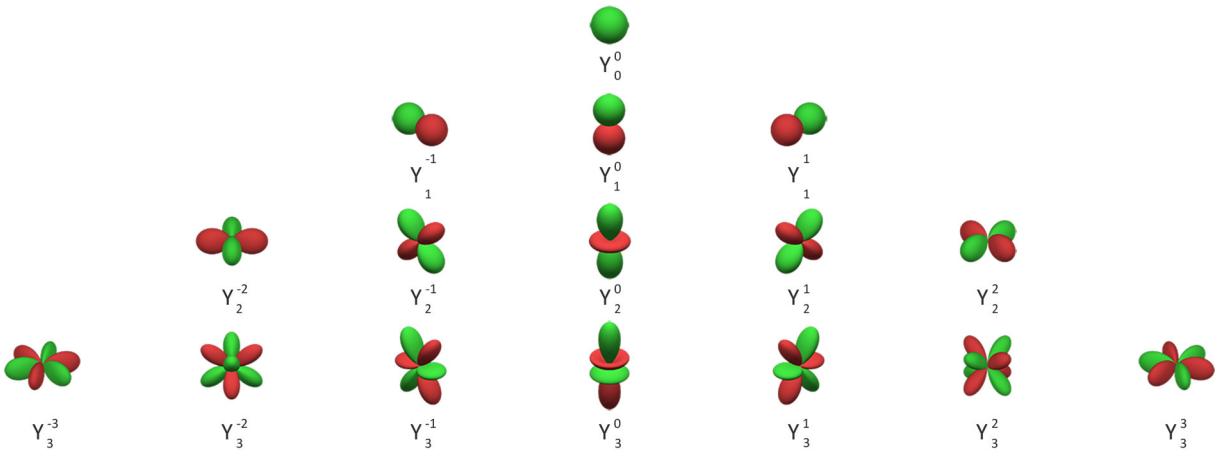


Figure 2.4: The first 4 frequency bands $l = [0, 1, 2, 3]$ of the real spherical harmonics functions where the values represent distance from the origin. Green and red represent positive and negative values respectively.

The associated Legendre polynomials can also be defined in a numerically efficient way according to a recurrence formula [Pre+96], where each polynomial generated is based on prior computations:

$$\begin{aligned} P_m^m(x) &= (-1)^m(2m-1)!!(1-x^2)^{m/2}, \\ P_m^{m+1}(x) &= x(2m+1)P_m^m(x), \\ (l-m)P_l^m(x) &= x(2l-1)P_{l-1}^m(x) - (l+m-1)P_{l-2}^m(x). \end{aligned} \tag{2.13}$$

An example of the first 4 degrees of the real spherical harmonics is shown in Figure 2.4. We next discuss some important operations and properties of spherical harmonics. However, properties such as the integral of triple products and rotations of projected functions are omitted, as they are outside the scope of this thesis.

Orthonormality

Spherical harmonics obey an orthonormality relationship. More specifically, two functions f_i and f_j are said to be *orthogonal* if:

$$\int f_i(x) f_j(x) \mathrm{d}x = k_{ij}\delta_{ij}, \tag{2.14}$$

where δ_{ij} is the Kronecker delta:

$$\delta_{ij} = \begin{cases} 0 & \text{if } i \neq j, \\ 1 & \text{if } i = j. \end{cases} \tag{2.15}$$

Essentially, the inner product of two different functions forming an orthogonal basis is 0, otherwise it is equal to a constant factor k_{ij}. In the special case of $k_{ij} = 1$, the basis set is also *orthonormal*.

As in linear algebra, where the use of an orthonormal basis, such as the standard basis, simplifies various operations in vector space, orthonormal basis functions simplify similar operations in function space. For example, a vector can be described as a linear combination of orthonormal basis vectors, as a function is described as a linear combination of orthonormal basis functions. The coefficients required for the linear combination in vector space describe the *similarity* of this vector against each orthonormal basis vector and are obtained through a projection operation which resolves to a simple dot product. In a similar analogy, the coefficients required for the linear combination in function space are obtained through a projection operation which resolves to a simple inner product:

$$c_i = \int f(x) B_i(x) \, \mathrm{d}x, \tag{2.16}$$

where the coefficient c_i describes the amount of similarity between the target function f and a basis function B_i.

Projection

Spherical harmonics are an orthonormal basis set of functions. Therefore, any spherical function can be approximated as a linear combination, or weighted sum, through an *expansion* or *reconstruction* operation. The coefficients, or weights, are obtained by a *projection* operation of the target function f against each of the basis functions. The formula to compute each coefficient c_l^m is:

$$c_l^m = \int_{\phi=0}^{2\pi} \int_{\theta=0}^{\pi} f(\theta,\phi) Y_l^m(\theta,\phi) \sin\theta \, \mathrm{d}\theta \, \mathrm{d}\phi = \int_{\Omega_{4\pi}} f(\omega) Y_l^m(\omega) \, \mathrm{d}\omega, \tag{2.17}$$

where $\Omega_{4\pi}$ is the spherical domain and ω is the direction of integration. In practice, each coefficient is approximated through Monte Carlo integration (see Section 2.1.1), such as:

$$c_i \approx \frac{1}{N} \frac{f(\omega_i) Y_i(\omega_i)}{p(\omega)_i}, \tag{2.18}$$

where $i = l(l+1) + m$.

Reconstruction

The reconstructed function f is a linear combination of the SH coefficients and the basis functions expressed as an infinite series sum:

$$f(\omega) = \sum_{l=0}^{\infty} \sum_{m=-l}^{m=l} c_l^m Y_l^m(\omega). \tag{2.19}$$

This way, the target function can be reconstructed entirely. However, this requires an infinite number of coefficients and, in practice, the target function is *approximated* through a finite number of coefficients. The approximated function $\hat{f}$ is a band-limited version of the target function f:

$$\hat{f}(\omega) = \sum_{l=0}^{n} \sum_{m=-l}^{m=l} c_l^m Y_l^m(\omega) = \sum_{i}^{N} c_i Y_i(\omega), \tag{2.20}$$

where $(n+1)^2$ coefficients are needed to obtain an band-limited approximation of degree n.

Note that the actual number of coefficients required for a particular nth degree approximation often leads to confusion in the computer graphics literature. For example, a $2nd$ degree truncated series requires a total of $1 + 3 + 5 = 9$ coefficients, corresponding to the number of basis functions present in each frequency band, since the indexing starts at zero. However, this is usually referred as $3rd$ *order* spherical harmonics in graphics research papers. We follow the same naming convention within this thesis, in order to avoid further confusion.

Product Integral

A second important property of orthonormal functions relates the integral of a double product to a set of coefficients in frequency space:

$$\int a(x) b(x) \mathrm{d}x = \sum_{i}^{N} a_i b_i, \tag{2.21}$$

where two SH functions a and b are expressed as a series sum of their coefficients, a_i and b_i.

Convolution

Similar to Fourier analysis, where convolution of two functions f and h in the spatial domain is equal to a multiplication operation in the frequency domain, spherical harmonics define a similar convolution property. Given a spherical function f and a circularly symmetric kernel h, i.e., which has no dependence on the azimuthal angle ϕ, the convolution of these two functions is equal to the product of their spherical harmonics coefficients, weighted by a constant factor:

$$(h * f)_l^m = \sqrt{\frac{4\pi}{2l+1}} h_l^0 f_l^m. \tag{2.22}$$

This property has been exploited in irradiance environment maps, where their computation can be seen as a convolution between the clamped cosine factor and incoming radiance [RH01].

Rotational Invariance

Spherical harmonic functions are rotationally invariant. This property, which can be thought of as analog to the shift-invariance property of the 1D Fourier Transform, means that projection operations are unaffected by rotations applied to the input functions. In other words, rotating a light source and projecting it to spherical harmonics will produce the same result as projecting the (unrotated) light source and rotating its SH coefficients.

2.2 Light Transport Theory

The development of *illumination algorithms* requires a deep understanding of the physical laws and properties that explain the interaction of light and matter as well as the derivation of mathematical models that formulate the desired phenomena. In computer graphics, an important distinction exists between *local* and *global* illumination algorithms based on the number of light bounces in a virtual environment. Local illumination algorithms compute the primary bounce of light, that is, surfaces are lit directly from the light sources. Global illumination algorithms on the other hand, compute all bounces of light; surfaces are lit both directly from the light sources *and* indirectly through the scattering and transmission of photons within the environment. The latter class of algorithms can be considered as a superset that includes the local illumination methods, while the underlying theory is the same.

This section starts with a brief description of the different light models, in Section 2.2.1. Basic radiometry, which defines the elementary quantities needed for light transport, is explained in Section 2.2.2. The interaction of light with different materials is discussed in Section 2.2.3. Finally, the most important mathematical formulations of light transport with and without the presence of participating media are presented in Section 2.2.4.

2.2.1 Light Models

The theory of light transport describes different phenomena by assuming different models of light representation. These are classified as *geometric*, *wave* and *quantum* optics. Each of these models captures different properties of the dual wave-particle nature of light as it interacts with the environment at different levels of detail.

Geometric optics or *ray optics* is the simplest light model and includes phenomena such as *absorption*, *emittance*, *reflection* and *refraction*. It assumes that light propagates in straight lines (as a ray) in homogeneous media or void space and is not affected by magnetic or gravitational fields. It comprises a valid approximation of the light transport modeling, provided that light interacts with objects larger than the wavelength of light.

Wave optics, described by Maxwell's equations, study the light as an electromagnetic wave. This model encompasses the properties of geometric optics but also explains effects such as *interference*,

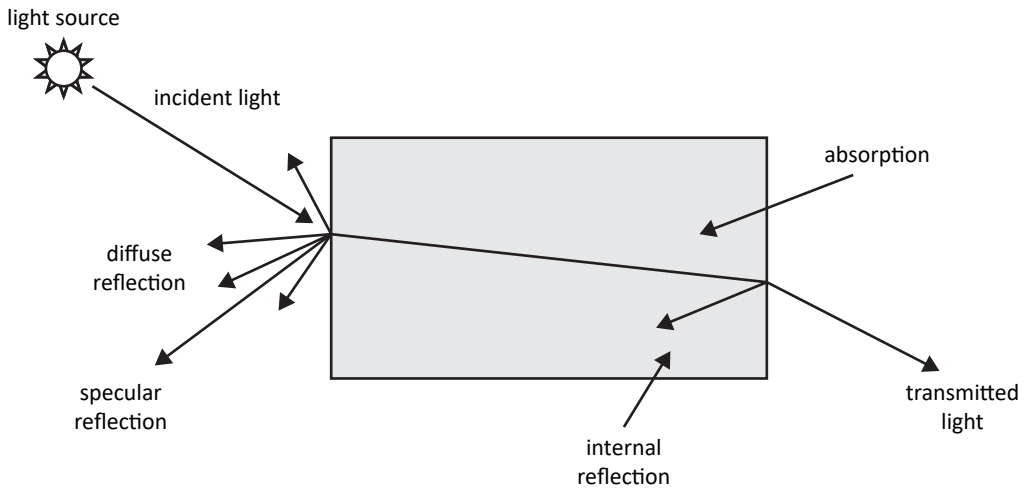


Figure 2.5: Simplified illustration of the geometric optics model. Light emitted from a light source can undergo reflection, transmission and absorption as it interacts with the various objects in the environment. (Image source: [The+08])

diffraction and *polarization*. These phenomena occur when light interacts with geometry of comparable size to the wavelength of light.

Quantum optics, the most comprehensive and complete light model to date, explains the behavior of light at the subatomic level, which exhibits both particle and wave properties. This model can capture photoluminescent effects such as *fluorescence* and *phosphorescence*. In fluorescence, photons are absorbed after interaction with matter and emitted at a different wavelength. In phosphorescence they are emitted at different times and wavelengths. Other phenomena such as blackbody radiation and the photoelectric effect are also explained by quantum optics. Feynman's book [Fey88] in quantum electrodynamics offers an intuitive look on the theory of light and matter.

Quantum optics comprise a very complicated model for use in computer graphics. The dual particle-wave nature of light, the required knowledge of quantum mechanics and the probabilistic behavior of photons make quantum optics a highly complex and unintuitive model to be incorporated in rendering applications. Wave optics explain several important effects, but this model is still not included even in production level renderers, despite some promising work in the area [WK90; WTP01; WW08; WW12; Cuy+12; HP15].

Computer graphics typically use the geometric optics model (see Figure 2.5). This has proven sufficient as it covers some of the most useful optical phenomena of what we see everyday and is easier to simulate. A further simplification in several illumination algorithms is related to the absence of *participating media*. When the medium between surfaces is considered to be vacuum (or a non-interacting solid medium) instead of air and devoid of any particles, light scattering occurs only at the interface of surfaces and therefore, effects such as fog cannot be simulated. A more detailed discussion on light transport and how it relates to several fields (including computer graphics) can be found in Veach's thesis [Vea98].

2.2.2 Radiometry

The field of radiometry describes the quantities needed for measuring electromagnetic radiation. Radiometric quantities are used in global illumination algorithms to measure the way photons propagate and interact with different media in a scene. A related field, *photometry*, provides measurement units that take into account the fact that the human eye photoreceptors have different response curves to the different wavelengths of visible light, which ranges roughly between 380 and 750 nanometers. Since radiometric and photometric units differ only by a weighting function, all quantities used in global illumination calculations are in radiometric units by convention. We now discuss the basic radiometric quantities, which are also summarized in Table 2.1 and illustrated in Figure 2.6.

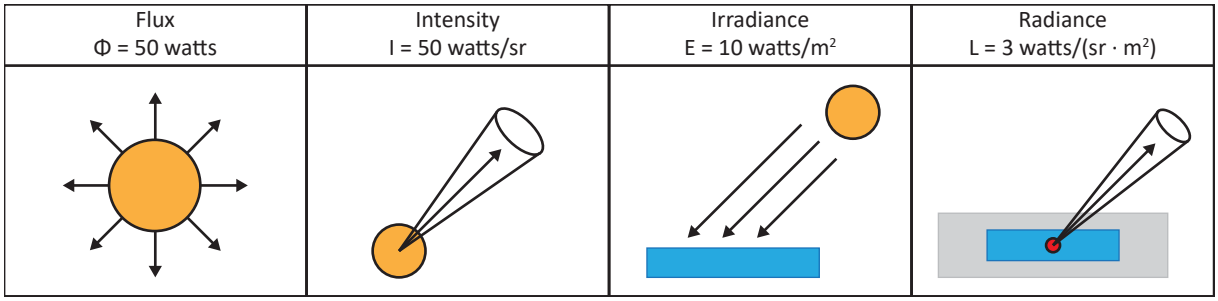


Figure 2.6: Different radiometric quantities. From left to right: Radiant flux, the total power with no notion of direction or area. Radiant Intensity, expressing radiant flux with respect to *direction*. Irradiance, which is radiant flux over *area*. Radiance, measuring radiant power with respect to both area **and** direction.

Radiant Flux

The most basic radiometric quantity is *radiant flux* or *radiant power*. It expresses the energy of electromagnetic radiation (in joules) per unit time (seconds). It is denoted by Φ and measured in Watts $[\mathrm{W} = \mathrm{J} \cdot \mathrm{s}^{-1}]$. This quantity has no notion of surface, direction of propagation or distance. It is used to express the amount of power, that is either emitted by a light source or propagated through a surface, integrated over all wavelengths. All other radiometric definitions are based on the quantity of radiant power.

Radiant Intensity

Radiant intensity I expresses the propagation of power Φ over a certain direction. Direction is expressed as a solid angle $\mathrm{d}\omega$, which is the three-dimensional equivalent of an angle, and the unit of its measurement is the *steradian* (sr). Intensity is given by:

$$I = \frac{\mathrm{d}\Phi}{\mathrm{d}\omega}, \tag{2.23}$$

and has units of $[\mathrm{W} \cdot \mathrm{sr}^{-1}]$. For example, an isotropic light source (which emits light uniformly) can emit 10 Watts per steradian in all directions.

Irradiance

The *irradiance* E is defined as the radiant power Φ incident per unit surface area A. This quantity has no notion of direction and is expressed in $[\mathrm{W} \cdot \mathrm{m}^{-2}]$. Irradiance is given by:

$$E = \frac{\mathrm{d}\Phi}{\mathrm{d}A}. \tag{2.24}$$

Equivalently, the *radiant exitance* or *radiosity* B (or M in some texts) expresses the radiant power leaving a surface (in all directions):

$$B = \frac{\mathrm{d}\Phi}{\mathrm{d}A}. \tag{2.25}$$

For example, a uniform emitter may have a radiant exitance of 5 Watts per square meter at each surface point.

Projected Area

Light striking a surface is maximized (in terms of photon density) when the light direction is perpendicular to the surface. As the light direction changes, the number of photons intercepted by a particular patch on the surface of area A decreases proportionally to the cosine between the light direction and the surface normal vector. In other words, the density of the photons per unit area decreases as a fixed amount of

photons per incident solid angle spreads out over a larger (projected) surface. The relationship between the projected area (the area perpendicular to the light direction) and the surface area is:

$$A_{proj} = A\cos\theta = A(\mathbf{n}\cdot\omega), \tag{2.26}$$

where θ is the angle between the surface normal and the light direction and $(\mathbf{n}\cdot\omega)$ is the dot product between the normalized surface normal and the normalized direction of incidence. In general, light incident on a surface is considered for a single side of the surface interface, usually over the hemisphere above the surface.

Radiance

The quantity of *radiance* $L(\mathbf{x},\omega)$ is a measurement of power per solid angle and per projected area, with units of $[\mathrm{W}\cdot\mathrm{sr}^{-1}\cdot\mathrm{m}^{-2}]$. It measures the amount of light passing through an imaginary infinitesimal area $\mathrm{d}A_{proj}(\mathbf{x})$ perpendicular to a differential direction $\mathrm{d}\omega$. Radiance is defined as:

$$L(\mathbf{x},\omega) = \frac{\partial^2\Phi(\mathbf{x})}{\partial A_{proj}(\mathbf{x})\cdot\partial\omega} = \frac{\partial^2\Phi(\mathbf{x})}{\partial A(\mathbf{x})\cdot\partial\omega\cdot\cos\theta}. \tag{2.27}$$

Radiance is a fundamental quantity in computer graphics and especially in lighting calculations because it represents the amount of light traveling on infinitesimally thin beams of light (rays), which are mathematically convenient for the modeling of the geometry of light transport in geometric optics. A common distinction when computing radiance relates to the absence/presence of participating media. When radiance is computed only at surfaces, participating media is not taken into account and light is assumed to travel through vacuum, i.e., without any change of magnitude through the medium. In this case, radiance is referred to as *surface radiance* and it remains constant as it propagates along a path between two surface locations $\mathbf{x}$, $\mathbf{y}$. Mathematically, this is expressed by:

$$L(\mathbf{x},\omega) = L(\mathbf{y},-\omega). \tag{2.28}$$

The above equation states that for two mutually visible surface points $\mathbf{x}$ and $\mathbf{y}$, the radiance arriving at $\mathbf{x}$ from direction ω is the same as the radiance leaving $\mathbf{y}$ towards direction $-\omega$. This is a very convenient reciprocal property for algorithms which ignore participating media, since global illumination can be computed once the exitant surface radiance from all contributing surface points is known. Alternatively, when participating media are present, light can be affected by the medium between surfaces (air, cloud particles, etc.). In this case, radiance needs to be re-evaluated at all locations in-between the given path endpoints due to scattering events within the participating medium (see Section 2.2.4.3) and is referred to as *field radiance*.

Dependence on Wavelength

All the above quantities compute the radiation for all frequencies of the electromagnetic spectrum. *Spectral radiant power*, *spectral radiant intensity*, *spectral irradiance/radiosity* and *spectral radiance* are the same quantities defined per wavelength, so each one accounts also for the contribution from specific wavelength values λ. Integrating each spectral radiometric unit over the wavelength domain computes its "all-frequency" counterpart. For example, the radiance is defined as the integral over all wavelengths in the electromagnetic spectrum as:

$$L(\mathbf{x},\omega) = \int_{spectrum} L(\mathbf{x},\omega,\lambda)\,\mathrm{d}\lambda. \tag{2.29}$$

In rendering applications, lighting operations are usually based on RGB color space triplets, which is defined with respect to the centers of the wavelength response curves of the human photoreceptors (see Section 2.5). For precise illumination models, however, calculations need to be performed between

different spectral distributions, since multiplying different spectra and converting their result to RGB (to be displayed on the screen) is not the same as multiplication between their collapsed RGB representations [Pee93; Hal99; JF99; Smi99; WE02; Bli05]. Still, this approximation is frequently used for efficiency reasons.

Radiometric relationships

Integrating Equation 2.27 over the hemisphere of directions Ω and using Equations 2.24, 2.25 and 2.26, the following relationships between radiometric quantities can be derived:

$$E(\mathbf{x}) = \int_{\Omega} L(\mathbf{x}, \omega_i)(\mathbf{n}_\mathbf{x}\, \omega_i)\, \mathrm{d}\omega_i, \tag{2.30a}$$

$$B(\mathbf{x}) = \int_{\Omega} L_o(\mathbf{x}, \omega)(\mathbf{n}_\mathbf{x}\, \omega_o)\, \mathrm{d}\omega_o, \tag{2.30b}$$

where $L(\mathbf{x}, \omega_i)$ is the incident radiance at a point x and $L(\mathbf{x}, \omega_o)$ is the radiance exiting a point x in direction ω_o. These equations state that the energy arriving/leaving at a point x is the cosine weighted integral of incoming/outgoing radiance over the hemisphere Ω.

Similarly, by integrating Equation 2.27 over Ω and surface area A and using Equation 2.26 we can obtain:

$$\Phi = \int_A \int_{\Omega} L(\mathbf{x}, \omega_o)(\mathbf{n}_\mathbf{x} \cdot \omega_o)\, \mathrm{d}\omega_o\, \mathrm{d}A, \tag{2.31}$$

which describes the total energy passing through a surface A (real or imaginary) for all directions ω.

Summary

The radiometric quantities described in this section, as well as their units, are summarized in Table 2.1.

Table 2.1: List of radiometric quantities

Quantity	Symbol	Unit	Summary
Flux or power	Φ	$\mathrm{W} = \mathrm{J} \cdot \mathrm{s}^{-1}$	Electromagnetic energy per unit time
Intensity	I	$\mathrm{W} \cdot \mathrm{sr}^{-1}$	Radiant power per unit solid angle
Irradiance	E	$\mathrm{W} \cdot \mathrm{m}^{-2}$	Incident radiant power per unit area
Radiosity	B or M	$\mathrm{W} \cdot \mathrm{m}^{-2}$	Exitant radiant power per unit area
Radiance	L	$\mathrm{W} \cdot \mathrm{m}^{-2} \cdot \mathrm{sr}^{-1}$	Radiant power per unit projected area per unit solid angle

2.2.3 Surfaces

When light is emitted from a light source, it propagates within the environment and interacts with surfaces. Each surface's internal structure has different characteristics that define its interaction with light and, therefore, its visual appearance. According to the laws of geometric optics, light striking a surface is either absorbed or scattered. When light is scattered, it can be either reflected or transmitted (see Figure 2.7).

Absorption happens when a photon interacts with the internal structure of an object. The photon's energy is transformed into some other form of energy, such as thermal energy. Absorption is related to the reduction of light energy and does not change its trajectory.

Scattering of light is associated with energy being re-radiated back after colliding with a medium. Hence, it is related to the change of direction of an electromagnetic wave instead of its energy. Scattering can occur either at the boundary of the object or at its interior, resulting in two different phenomena that coexist. In the first case, light reaches the boundary of an object and changes direction without penetrating the surface. This is called *surface reflection*. In the second case, photons are transmitted (refracted)

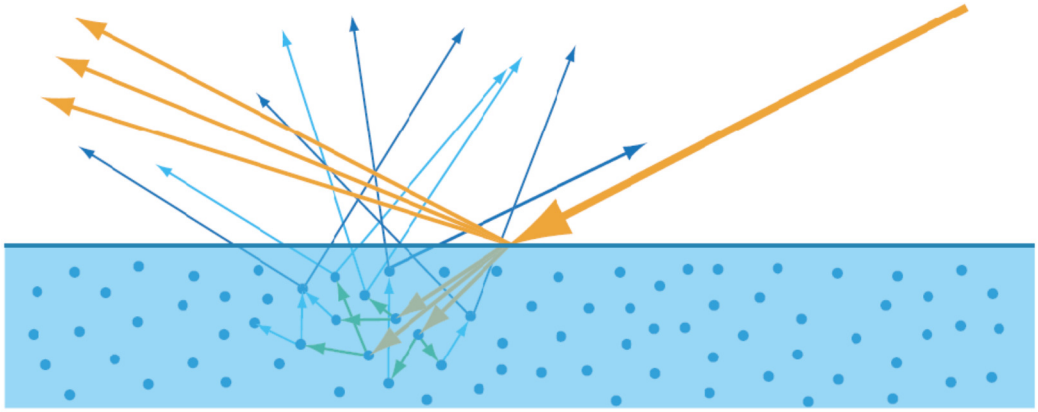


Figure 2.7: Interaction of light with matter. Photons striking an object are either reflected at its boundary (yellow arrows), or transmitted through it. In the latter case, they are internally absorbed and scattered (green arrows) or exit back at the surface (blue arrows). (Image source: [McA+13])

through the boundary of the object and if they are not absorbed, they are internally scattered. This is called *body* or *volume reflection*. When light is internally scattered it can exit back at some other point on the surface, a process called *subsurface scattering*.

Metals, or conductors, are mostly characterized by their surface reflectance since most of the transmitted light is absorbed. Insulators, or dielectrics, however, have lower absorption coefficients, which allows more light to be transmitted and internally scattered. Thus, since surfaces interact with light at their boundary as well as at their interior, both surface and body reflection must be simulated to achieve the correct appearance. *Homogeneous* insulators such as crystals, glass and pure water, transmit light in directions that do not deviate much from *Snell's law* and are considered transparent materials. *Heterogeneous* insulators such as skin, snow and plastic cause light to scatter and change orientation multiple times inside their substance before exiting and are considered translucent materials.

2.2.3.1 Modeling of Surface Reflectance

During rendering, a proper representation of the material properties is required to accurately model and visualize each different object in the virtual environment. In radiometry, this is accomplished by scattering functions. These are mathematical models that are either represented analytically (mostly through statistical distributions) or by using actual measured data.

The most common scattering function used in rendering applications is the *bidirectional reflectance distribution function* or BRDF [Nic+77]. The BRDF f for a particular point $\mathbf{x}$ depends on the incident and exitant light directions and is defined as the ratio of the differential outgoing radiance to the differential irradiance:

$$f(\mathbf{x}, \omega_o, \omega_i) = \frac{\mathrm{d}L(\mathbf{x}, \omega_o)}{\mathrm{d}E(\mathbf{x}, \omega_i)}, \tag{2.32}$$

where ω_o is the exitant direction under consideration and ω_i is the light's incoming direction. Intuitively, the BRDF describes the amount of radiant power that exits the surface at a direction ω_o on the same side of the surface interface as the direction of incidence, given the contribution of radiant power to $\mathbf{x}$ from light arriving at the surface from a direction ω_i.

The scattering function describing the interaction of light with both sides of the surface interface, is the *bidirectional scattering distribution function* or BSDF. BSDF is defined over the entire sphere of directions. For practical reasons, BSDF is typically split into a BRDF term defined over the hemisphere of incident directions *above* the surface (i.e., on the same side as the direction of incidence) and a function of scattering from the interior of the surface, the *bidirectional transmittance distribution function* or BTDF.

However, when the scale of observation is such that the entry point can be considered equal to the exit point and the material represented is assumed to have uniform boundary properties, certain subsurface scattering phenomena are commonly modeled as reflective properties (body reflection or *albedo*) and represented via the BRDF. The BRDF has some interesting properties:

- *Symmetry*. For isotropic materials, where reflectance has the same value for a fixed azimuthal difference between incoming and outgoing directions, the dependence on absolute longitude angle ϕ can be dropped from the equation. For anisotropic materials, such as brushed metal the above assumption does not hold and the anisotropic BRDF is a 4D function of incident and outgoing longitude and latitude.

- *Reciprocity*. For the majority of materials, the BRDF follows *Helmholtz reciprocity*. This means that the directions can be interchanged and the value will remain the same:

$$f(\mathbf{x}, \omega_o, \omega_i) = f(\mathbf{x}, \omega_i, \omega_o). \tag{2.33}$$

  This property is important in some illumination algorithms, since it allows tracing paths either in the direction of light propagation or backwards, without affecting the contribution of the material to the light's surface interaction.

- *Energy conservation*. Physically correct BRDF models obey the law of energy conservation. This means that the total exitant power from the surface is less than or equal to the total incoming flux, unless the surface is also an emitter. A non energy-conserving model does not produce correct results. For example, rough surfaces might exhibit loss of energy, i.e., they appear darker, at glancing angles due to the absence of multiple scattering events [Hei+16]. In general, enforcing both reciprocity and energy conservation is a difficult task [Edw+06].

Common BRDF models

In general, BRDFs and reflectance models are based either on empirical observations or on simplifications of physical laws. BRDF models that focus on modeling the body reflectance are referred to as *diffuse* BRDFs. The simplest body reflectance BRDF and the one that is still widely used in interactive applications is the *ideal diffuse*, *pure diffuse*, or *Lambertian* BRDF [Lam60], which states that a lit surface is viewed equally bright from all directions. In this case, the BRDF is constant, equal to $\rho(\mathbf{x})/\pi$. The value $\rho(\mathbf{x})$, is the *albedo* of the surface at $\mathbf{x}$, i.e., the amount of energy being reflected instead of absorbed.

The simplest surface reflectance BRDF is the *ideal specular* BRDF where the incoming energy is totally reflected in the ideal reflection direction, similar to a perfect mirror reflection. However, the behavior of a more realistic BRDF is characterized by a reflectance lobe in which the scattering of light

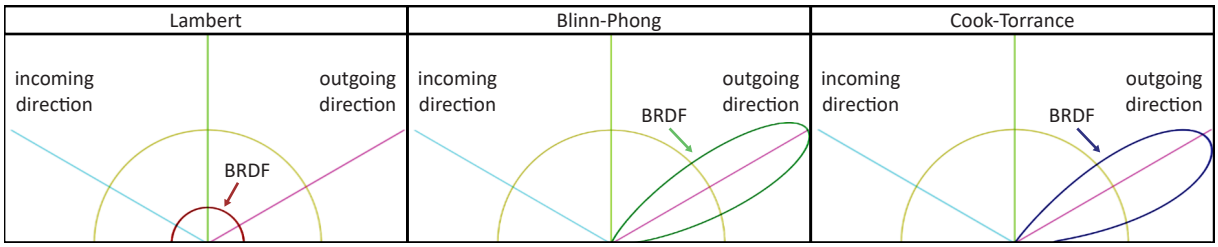


Figure 2.8: Examples of different BRDF models. Left: Lambertian (ideal diffuse). Note that there is no directional dependence. Center: Blinn-Phong BRDF. The directionality and the size of the lobe signifies the direction and width of the specular highlight. The lobe is strongest in the ideal reflection direction. Right: The microfacet-based Cook-Torrance BRDF. The highlight direction is towards more grazing angles compared to Blinn-Phong since it is based on Fresnel reflectance, see [CT82]. BRDF models rendered using Disney's BRDF Explorer [Wal].

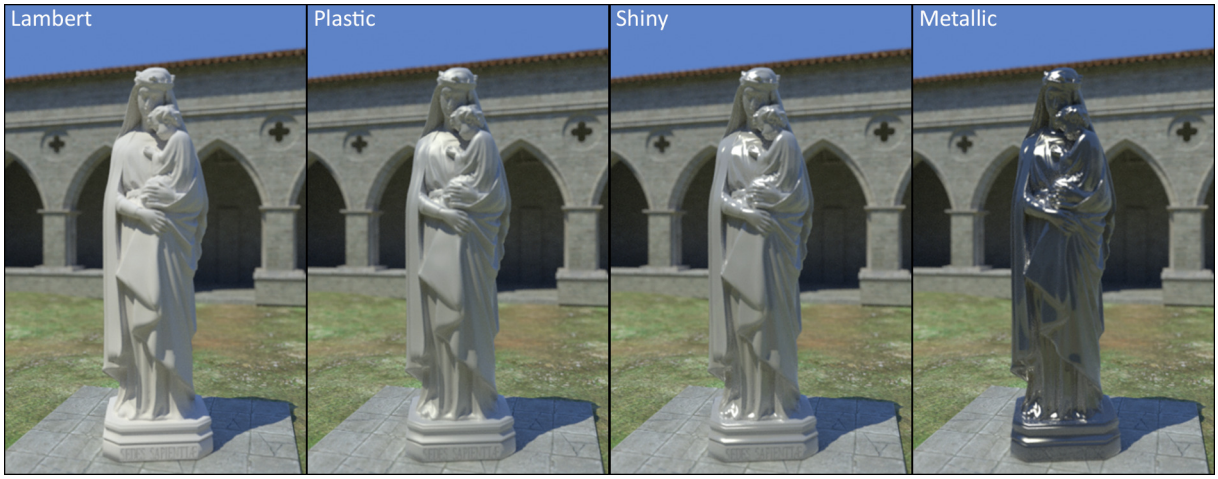


Figure 2.9: BRDF modeling allows for efficient visualization of different types of materials. In this example, different types of materials are illustrated by applying different parameters using the Cook-Torrance model [CT82].

covers a range of directions, instead of a single one. Surface reflectance BRDF models are known as *glossy* BRDFs, with the most historically popular being the Blinn-Phong BRDF [Bli77]. Other models are based on microfacet theory [TS67; CT82; Wal+07], which models the surface as a number of tiny mirror-like or ideal specular planar reflectors. Historically, the most widely used isotropic reflectance model has been the Cook-Torrance model [CT82], in which the micro-facet distribution is taken into account (surface roughness), as well as the Fresnel reflection and transmission coefficients of the material for the particular direction of incidence, along with light masking and shadowing side effects of the micro-facet geometry.

Reflectance lobes for several well-known BRDFs are shown in Figure 2.8. This is a vast area of research and numerous models have been proposed [Pho75; PF90; Str90; He+91; War92; ON94; Laf+97; AS00; Don+09b; MLH02]. We omit any additional details of this area and we refer readers to three comprehensive surveys [Sch94; Shi+97; KE09]. Figure 2.9 illustrates the appearance of surfaces using the Cook-Torrance model.

Measuring real surfaces with the use of gonioreflectometers and image-based methods is an alternative way to acquire BRDFs [Dan+99; Len04; Mat+03; HLZ10; Hul+10; Don+10; AWL13]. A subset of them is available for download in online databases [MER; Cor].

Generalizing the BRDF

The BRDF makes several assumptions regarding the behavior of light, such as the scale of observation and material uniformity. As long as these are respected, a reciprocal and energy preserving BRDF can accurately represent the interaction of light with matter. By lifting these simplifications, more complex surfaces can be accounted for [Wey+08]. However, most of these representations are quite complex and are not very that common in computer graphics. In the most general case (ignoring interference and light polarization) these functions need to describe the behavior of light in multiple domains: spatial, directional, spectral and temporal [VH74], as given by the 7D plenoptic function [AB91]. The most general scattering function, S, is a 14D function which describes the interaction of light with a medium by using two plenoptics, describing light arriving and exiting at an arbitrary position in space. Ignoring phosphorescence and fluorescence phenomena and assuming radiance is constant along rays (i.e., no volumetric scattering), thus omitting temporal and spectral dependencies as well as two spatial dimensions, the *bidirectional surface scattering and reflectance distribution function*, or BSSRDF [Nic+77], can be used. This 8D function captures subsurface scattering phenomena, such as light entering at one location and exiting at another, regardless of the scale of observation and can be used to represent the

appearance of surfaces such as skin, wax, etc. Ignoring subsurface scattering, therefore dropping two more dimensions (of position), the 6D *Spatially-varying Bidirectional Surface Reflectance Distribution Function*, or SVBRDF, can be employed to simulate materials that do not depend highly on subsurface scattering, but do not have uniform properties on their surface. Going further and assuming uniformity over the surface which, strictly speaking, is only valid if the object is viewed from a certain distance, the spatial dependence is dropped and the 4D anisotropic BRDF is used.

2.2.4 Formulations of Light Transport

The goal of global illumination algorithms is to estimate as accurately as possible the photon propagation in an environment. Light is emitted from the light sources, bounces around the scene and interacts with different media such as surfaces and air, undergoes scattering and absorption, under a local energy-preserving equilibrium state. Several formulations have appeared over the years and all algorithms can be expressed through one or several of these. The most fundamental mathematical model in computer graphics was presented by Kajiya [Kaj86], who formulated light transport in computer graphics in the form of an integral equation, called *the rendering equation*:

$$L(\mathbf{x}, \omega_o) = L_e(\mathbf{x}, \omega_o) + L_r(\mathbf{x}, \omega_o), \tag{2.34}$$

which states that the radiance leaving a surface point $\mathbf{x}$ in direction ω_o is equal to the sum of the emitted radiance L_e from that point in the ω_o direction and the radiance L_r leaving the surface in the same direction due to light scattering events at $\mathbf{x}$.

2.2.4.1 Hemispherical Formulation

L_r can be derived by differentiating Equation 2.30a to relate irradiance with incident radiance and integrating the BRDF (Equation 2.32) with respect to solid angle:

$$\frac{\mathrm{d}L(\mathbf{x}, \omega_o)}{\mathrm{d}E(\mathbf{x}, \omega_i)} = f(\mathbf{x}, \omega_o, \omega_i) \Rightarrow \frac{\mathrm{d}L(\mathbf{x}, \omega_o)}{\mathrm{d}L(\mathbf{x}, \omega_i)\cos\theta_i} = f(\mathbf{x}, \omega_o, \omega_i) \Rightarrow$$

$$\mathrm{d}L(\mathbf{x}, \omega_o) = f(\mathbf{x}, \omega_o, \omega_i)\,\mathrm{d}L(\mathbf{x}, \omega_i)\cos\theta_i \Rightarrow$$

$$L_r(\mathbf{x}, \omega_o) = \int_\Omega f(\mathbf{x}, \omega_o, \omega_i) L(\mathbf{x}, \omega_i)\cos\theta_i\,\mathrm{d}\omega_i. \tag{2.35}$$

This is called the *reflectance equation* and is defined over the hemisphere of reflected/incident directions Ω. The corresponding equation for the BSDF is defined over the entire sphere of directions and is called the *scattering equation*.

If participating media are ignored, then the incoming radiance depends only on the outgoing radiance of other surface locations. However, this is a recursive problem. To compute the illumination at a specific surface location, the incoming radiance at point $\mathbf{x}$ from a given direction must be found. This incoming radiance equals the outgoing radiance of another point $\mathbf{y}$, therefore, $L(\mathbf{y}, -\omega_i)$ must be also found and so on. The unknown quantity, radiance, is also shown inside the integral. Mathematically, the rendering equation is known as a Fredholm integral equation of the second kind and is hard be to solved analytically.

2.2.4.2 Area Formulation

The hemispherical formulation considers radiance quantities with two different roles (outgoing / incident) defined locally at point $\mathbf{x}$, although the incident radiance depends on the contribution of other locations on scene geometry. An alternative form of the rendering equation removes the incident radiance at $\mathbf{x}$ and replaces it with the corresponding outgoing radiance of the contributing point $\mathbf{y}$. Starting from the hemispherical formulation of the rendering equation (Equation 2.35), after replacing $\mathrm{d}\omega$ with the

corresponding contributing area of geometry $\mathrm{d}A$ at $\mathbf{y}$, the integration domain is transformed from the hemisphere above $\mathbf{x}$ of all incident directions to the set of all visible points A_v subtended by the hemisphere. Using the definition of a solid angle, in this change of variable, Equation 2.35 becomes:

$$\begin{aligned} L_r(\mathbf{x}, \omega_o) &= \int_{A_v} f(\mathbf{x}, \omega_o, \omega_i) L(\mathbf{y}, -\omega_i) G(\mathbf{x}, \mathbf{y}) \, \mathrm{d}A, \\ G(\mathbf{x}, \mathbf{y}) &= \frac{\cos\theta_i \cos\theta_j}{||\mathbf{x} - \mathbf{y}||^2}, \end{aligned} \tag{2.36}$$

where θ_j is the angle between the normal vector at $\mathbf{y}$ and the normalized direction vector $-\omega_i$. This formulation of the rendering equation involves the determination of the visible surfaces A_v from $\mathbf{x}$ for all sampled points in the scene. A more practical version, which is commonly used in many algorithms is derived if we introduce a binary *visibility* term $V(\mathbf{x}, \mathbf{y})$, which is 1 if the two points are mutually visible, or 0, when the line of sight between them is interrupted by other geometry. Now, Equation 2.36 can be expanded to the entire domain A of all surfaces in the scene, since their contribution to the receiving point $\mathbf{x}$ can be controlled by $V(\mathbf{x}, \mathbf{y})$:

$$L_r(\mathbf{x}, \omega_o) = \int_A f(\mathbf{x}, \omega_o, \omega_i) L(\mathbf{y}, -\omega_i) V(\mathbf{x}, \mathbf{y}) G(\mathbf{x}, \mathbf{y}) \, \mathrm{d}A. \tag{2.37}$$

Using the rendering equation, global illumination algorithms solve the light transport problem recursively by tracing photons emitted from the light sources, tracing random paths from the sensor, or by a combination of both strategies.

Let us now for the sake of simplicity define an operator $Int(L(\mathbf{y}), A)$ corresponding to the integral of Equation 2.37. We can do the same for the hemispherical rendering equation as well. Then, the total outgoing radiance from a point $\mathbf{x}$ is:

$$\begin{aligned} L(\mathbf{x}, \omega_o) =& L_e(\mathbf{x}, \omega_o) + L_r(\mathbf{x}, \omega_o) = \\ & L_e(\mathbf{x}, \omega_o) + Int(L(\mathbf{y}), A) = \\ & L_e(\mathbf{x}, \omega_o) + Int(L_e(\mathbf{y}) + L_r(\mathbf{y}), A) = \\ & L_e(\mathbf{x}, \omega_o) + Int(L_e(\mathbf{y}), A_L) + Int(L_r(\mathbf{y}), A)) = \\ & L_{\text{emitted}} + L_{\text{direct}} + L_{\text{indirect}}, \end{aligned} \tag{2.38}$$

where $A_L \subset A$ represents the union of all light-emitting surfaces in the scene (the luminaries). The above formulation splits the energy contribution of the light incident to a point to a *direct* component, i.e., light coming directly from the light sources, and an *indirect* one that includes all light bounces up to $\mathbf{x}$. Breaking the integral equation into these components is a common approach in illumination algorithms, where each part is solved separately.

2.2.4.3 Participating Media

Light transport as described by the rendering equation assumes that light interacts either with surfaces, or travels in vacuum. While that may be a reasonable approximation, effects such as fog, the blue color of the sky, fire and smoke cannot be approximated. If the medium between two surfaces participates in the lighting calculations then photons can also be absorbed, emitted and scattered. Scattering occurs when a photon interacts with particles in the participating medium causing it to change direction. *Out-scattering* refers to the case where photons are being scattered in other directions and *in-scattering* refers to the case where scattered photons arrive from other directions towards the direction of observation. Absorption and out-scattering cause a decrease in field radiance while emission and in-scattering cause an increase in field radiance.

Figure 2.10: Left: Scene lit with direct illumination only. Areas where light is not present appear dark. Right: Direct and one bounce of indirect illumination. (Image source: [TL04])

Figure 2.11: A typical outdoor scene rendered with ambient occlusion only. Note the resemblance to shading from an overcast sky. (Image source: [Sol])

Extinction

Extinction is the combined effect of absorption and out-scattering. As light travels along a ray through a participating medium, the probability that photons will either be absorbed or deflected outside the path of the ray, causing a decrease in field radiance, is described by the *extinction coefficient* σ_t:

$$\sigma_t = \sigma_a + \sigma_s, \tag{2.39}$$

where σ_a is the *absorption coefficient* and σ_s is the *scattering coefficient*. The fraction of photons that are not scattered or absorbed travelling from a point $\mathbf{x}$ to a point $\mathbf{x}'$ in direction ω is given by the *transmittance function* T_r:

$$T_r(\mathbf{x}, \mathbf{x}') = e^{-\int_0^d \sigma_t(x+t\omega,\omega)\,\mathrm{d}t}, \tag{2.40}$$

where d is the distance between the two points. The negative exponent is called the *optical depth* of the medium. For a homogeneous medium, the scattering coefficient does not vary along the path and the transmittance function becomes:

$$T_r(\mathbf{x}, \mathbf{x}') = e^{-\sigma_t d}, \tag{2.41}$$

which is also known as *Beer's law*. Given the transmittance function, the total radiance arriving at point x' is:

$$L(\mathbf{x}', \omega) = T_r(\mathbf{x}, \mathbf{x}')L(\mathbf{x}, \omega). \tag{2.42}$$

Emission and In-scattering

The phenomena of emission and in-scattering relate to the increase of radiance along the path of the ray. Emission increases radiant energy due to physical processes converting other forms of energy to visible light, while in-scattering increases radiant energy due to photons being scattered from other directions. The total radiance increase at a point $\mathbf{x}$ in direction ω is given by the emitted radiance and the incident radiance around that point:

$$L(\mathbf{x}, \omega) = L_e(\mathbf{x}, \omega) + \sigma_s(\mathbf{x}, \omega) \int_{\Omega_{4\pi}} p(\mathbf{x}, \omega, \omega')L(\mathbf{x}, \omega')\,\mathrm{d}_{\omega'}, \tag{2.43}$$

where L_e is the emitted radiance and p is the phase function which indicates the angular distribution of light arriving at $\mathbf{x}$ in direction ω from the entire sphere of directions $\Omega_{4\pi}$. The simplest phase function is the isotropic which has a constant value of $\frac{1}{4\pi}$.

2.2.4.4 Path Notation

Global illumination algorithms are often classified based on the different phenomena they attempt to solve. This taxonomy is usually based on following the trajectory of photons throughout the environment and their interaction with different surfaces. For example, many algorithms have been developed to simulate diffuse-only environments. Other techniques are designed for capturing caustic effects, which require the tracing of specular paths. Some approaches start measuring the contribution of light at the camera sensor, others trace the emitted radiance from the light sources and hybrid algorithms use a combination of both. A distinction on the number of bounces is also common, as it imposes an increase on the rendering cost.

To characterize these different algorithms, a convenient way is to use Heckbert's notation [Hec90]. According to this notation, photons can interact with different environment types: the camera sensor or eye (E), the light sources (L) and different types of materials (D for diffuse and S for specular). These interactions are frequently called *events*. A number of regular expression operators is also defined to describe sequences of events, as shown in Table 2.2.

Table 2.2: Regular expression notation for photon trajectory events

Notation	Description	Example	Meaning of example
$\star$	Zero or more	D$\star$	Zero or more diffuse bounces
+	One or more	D+	One or more diffuse bounces
?	Zero or one	S?	Zero or one specular bounces
None	One	S	One specular bounce
|	Or	D|S	One diffuse or one specular bounce
()	Group	(D|S)+	One or more diffuse or specular bounces

Using this notation, the whole light transport can be described by L(D|S)$\star$E. That is, light is emitted from the light sources and photons can either reach the eye directly or bounce around any type of surface, any number of times before they finally reach the eye.

2.3 Special Cases of Light Transport

Simulating all light transport effects is a challenging and time-consuming task even in current production renderers. Therefore, the majority of algorithms have been designed with certain phenomena in mind. By excluding certain phenomena, the rendering equation can be simplified significantly. For example, by limiting the type of surfaces to ideally diffuse, the BRDF is set to a constant factor and moved outside the integral. By reducing the number of bounces to a small fixed number, the rendering equation is replaced by a simple summation of finite paths. This section explains the most basic but also the common special cases of the generic light transport equation that are used in interactive and real-time graphics algorithms. These different phenomena include: direct illumination (Section 2.3.1), ambient occlusion (Section 2.3.2), environment mapping (Section 2.3.3), caustics (Section 2.3.4) and diffuse interreflections (Section 2.3.5).

2.3.1 Direct Illumination

The simplest type of illumination is local or direct illumination. These algorithms take into account the direct interaction of surfaces with light emanating only from light sources (luminaries), i.e, those surfaces for which $L_e \neq 0$. In this case, surfaces are illuminated only by the primary bounce of light, without regarding the incident light from transmission on other surfaces or media (see Figure 2.10). In terms of notation, the simulated paths are of type L(D|S)E. Commonly, this type of illumination is implemented separately using local shading models, while the indirect illumination contributes additively to the result via various other algorithms. As a result, some global illumination algorithms are specifically designed to compute only the indirect part of light transport.

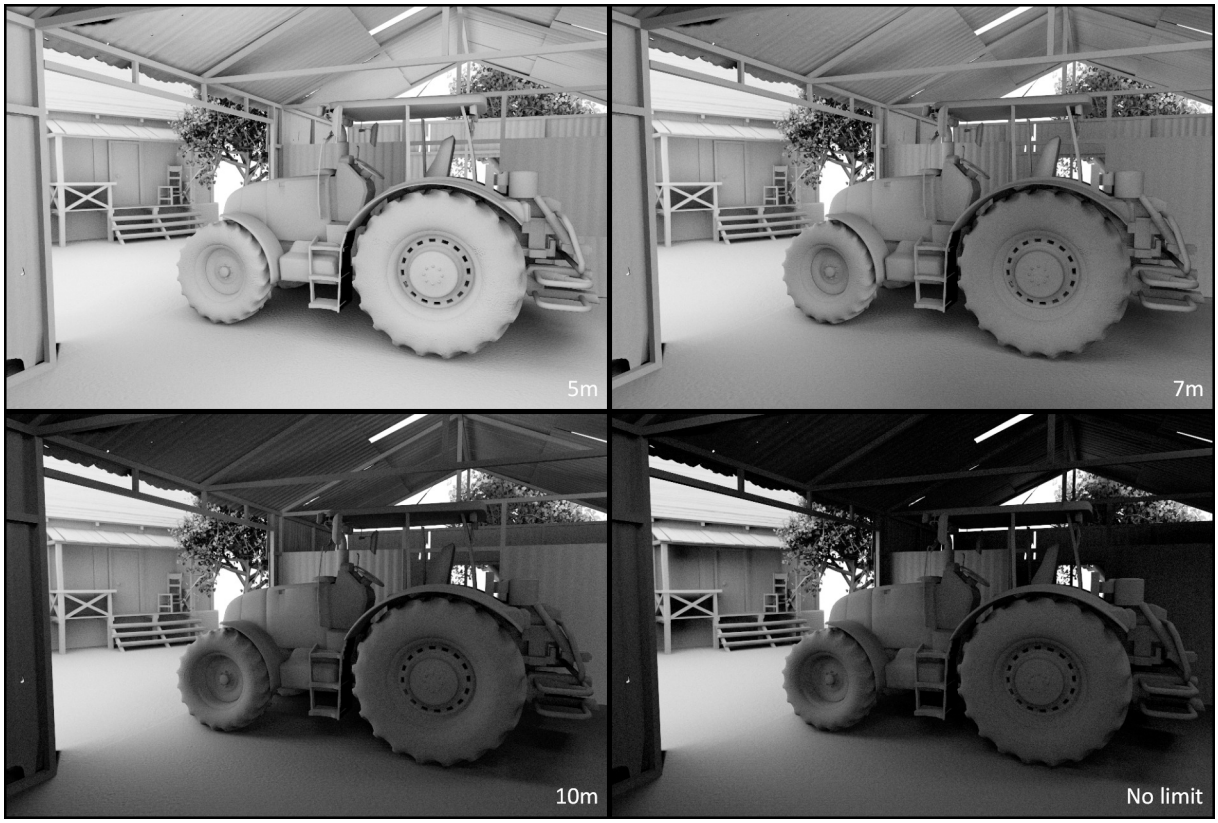


Figure 2.12: Ambient occlusion examples with different range parameters. Note how the images gradually darken by increasing the sampling range. Image rendered with XEngine [VP], using the method presented in Chapter 6.

Using the operator formulation of the rendering equation described in Equation 2.38, direct-only illumination can be isolated by setting the L_{indirect} component to zero:

$$L(\mathbf{x}, \omega_o) = L_{\text{emitted}} + L_{\text{direct}} = \\ L_e(\mathbf{x}, \omega_o) + \int_{A_L} f(\mathbf{x}, \omega_o, \omega_i) L_e(\mathbf{y}, -\omega_i) V(\mathbf{x}, \mathbf{y}) G(\mathbf{x}, \mathbf{y}) \, \mathrm{d}A. \tag{2.44}$$

The above equation includes $L_e(\mathbf{x}, \omega_o)$ at the points considered, to include light sources to the set of visible surfaces (shaded) in the rendering process. In the early years, the missing part of indirect light was replaced by a constant ambient term. The integral is commonly approximated by Monte Carlo integration, using samples $\mathbf{y}$ chosen on the light sources. For the special case of N_L punctual (point) lights, the above equation is further simplified to:

$$L(\mathbf{x}, \omega_o) = \sum_{j=1}^{N_L} f(\mathbf{x}, \omega_o, \omega_i) L_j V(\mathbf{x}, \mathbf{l}_j) \frac{(\mathbf{n}_\mathbf{x} \cdot \overrightarrow{\mathbf{xl}_j})}{||\mathbf{x} - \mathbf{l}_j||^2}, \tag{2.45}$$

where $\mathbf{l}_j$ is the position of the j-th light source, L_j its omnidirectional radiance, and $\overrightarrow{\mathbf{xl}_j}$ is the normalized direction vector from $\mathbf{x}$ to l_j.

2.3.2 Ambient Obscurance

Ambient obscurance [ZIK98] (AO) is an empirical illumination method that tries to approximate the amount of indirect light that reaches a point, without being blocked by the surrounding geometry of the point and without taking into account interreflections. Hence, the incident radiance from blocked

directions is zero. This produces darker results than a full indirect illumination simulation, but visually convincing under the assumption that the environment is lit by distant lighting, such as an overcast sky (see Figure 2.11). The ambient obscurance equation is:

$$AO(\mathbf{x}) = \frac{1}{\pi} \int_{\Omega} \rho(d(\mathbf{x}, \omega))(\mathbf{n}_{\mathbf{x}} \cdot \omega) \, \mathrm{d}\omega, \tag{2.46}$$

where Ω is the hemisphere centered at the normal vector $\mathbf{n}_{\mathbf{x}}$ of the receiving point $\mathbf{x}$. $\rho(d(\mathbf{x}, \omega))$ is an obscurance attenuation function of the distance $d(\mathbf{x}, \omega)$ of the closest point to $\mathbf{x}$ in the incident direction ω. When only visibility of the distant environment is considered, the distance attenuation function is replaced by a binary visibility term V and ambient obscurance becomes *ambient occlusion* [1]:

$$AO(\mathbf{x}) = \frac{1}{\pi} \int_{\Omega} V(\mathbf{x}, \omega))(\mathbf{n}_{\mathbf{x}} \cdot \omega) \, \mathrm{d}\omega. \tag{2.47}$$

For efficiency reasons, it is common to assume that the visibility/attenuation function simulates the cancellation of light attenuation up to a certain distance, i.e., on the geometry near $\mathbf{x}$ (see Figure 2.12).

2.3.3 Environment Illumination

An efficient way to capture the illumination objects receive from the surrounding environment is with the use of *environment maps*. An environment map encodes the illumination around a single point, which is a sufficient approximation if the environment is assumed to be infinitely far away from the objects to be illuminated. Mathematically, the simplification here relates to the incoming illumination, which depends only on direction. This technique has been used both for specular and diffuse reflections, so the light paths that can be simulated are L(D|S)⋆(D|S)E, L(D|S)⋆ being captured and provided by the environment map. Commonly, these maps are captured from real world conditions using a camera and a highly reflective sphere, called a *light probe*. This information is stored and indexed later, based on a lookup direction. In their most simple form, environment maps store $L_i(\omega)$ (note that $\mathbf{x}$ is dropped here for the lack of positional dependence). If ideally specular surfaces are considered, a single direction ω_i suffices for the plausible rendering of the material's appearance (*reflection maps*). For glossy surfaces, however, the full irradiance integral has to be evaluated (see Figure 2.13). Environment illumination is expressed mathematically as:

$$L_r(\mathbf{x}, \omega_o) = \int_{\Omega} f(\mathbf{x}, \omega_o, \omega_i) L_{map}(\omega_i)(\mathbf{n}_{\mathbf{x}} \cdot \omega) \, \mathrm{d}\omega_i, \tag{2.48}$$

where L_{map} is the environment map. It is many times convenient, for sake of computational efficiency, to preintegrate the incident radiance $L_{map}(\omega_i)$ for pure Lambertian BRDFs, and directly index the environment map, where these values are stored with respect to a representative incident direction ω_i. An extreme but very common case arises when the environment map stores the preintegrated result for Lambertian surfaces. Then, the BRDF $\rho(\mathbf{x})/\pi$ is moved outside the integral and not stored, while the remaining part represents the surface irradiance. Irradiance is estimated, stored in *irradiance environment maps* and post-multiplied with the BRDF after indexing it with the points's normal direction:

$$L_r(\mathbf{x}, \omega_o) = \frac{\rho(\mathbf{x})}{\pi} E_{map}(\mathbf{n}_{\mathbf{x}}). \tag{2.49}$$

[1] The acronym AO is often used to refer to both ambient obscurance and ambient occlusion in the literature, despite the difference in the visibility/attenuation term. While not strictly correct, we follow the same convention throughout this thesis as the majority of ambient obscurance/occlusion methods can be trivially converted between each other.

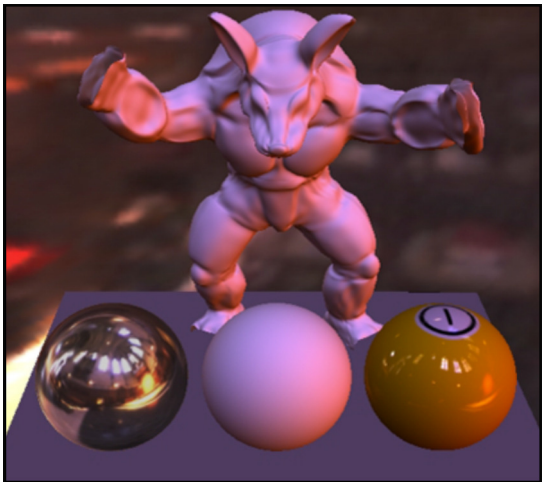


Figure 2.13: The effect of environment mapping on different surfaces. The armadillo, the sphere on the center and the table are diffuse surfaces lit from irradiance environment maps. The environment present in the specular highlights of the left and right reflective spheres is computed with reflection mapping. (Image source: [RH01])

Figure 2.14: Simulation of caustics cast from a wine glass on the wooden floor. Note the circular bright highlights, appearing both at the base of the glass and at its shadow, due to many light paths being reflected and refracted at the wine glass and eventually converging at the same location on the wooden floor. (Image source: [Tob])

2.3.4 Caustics

Caustics appear due to the concentration of multiple reflective or refractive rays onto a diffuse surface (LS+DE). These are most frequently observed through a curved glass of a transparent liquid (see Figure 2.14), on surfaces underneath shallow waters or as light is reflected on waves. Accurate simulation of caustics at interactive times is a demanding task, since a large number of photons is concentrated only on small areas of the environment. These algorithms usually require multiple passes. One pass is required for tracing and storing photons as they interact with reflective and refractive surfaces until they reach a diffuse surface. A second pass then uses the stored information from the first pass to illuminate the appropriate surfaces.

2.3.5 Diffuse Interreflections

A large category of algorithms have focused on simulating global illumination effects based on diffuse surfaces. Typical physical materials exhibit a significant amount of body reflection (diffuse scattering), which due to its omni-directional distribution, provides significant energy to nearby geometry. Therefore, diffuse indirect lighting contributes drastically to the overall appearance of a scene (see Figure 2.15). The simplified version of the rendering equation for diffuse illumination has been extensively used, especially in interactive applications. This is due to the fact that the ideal diffuse BRDF is constant, the diffuse indirect lighting varies slowly across the environment (low-frequency signal) and is therefore more efficient to compute. In terms of light paths, diffuse global illumination is LD⋆E. For diffuse surfaces with constant BRDF, the reflectance equation becomes:

$$L_r(\mathbf{x}, \omega_o) = \frac{\rho(\mathbf{x})}{\pi} \int_{\Omega} L_i(\mathbf{x}, \omega_i) \cos\theta_i \, \mathrm{d}\omega_i, \tag{2.50}$$

where $\rho(\mathbf{x})$ is the albedo of the surface. The majority of algorithms, regardless of their efficiency, can model multiple bounces of indirect diffuse illumination.

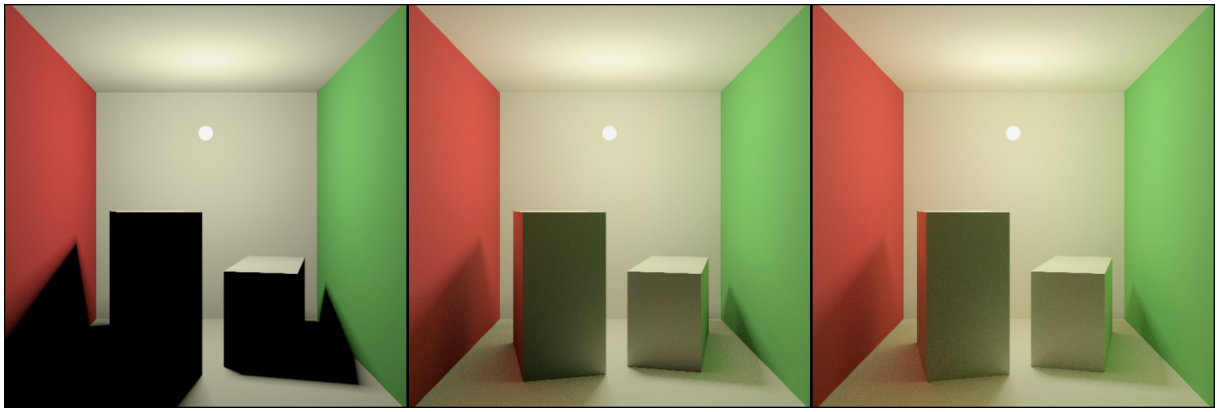


Figure 2.15: Left: Scene lit only with direct illumination. As in Figure 2.10, areas where light sources are unreachable directly, appear dark. Middle and right: Direct and one/two-bounce diffuse indirect illumination respectively. Notice the reflected colors of the red and green walls "bleeding" on the boxes. Image rendered with XEngine [VP], using the method presented in Chapter 7.

2.4 Common Global Illumination Algorithms

In this section, we discuss the most popular algorithms for approaching the light transport problem. These methods attempt to either solve the rendering equation entirely, by considering all possible light paths, or special cases of it, as discussed in the previous section. We explain the basic idea behind each of these methods and discuss some of their related work.

Note that we do not focus on interactive methods in this section, but rather on the original approaches that have significantly influenced the field of computer graphics, in general. As such, almost all methods in interactive rendering are based on the approaches discussed here. Prior work in interactive and real-time methods, which is the focus of this thesis, is provided in Chapter 3.

We begin our discussion in Section 2.4.1 with the radiosity methods, which perform particularly well on diffuse global illumination problems. In Section 2.4.2 we review the unbiased path tracing method. In Section 2.4.3 we explain photon mapping, the most efficient algorithm for the simulation of caustics. In Section 2.4.4 we discuss instant radiosity, which computes global illumination using a set of virtual point lights and, finally, in Section 2.4.5 we review point-based global illumination, a very interesting technique which approximates the rendering equation based on point clouds.

2.4.1 Finite Elements

Finite Element methods, commonly called *radiosity* methods [Gor+84; CG85; NN85], have been considered one of the most popular approaches to solve the light transport problem for diffuse surfaces ($LD\star E$), since their introduction to computer graphics in the 80's. Readers are referred to the works of [CWH93; SP94; Dut+06] for further information. These methods are based on radiative heat transfer [Wie66; HW91] which describes the energy exchange between surfaces when they have been thermally excited. The main idea in radiosity algorithms is based on the observation that ideal diffuse illumination is of low frequency and, as such, varies slowly amongst surfaces. So, the environment can be discretized into a set of finite elements, or *patches*. This way, the light transport problem can be solved by exchanging energy between them (see Figure 2.16).

On purely diffuse surfaces, radiosity and radiance are related as $B(\mathbf{x}) = \pi L_r(\mathbf{x})$ and $B_e(\mathbf{x}) = \pi L_e(\mathbf{x})$. Furthermore, the self-emitted radiance L_e and the BRDF do not depend on incoming or outgoing directions. Multiplying the surface integral form of the rendering equation (Equation 2.37) for diffuse BRDFs by π of the left- and right-hand side of the above equation yields the *radiosity integral equation*:

$$B(\mathbf{x}) = B_e(\mathbf{x}) + \frac{\rho(\mathbf{x})}{\pi} \int_A K(\mathbf{x},\mathbf{y}) B(\mathbf{y}) \, \mathrm{d}A_{\mathbf{y}}, \tag{2.51}$$

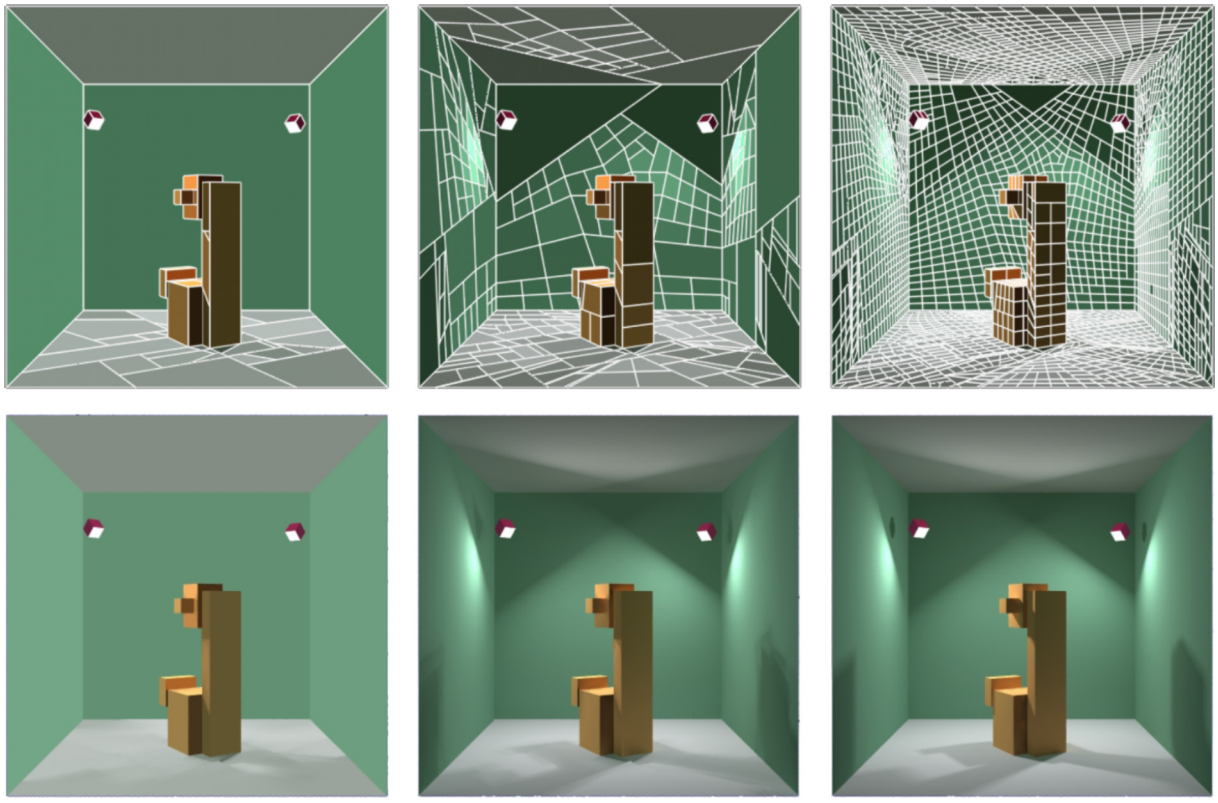


Figure 2.16: Radiosity approaches discretize the scene into patches and compute the indirect illumination based on the sparse computations on these patches. In this example, the scene is discretized hierarchically based on radiance discontinuities to provide more accurate results on shadow boundaries. (Image source: [LTG93])

in which the kernel $K(\mathbf{x}, \mathbf{y}) = G(\mathbf{x}, \mathbf{y})V(\mathbf{x}, \mathbf{y})$. Equation 2.51 can be converted to a linear system of equations (see [Dut+06] for a complete derivation):

$$B'_i = B_{ei} + \rho_i \sum_j F_{ij} B'_j, \tag{2.52}$$

where F_{ij} are the *patch-to-patch form factors*:

$$F_{ij} = \frac{1}{a_i} \int_{A_i} \int_{A_j} \frac{K(\mathbf{x}, \mathbf{y})}{\pi} \, \mathrm{d}A_{\mathbf{y}} \, \mathrm{d}A_{\mathbf{x}}. \tag{2.53}$$

The form factors represent the amount of energy transfer between two differential surface patches i and j based on their surface normals, per-point distance and mutual visibility (see Figure 2.17), and are non-trivial four-dimensional integrals. They are only dependent on the geometry of the scene, and not on any specific configuration of light sources in the scene. Once the form factors have been computed, the radiosity equation can be represented as a system of linear equations. The system could be solved numerically, using Gaussian elimination, Jacobi and Gauss-Siedel iteration [CG85] or using Southwell relaxation [GCS94]. The major bottleneck is usually the accurate computation of the form-factors, the efficient discretization of the environment, which presents visual artifacts due to discontinuities and the fact that the full matrix solution is computationally and memory intensive.

In the past, many improvements to the original algorithm have been proposed. These include alternative visibility, patch subdivision and representations and caching mechanisms, such as the Hemicube form factor representation [CG85], hierarchical radiosity [HSA91], discontinuity meshing [LTG92] and form factor computation via ray tracing [SP89; SS96]. Also support for non-diffuse phenomena has been proposed [ICG86] and alternative radiosity representations, such as wavelet projection [Gor+93].

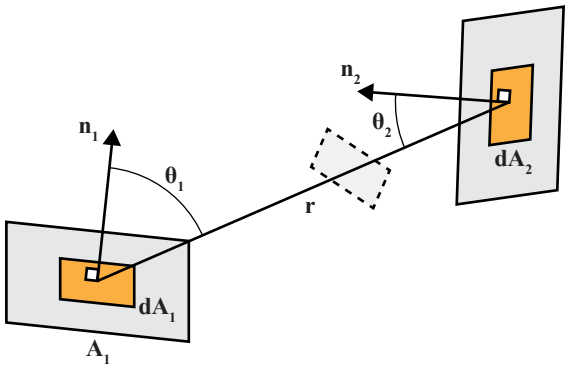


Figure 2.17: The form factors represent the fraction of energy leaving a differential patch $\mathrm{d}A_i$ towards a differential patch $\mathrm{d}A_j$. The amount of energy depends on their per-point distance, surface normals and mutual visibility, i.e., there is no energy exchange if a third patch lies between them (dashed patch).

2.4.2 Monte Carlo Methods and Path Tracing

The most popular methods for solving the rendering equation are based on Monte Carlo Integration approaches [Sob94]. As already discussed (see Section 2.1.1), the advantages of Monte Carlo-based algorithms are that they are general, unbiased and the number of samples does not depend on the integral dimensionality. Practically, this means that they can support all light paths, $L(D \mid S) \star E$ in path notation, and that as the number of samples increases, the estimated value converges to the correct solution.

In a typical Monte Carlo-based approach which estimates the rendering equation, the integrand is randomly sampled by recursively emitting (*tracing*) rays from the gathering point and collecting the radiance from the hit points or the environment. Historically, the idea of tracing rays was initially to resolve visibility for direct illumination by *ray casting*, i.e., sending rays from the camera until an intersection with an object was found [App68]. Later on, Whitted [Whi80] continued the process by spawning new rays at each intersection based on the material properties. Upon intersection, reflection, refraction and shadow (visibility) rays were generated recursively. This is commonly referred as *Whitted-style ray-tracing*. Cook et al. [CPC84] refined ray tracing to *distributed ray tracing* to account for fuzzy phenomena by sending multiple rays amongst various domains. This way, effects such as glossy reflections, motion blur and soft shadows were feasible. Kajiya [Kaj86] introduced the rendering equation and suggested Monte Carlo path tracing as a numerical solution to it, where samples were chosen probabilistically and weighted according to their expected contribution. Instead of following rays in the "reverse" way the light travels, i.e., starting at the camera and eventually hitting the light source (left part of Figure 2.18), *light tracing* [AC86; DLW93] methods start at the light source and follow rays until they reach the camera sensor (middle part of Figure 2.18). Path and light tracing perform better in different scenarios. In environments that contain caustics, specular reflections and have small light sources or are indoor scenes where the light contribution comes from small areas, light tracing is more suitable. Rendering scenes with large area sources and strong diffuse indirect illumination presence, converges faster using path tracing. Bidirectional path tracing [LW93; Vea98] unifies both unidirectional methods in a single bidirectional framework (right part of Figure 2.18). Metropolis Light Transport (MLT) [Vea98] is based on a Markov Chain Monte Carlo method and provides fast convergence in difficult scenes, such as environments that are illuminated from a small set of light paths (a small hole, a slightly open door). The algorithm works by searching for important light paths. Once such a path is found, the algorithm explores nearby paths based on mutation strategies to find other paths that have high contribution to the final image.

In practice, each of the above algorithms works best under different settings. Most of the subsequent work is either extending the aforementioned algorithms [KS01; Bou+13; Leh+13; KD13] or is using hybrid approaches based on Monte Carlo/Markov Chain Monte Carlo and photon mapping variants (see Section 2.4.3), such as [CTE05; Lai+07a; Geo+12; JM12; HPJ12]. Refer to Davidovic et al. [Dav+14] and the recent SIGGRAPH courses [Kři+13; Kři+14] for additional information.

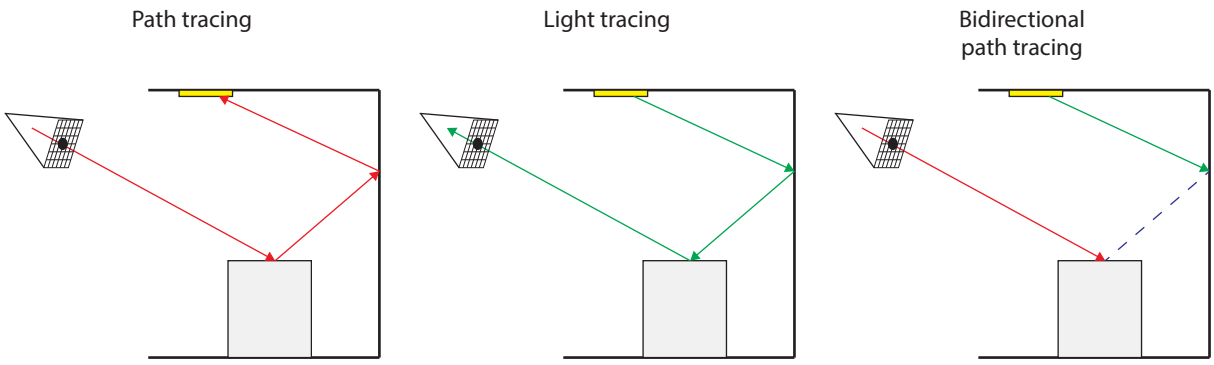


Figure 2.18: Simplified illustration of path tracing variants. Left: Unidirectional path tracing starts at the camera, eventually hitting the light source. Center: Unidirectional light tracing starts at the light source, eventually hitting the camera. Right: Bidirectional path tracing starts both at the camera and the light source and creates sub-paths for faster convergence.

2.4.3 Photon Mapping

Jensen et al. [Jen96; Jen01] introduced the very popular technique of *photon mapping*. This algorithm can simulate all global illumination phenomena, L (D| S)⋆ E in path notation, but is particularly good for handling specular effects, such as caustics and specular-diffuse-specular (SDS) paths.

The original technique operates in two steps. In the first step, emitted photons from the light sources are traced throughout the scene and stored in two photon maps represented by balanced kd-trees. The *caustics photon map* stores photons, which, after a specular event, have participated in a diffuse event (see Section 2.3.4). The *global photon map* stores photons that are bounced throughout the scene via all other combinations of events. In the second step, the scene is rendered using either accurately, through standard Monte Carlo ray tracing or approximately, by using a *radiance estimate*, which is simply the gathered radiance from k nearest photons in a given radius (see Figure 2.19). More precisely, the rendering equation integral is split into four integrals (direct illumination, specular reflection or transmission, caustics and diffuse indirect illumination), where each one is computed differently for convergence as well as for efficiency reasons. Direct illumination is estimated using light source sampling (see Equation 2.44). Specular reflection or refraction is computed separately by path tracing using BRDF importance sampling. Caustics are estimated using the radiance density estimate on the caustics map and finally, diffuse indirect illumination is computed by a gathering step in which rays are traced and the radiance density estimate at the hit points is integrated. The last step can be optimized for Lambertian surfaces using *irradiance caching* [WRC88; WH92].

Photon mapping is a *biased but consistent* method due to the density estimation step since the radiance

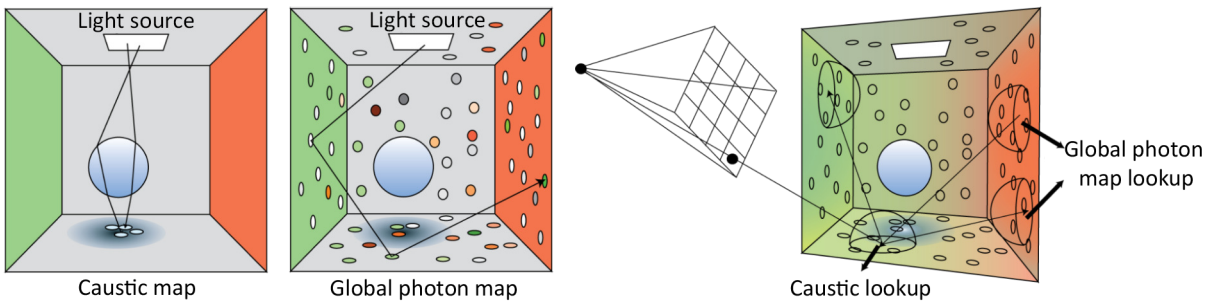


Figure 2.19: Photon mapping steps. In the first step, photons are traced and stored in a caustics map (left), which stores LS+D photons, and a global map (middle), which is used for indirect illumination. In the second step (right), surfaces are illuminated using radiance density estimation, either directly (caustics) or after a gathering step (other GI). (Image adapted from: [The+08])

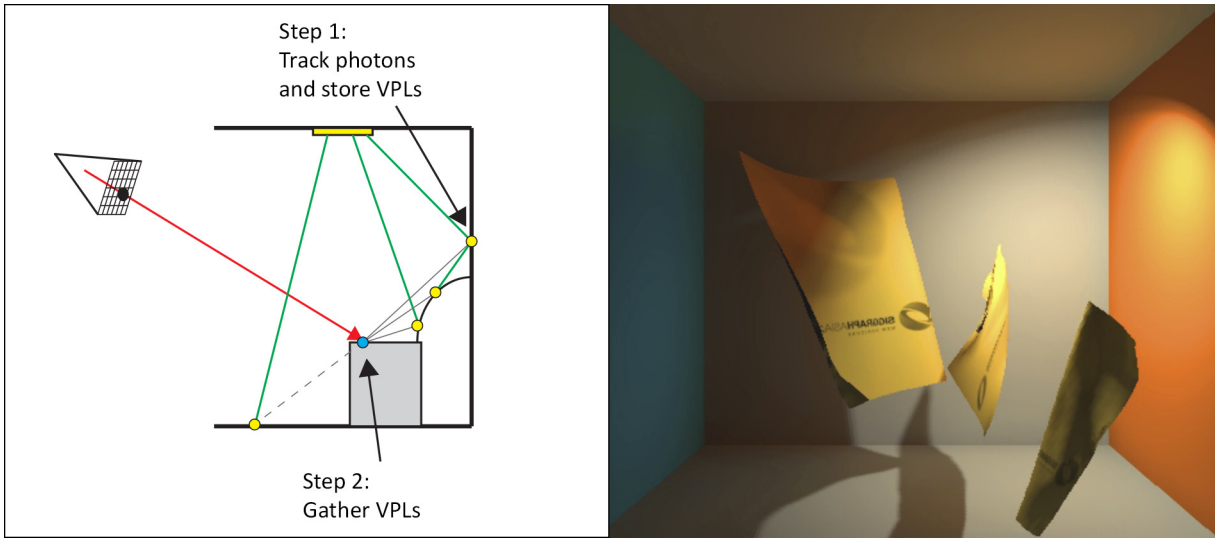


Figure 2.20: Left: Instant radiosity approaches work in two steps: Photons are traced and VPLs are stored at hit locations (yellow points). Then, surfaces (blue point here) are illuminated from visible VPLs. Right: A scene rendered with instant radiosity. (Image source: [Rit+08])

density estimator uses flux samples in a wide area around the sampled location $\mathbf{x}$ instead of the point itself. An immediate side effect of this is that sharp illumination transitions are blurred, depending on the radius of the disk within which the k-nearest samples are found. Given an infinite number of photons, however, the algorithm converges to the correct solution. Therefore, the quality of photon mapping is limited by the number of photons that can be stored in the photon map. Since this section covers the main photon mapping approaches, the readers are referred to the recent SIGGRAPH course by Hachisuka et al. [Hac+12] for more information.

2.4.4 Instant Radiosity

Instant radiosity by Keller [Kel97] introduced a whole family of algorithms that attempt to approximate the indirect illumination in an environment by replacing light bounces with direct illumination from a set of Virtual Point Lights (VPLs - see Figure 2.20).

The original CPU-based algorithm operated in two steps. In the first step, paths based on random walks are constructed from the light sources and photons are emitted and traced in the environment. At each intersection with the geometry, a VPL is positioned representing the outgoing flux at that point. In the second step, the scene is lit from the VPLs using standard forward visibility determination techniques (see shadow maps in Section 2.6.1). Each shaded point is checked against each VPL as if that VPL is an actual light source. As a result, the irradiance integral operation is replaced by direct lighting from these VPLs. The disadvantage of instant radiosity methods is the large number of VPLs required to converge to a correct solution and the introduction of bias in the energy exchange calculations. In the first case, the large number of VPLs also increase the number of shadow queries required for the illumination calculations and in the second case, the energy exchange between VPLs or between VPLs and points of interest require division by their squared distance, which for near points can produce singularities manifested as over-bright spots.

2.4.5 Point-based Global Illumination

Point-based global illumination is a popular technique that has been used extensively in film production (e.g., in Pixar's Renderman) [Chr08; Chr10], but has also attracted research interest in interactive rendering [Rit+09; Hol+11; MW11]. The original algorithm proceeds in three steps to compute one-bounce indirect illumination. In the first step, the direct illumination receivers are approximated as a

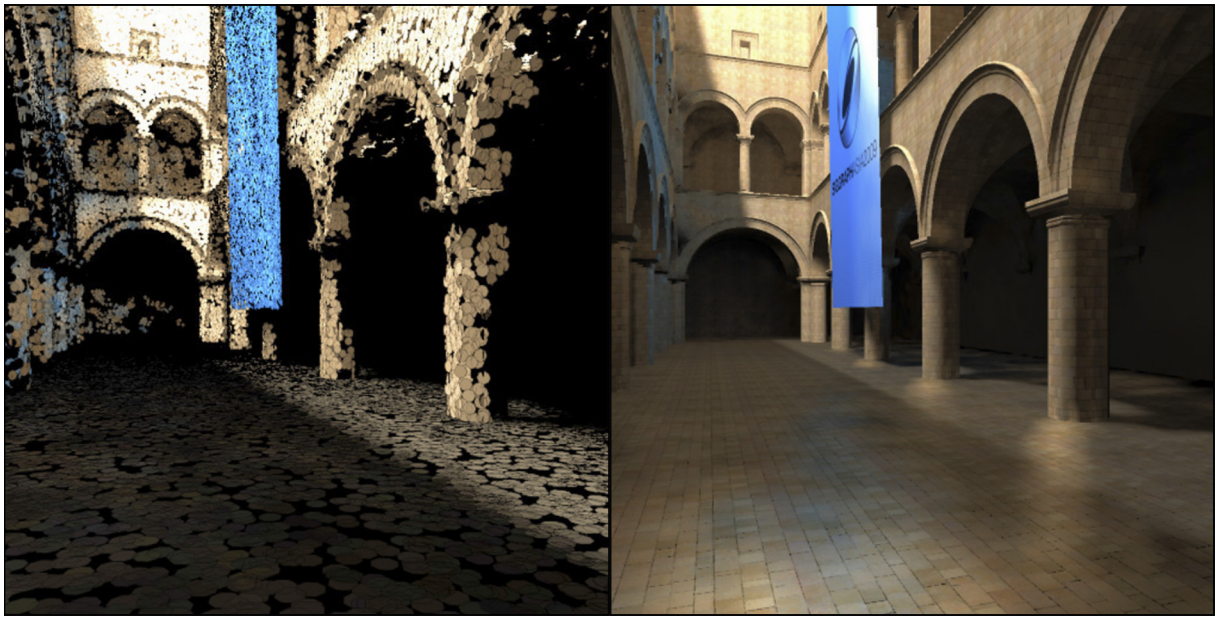


Figure 2.21: Point-based global illumination techniques approximate the scene using a collection of discs (surfels). Information relevant to illumination, such as normal, reflectance coefficients, etc. is stored for each disk and injected in a hierarchical data structure, such as an octree. Each point is shaded based on a micro-buffer, which contains the rasterized surfels visible to that point by traversing the octree. Left: A scene approximated as surfels. Right: The final result. (Image source: [Rit+09])

densely populated point cloud where normal, direct illumination and radius information are stored with each point. The term *surface elements* or *surfels* is also used, since points are actually represented as disks of radius r [Bun05] (see Figure 2.21). In the second step, the surfels are structured hierarchically into an octree, storing the radiosity value at each surfel or cluster of surfels. Finally, coarse indirect illumination is estimated by creating a low resolution cube raster at the receiving point, where each fragment stores the incoming indirect illumination for each direction as seen from that receiver point. For this micro-buffer population, the surfels are rasterized recursively based on a distance-based metric between the receiving point and the surfels. Close surfels are looked up using ray casting, nearby surfels are rasterized individually and surfels located farther away are approximated using their cluster representation in the octree. The rendering equation for each point is then evaluated by convolving the BRDF with the incoming indirect illumination, where the classical ray tracing procedure is replaced with an iteration over the fragments in the micro-buffer. This algorithm can be also utilized to replace the expensive final gathering step in photon mapping based on ray tracing.

2.5 Color spaces

An important part of our work in Chapter 5 involves the exploitation of a subsampling scheme for the compression of functions in the spherical domain, borrowing ideas from the image compression field. In this section, we provide a basic overview on color spaces and the concept of chroma subsampling, as they are related to this thesis.

As already mentioned in our introduction to radiometry (see Section 2.2.2), only a tiny portion of the electromagnetic spectrum comprises what is perceived as light, which is roughly the range of 380 to 750 nanometers (see Figure 2.22). Each visible wave of light is typically represented as a *spectral power distribution* function (SPD). The SPD associates the various frequencies (or wavelength) contained within each light wave with respect to any radiometric or photometric quantity. The SPDs of light waves existing in nature are usually continuous, containing a large variation of different frequencies. The human visual system, however, interprets light as color based on the response curves of only three distinct photoreceptors,

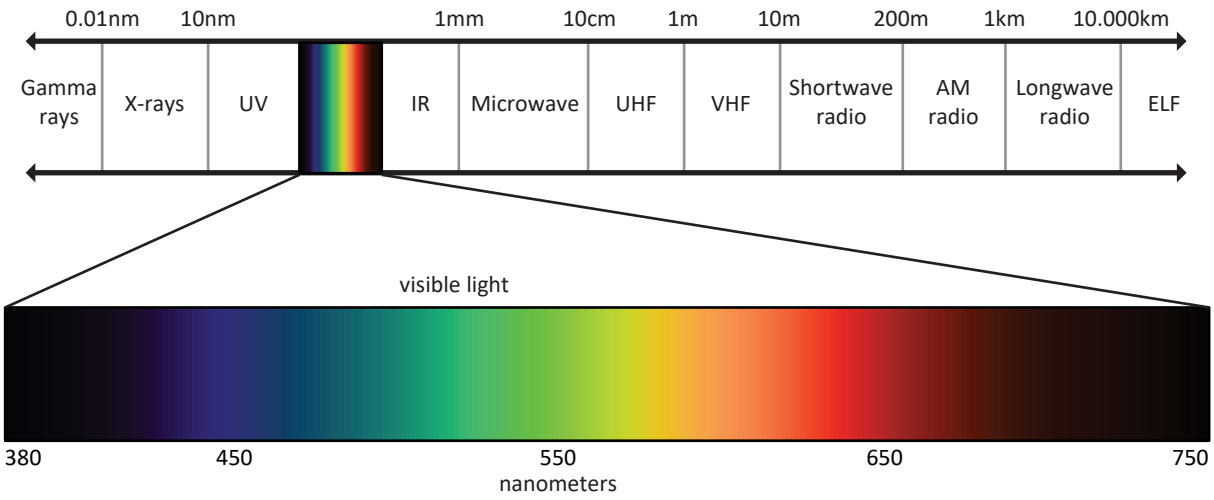


Figure 2.22: The visible light encompasses only a small part of the electromagnetic spectrum. (Image adapted from: [Hil+15])

essentially mapping the infinite-dimensional SPD to a three-dimensional one [Hil+15]. As such, the visible color spectrum can be represented as a combination of only three numerical values [Sto05]. The various ways light is perceived by the human brain, both perceptually as well as physically, are investigated by the field of color science, or *colorimetry*. Mathematically, colors can be represented in terms of a *color model*. This is commonly associated with a *color space*, i.e., the set of colors a model can reproduce.

The CIE XYZ, XYy and CIELAB color spaces are the most widely adopted systems used in order to simulate human vision. In these spaces, each color is essentially represented by a triplet of values. These generic spaces cannot be wholly reproduced by physical devices, and are considered more as mathematical constructions upon which other, more practical models, are defined. They are also used as a conversion intermediate between other color spaces.

Display devices are typically based on the RGB color model. This is an additive model, defined by a triplet of red, green and blue values, called primaries (see top row in Figure 2.23). These components are additively combined to create a large range of colors, according to the color space they are associated with, and the gamut, i.e., the range of colors, each device can represent. This model is considered additive since setting the components of these primaries to zero forms the color black, while setting the components to their maximum values forms the color white.

The CMYK color model is a four-tuple model, commonly employed by printing devices. This is a subtractive model; a combination of cyan, yellow and magenta are added on a white surface (such as paper), essentially masking out the reflected color. The choice of colors is, of course, not random. Adding cyan, yellow and magenta on a white surface absorbs the red, blue and green colors, respectively. The fourth component in the CMYK model refers to the color black. Even though black can be represented as a linear combination of the other three colors, for practical reasons, it is applied separately.

Several color models are based on hue/saturation components, such as the HSV (hue, saturation, value) and HSL (hue, saturation, lightness). Hue is associated with *pure* colors; different hues are lights at different wavelengths. Saturation represents how "clear" a color is, or how much its curve is spread out. The higher the saturation is the more pure the color appears. Finally, the last component is a representation of brightness. Since a color can be directly selected with respect to its hue, instead of a combination of three different colors, these models are considered more intuitive for artistic applications.

Finally, a family of color spaces is based on the decomposition of a signal into a *luminance* (Y) and two *chrominance* components, representing brightness and color information, respectively. Note that the terms *luma* (Y') and *chroma* are also used in the literature, sometimes erroneously, which are the non-linear (gamma-compressed) representations of luminance and chrominance. Historically, the YUV color space has been employed during analog television broadcasting in order to transmit both achromatic

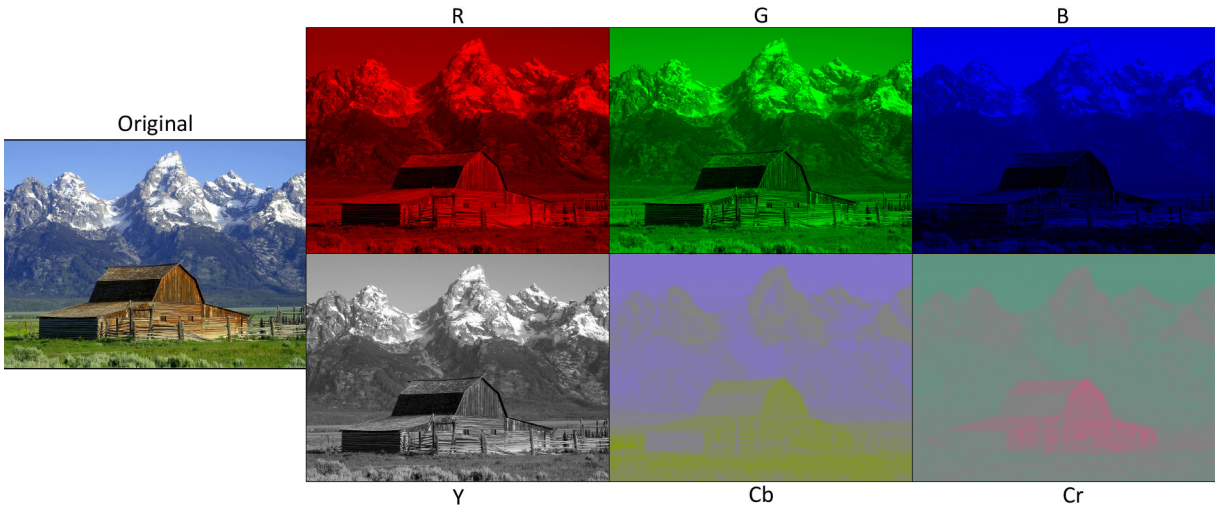


Figure 2.23: Expressing a signal in a luminance/chrominance based representation instead of RGB, allows for efficient compressions methods, expressing the chrominance channels in lower detail as the eye is more sensitive to spatial frequencies in intensity than in color. (Image adapted from: [Wikb])

and color images in black-and-white and color televisions simultaneously. In digital video and imaging, the YCbCr color spaces have been widely used in various compression algorithms, such as JPEG and MPEG. There, the observation that the human visual system's spatial sensitivity in luminance is higher than in color, has been exploited by encoding the chrominance components at a lower frequency compared to luminance, a technique called *chroma subsampling*. This is a much more efficient encoding scheme compared to an RGB representation, as the spatial details are preserved (see Figure 2.23). More recently, the better decorrelation properties of the YCoCg transform [MS03] compared to YCbCr have resulted in its adoption it in the H.264 standard for video compression, as well as in its integration in texture compression techniques for real-time rendering systems [VC07; MP12].

2.6 The Rasterization Pipeline

As already discussed briefly in the introductory chapter, the goal of any rendering algorithm is to generate synthetic images, accomplished through rendering pipelines. In this section, we focus on the rasterization pipeline, which is heavily used in real-time rendering and forms the basis of all the methods introduced in this thesis.

In general, the rasterization process receives as input a mathematical description of geometric shapes and converts it to a colored set of two-dimensional pixels, forming an image. On a conceptual level, this process can be easily pipelined into four basic steps. First, each geometric shape is decomposed into a set of basic primitives. Then, each object undergoes a series of transformations, eventually projected onto an image plane according to sensor specification, which can be viewed as placing a virtual camera (given a specific location and viewing direction) in the environment. Third, the projected objects are sampled, usually at a fixed spatial rate, to form tokens of geometric attributes called *fragments*, positioned at pixel locations within the image. Finally, each pixel is colored based on the sampled information.

Note that the term rasterization effectively describes only the conversion procedure (third step) of the aforementioned pipeline. Throughout the years, however, the rasterization process has resulted in expressing the entire pipeline, rather than an explicit part of it. So, these terms are used interchangeably in the literature to denote the complete rasterization pipeline instead of just the sampling process.

Modern Graphics Pipelines

Traditionally, the rasterization pipeline was implemented in a graphics processing unit (GPU) and all stages were *fixed*, i.e., they could only be programmed by modifying states in the underlying API. In its simplest form, it could efficiently compute direct illumination paths by operating in conjunction with the Z-buffer algorithm for visible surface determination.

Since then, however, GPUs have evolved to general purpose streaming processors, able to perform both the traditional graphics operations as well as to accelerate various other types of parallelizable computational problems. Consequently, the rasterization pipeline has also evolved from a strictly configurable to a partially programmable one.

Each programmable stage is executed through *shader* programs. Each shader is written in some high-level language, such as GLSL or HLSL, and can execute an arbitrary set of operations regardless of the stage involved. However, since programmable stages are executed in certain parts of the pipeline, each shader is also expected to conform with a particular set of input and output operations, associated with the stage it operates. For example, a vertex shader reads a vertex stream and writes the new vertex positions, usually after modifying them according to a linear and/or projective transformation. The geometry shader accepts as input an entire primitive, consisting of a set of vertices that define it, and can output zero or more new primitives. The fragment shader receives a generated fragment and can (optionally) write information to an arbitrary buffer. Shading operations are usually executed via the fragment shader and result in (at least) one color value being written to an image (frame) buffer.

While each shader stage has a specific set of required inputs and outputs, it can also perform random read/write memory access operations to arbitrary memory locations, accomplished via atomic operations. For example, a fragment shader can read/write to arbitrary pixel locations instead of only the one that a particular fragment invocation is associated with. In these cases, care must be taken to ensure that memory accesses are consistent; the program remains deterministic. This is often accomplished through manual synchronization procedures, such as setting memory barriers after atomic operations or shader stages. The exact behavior of these functions however, are implementation-dependent.

An example of the stages comprising a modern pipeline is shown in Figure 2.24. We now provide a more detailed discussion on the underlying stages, where we follow a naming convention similar to the OpenGL pipeline. However, the described process is rather generic and thus, applicable to other rasterization pipelines as well. For example, Direct3D follows a very similar approach [Gee08]. For a more thorough analysis of real-time graphics pipelines, the interested reader is referred to [AMH08].

Vertex Specification

Initially, a scene is decomposed to a set of tiny elements, called *primitives*. In a modern pipeline, these primitives are simple geometric shapes, such as points, lines and triangles. Each primitive is represented as an ordered sequence of *vertices*, where each vertex can be associated with an arbitrarily large set of *vertex attributes*. Usually, each attribute is defined uniquely per vertex. The minimum set of attributes includes the position vector of each particular vertex. In typical applications, the purpose of each attribute is to describe either a physical property (at a surface location) of each vertex, such as its position in some coordinate system, its normal vector and scattering coefficients, or any other arbitrary value that might be needed throughout the programmable stages of the pipeline.

The vertex specification is not a stage executed within the GPU. It can be thought of as a preparation stage, which submits a vertex data stream to the first stage of the pipeline.

Vertex Stage

This is the first crucial stage of the pipeline. This is a non-optional programmable stage and is executed through a *vertex shader*. The primary purpose of the vertex shader is to output a vertex position. In absence of any subsequent optional stages, such as the geometry or tessellation shaders, the expected behavior is to project the incoming three-dimensional coordinates of the vertex position to the corresponding

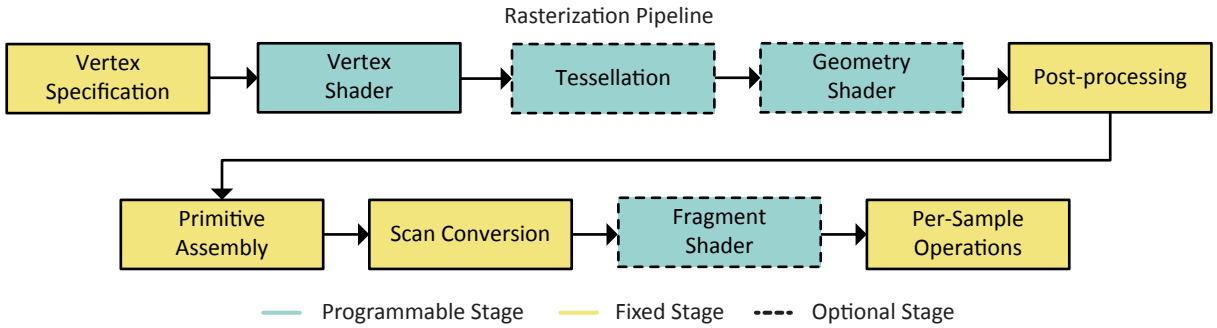


Figure 2.24: Simplified illustration of the OpenGL rasterization pipeline. Yellow and blue rectangles denote fixed and programmable stages respectively, while dotted rectangles represent optional stages of the pipeline. (Image adapted from: [Inc])

two-dimensional ones in the image plane. This is accomplished by a simple vector-matrix multiplication operation as the positions are usually defined in a different coordinate system, such as in object or world space.

Optionally, any other basic vertex operations can be performed at this stage depending on the application, such as transformations of other attributes to different coordinate spaces, displacement of vertex positions, etc. The output of these operations can be transferred to later stages of the pipeline.

An interesting observation regarding operations between different vertices is that they are independent of each other. The state of one vertex does not influence another. As such, the vertex stage can be highly parallelized by GPU hardware.

Tessellation Stage

In graphics, the primitives submitted for rendering can frequently represent curves or smooth surfaces via a piecewise linear approximation. The smaller (and denser) the primitives are, the finer the detail and quality of the shape is. However, a large number of primitives can impose both a vertex preparation and processing overhead as well as a bandwidth congestion between the host system and the GPU. To solve these issues, modern GPUs implement a programmable primitive *tessellation* stage that performs this "dicing" on the GPU device side. This means that coarse geometry is transmitted for rendering and storage in the GPU and tessellated on-the-fly (potentially in an adaptive manner) to better approximate the intended shape.

Currently, the tessellation process is divided onto three stages. In the first stage, a programmable shader is invoked to identify the degree of tessellation to be performed on each patch. Then, a fixed-function operation performs the actual subdivision, while the last stage, which is also programmable, computes the attribute interpolation data for the newly generated primitives.

A common example of applications involving tessellation shaders is the rendering of large terrains, which usually involves recursive surface subdivision in conjunction with displacement mapping [Coo84] to generate more complex environments. Other examples include hair rendering [YT10; And+16] and scene representations based on point clouds [NRS14].

Geometry Stage

As the optional tessellation stage is responsible for recursively subdividing a large number of patches, an optional geometry stage can be also employed to generate new primitives. This stage is also programmable via a *geometry shader*. Of course, low-quality subdivision can be also performed here. However, the intention of the geometry shader is to perform a different type of operations such as: (i) per-primitive calculations (such as the computation of per-triangle normals), and (ii) layered rendering. The latter is employed to perform multiple fragment shader invocations to write in a single rendering pass into more than one buffer locations, each one capturing information based on different camera configurations.

In this thesis, for example, the geometry stage is crucial for various parts of our algorithms: in Section 5.5 a geometry shader is invoked for the generation of a binary volume with increased sampling accuracy. In Chapter 6, we exploit layered rendering to render in multiple views in a single pass and in Chapter 7, we invoke a geometry shader both for multiview rendering as well as for the generation of an optional vertex buffer for ray tracing purposes.

Post-processing

This fixed stage denotes the end of any per-vertex operations. At this point, a clipping operation is performed in order to prepare primitives for the scan conversion process. Clipping is performed for optimization purposes as the subsequent stages can be computationally intensive. It ensures that primitives that lie entirely outside the viewing frustum are completely discarded, while primitives that are partially inside are clipped against the viewing frustum into new ones; omitting the outside part. This operation ensures that scan conversion will be performed on primitives that lie entirely within the view volume.

Alternatively, the subsequent stages can be discarded entirely. In that case, prior operations can be streamed on an output buffer, essentially exploiting the GPU as a general-purpose processor. For example, this is often exploited for the simulation of particle systems, such as fog, smoke and fire.

Primitive Assembly

This fixed stage serves as a preparation step for the subsequent scan-conversion operations. The functions of the Primitive Assembly stage are two. First, the incoming vertex stream is converted into a ordered sequence of basic, convex primitives. This primitive information is used, for example, to interpolate per-vertex attributes for each generated fragment.

Second, a primitive-based optimization called *face culling*, is performed. This technique switches off the rasterization of primitives that have a particular facing direction with respect to the viewer. A common application of face culling is associated with the rendering of closed, opaque surfaces, such as spheres, where primitives belonging at the back of the sphere can be safely discarded.

Note that this procedure is not always executed as a whole at this stage at this part of the pipeline. Its primary function is to convert vertices into a stream of primitives. Therefore, the first sub-stage can occur before invoking the geometry or tessellation shaders, if any of these optional per-primitive stages are enabled.

Scan Conversion

This is the most important stage of the rasterization pipeline. Here, each (clipped) primitive, represented up to this point as a fixed sequence of vertices, is converted to a series of *discrete* point samples, or *fragments*, on the image plane. Each generated fragment is associated with a particular pixel on the screen and contains the interpolated attributes of the primitive at its image-space pixel location.

In general, the conversion process generates a fragment only if a primitive overlaps a pixel according to a certain set of rules. These depend on the primitive type and are implementation-dependent. For example, during triangle rasterization, a fragment is generated only if a pixel location lies inside the projected triangle on the image plane. This can be easily determined by comparing each pixel position against three edge functions defining the boundaries of the triangle [Pin88]. Usually, this sampling position is located at the *center* of the pixel. Hence, each primitive is represented by one fragment per pixel.

The scan conversion is a discretization procedure. As such, it is a source of serious aliasing artifacts in the graphics pipeline. We discuss these in more detail in Section 2.6.2.

Fragment Stage

The last programmable stage in the pipeline is executed via a *fragment shader*. There, each rasterized primitive, now represented as an arbitrarily large set of fragments, is forwarded into the fragment stage, along with its associated data and interpolated vertex attributes. As the previous programmable stages, the

fragment stage is also executed in parallel. The most common operation of the fragment shader is to write depth and color values into one or more output locations. For example, per-pixel lighting computations are performed here. In case a fragment shader is associated only with a depth-buffer, a depth value is written implicitly. In this case, the fragment shader is empty and thus, this stage is considered optional.

Merging

The output of the fragment shader is not directly written to the output buffers. This is decided by a set of per-pixel operations in the final stage of the pipeline, the merging stage.

For example, *depth tests* are employed for visible surface determination. There, the incoming fragment values are written in the output locations only if the stored depth value is larger (farther with respect to the camera) than the incoming one. Some other examples of per-pixel operations include: *stencil tests*, where fragments can be discarded based on bitwise operations, *scissor tests*, where fragments are written only in a particular rectangular area of the screen, and *blending*, where the color values in the image buffer(s) is determined based on a preconfigured linear function instead of a simple replace operation.

Note that under certain conditions, some of the aforementioned operations can be performed prior to the fragment processing stage, thus saving computations.

2.6.1 Common Applications

As already briefly discussed, the rasterization pipeline is particularly efficient on resolving primary visibility and, consequently, computing direct illumination paths. We now review the most popular rendering techniques, as they are implemented within the rasterization pipeline.

Primary visibility determination

The Z-buffer algorithm [Cat74] forms the basis of the majority of techniques in the rasterization pipeline, and works as follows: Initially, an image buffer is created to hold the closest depth values with respect to the viewer, called the *depth buffer*. This is usually associated with one or more *color buffers*. The set of these image structures form the *frame buffer*. The depth buffer contains values in the range $[0, 1]$, where 0 and 1 indicate that a particular point is located on the near and far clipping planes respectively. At first, the depth buffer is initialized with ones, i.e., everything is located on the far plane. Each fragment, during the fragment stage, outputs one depth value and zero or more color values, to be eventually written on the frame buffer. At the merging stage, the depth value of each fragment undergoes a depth-testing operation against the currently stored depth value in the frame buffer at each pixel. If the depth of the incoming fragment is smaller than the stored one, this indicates that the fragment is closer to the viewer than the currently stored one, and its values replace the ones present in the frame buffer. Otherwise, the fragment is farther than the currently stored one and the incoming values are discarded.

The important observation here, is that a depth buffer stores only the closest depth values from a particular viewpoint. In other words, the depth buffer holds only a view-dependent *single layer* of information. This is a discretized representation of a subset of the scene's geometry that is particularly useful for simulating various effects in real-time rendering, as it allows to calculate and maintain in the GPU geometric information in a format (sample array) of predetermined size (and preallocated memory) that is easy and efficient to access from the programmable stages of the pipeline in subsequent rendering passes. In general, the techniques which are based on such a view-based and layered representation of the scene are said to be operating in *screen-space*, or *image-space*.

Multi-layer visibility determination

The Z-buffer can determine the correct color of each pixel as long as the corresponding geometry consists of opaque materials. Effects such as the rendering of transparent/translucent surfaces require either the fragment information to arrive in sorted back-to-front order and evaluated using a blending operation, or

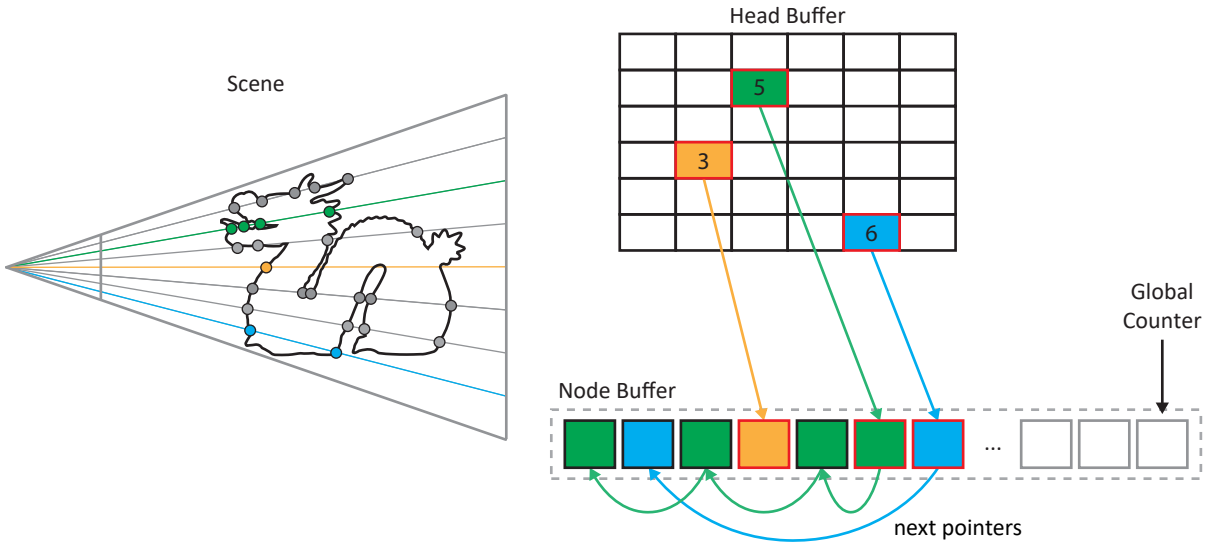


Figure 2.25: Multifragment rendering using per-pixel linked-lists. The address of the starting (head) node of each per-pixel list (red rectangles) is stored in the Head buffer, while the list connectivity and data information for all captured fragments is stored in the Node buffer. For each pixel, traversal is trivially accomplished by (i) fetching the head node and (ii) employing the next pointers stored within each node.

more complex data structures containing *multi-layer* or *multi-fragment* information, which extend the stored geometric representation to more than one layers (with respect to the camera viewpoint).

The most popular approach for multifragment rendering is based on the A-buffer [Car84]. The idea is simple: instead of storing only the closest visible fragment through the depth-buffer, a list of fragments is generated and stored in variable-length lists. In order to properly handle transparent surfaces, a Z-sorting operation arranges the fragment lists in a back-to-front sequence.

Nowadays, the same methodology is followed for order-independent transparency, by taking advantage of the capability of modern graphics hardware to perform atomic operations. In the rasterization pipeline, the most common algorithm is based on per-pixel linked-lists [Yan+10]. The idea is similar: during a rasterization pass, a linked-list is generated concurrently for each pixel. Since fragments are emitted (generated) in undetermined order, however, a sorting step is further employed to rearrange the fragments based on their depth, by modifying the link pointer contained within each node. The entire structure is maintained in two buffers: a *node* buffer storing the node connectivity and data information of all lists, and a *head* buffer containing an array of memory offsets of the node buffer, each pointing to the starting node location (head node) of each per-pixel list (see Figure 2.25). The algorithm operates in three steps. Initially, both buffers are allocated and cleared to zero, representing empty space and null pointers in the node and head buffer, respectively. This step is required as dynamic memory allocation is not supported within the rasterization pipeline, i.e., a memory block (such as the node buffer) cannot be reallocated within a fragment shader. In the second step, a geometry pass generates the linked-lists via atomic operations. During the fragment shader invocation, and for each incoming fragment (new node), the address of each new node in the node buffer is obtained by atomically incrementing a *global atomic counter*. An atomic exchange operation is then performed, which (i) links the pointer of the new node to the current head pointer and, (ii) replaces the head pointer with the new node's address. This operation effectively creates the linked-list in reverse, containing all per-pixel fragments in the form of an a unsorted sequence. In the final step, a Z-sorting operation is performed using a fragment shader. For each pixel, the linked-list is fetched, stored locally and reordered using any sorting algorithm, such as insertion sort. A further discussion on these methods is offered in Sections 3.1.1 and 3.2.2.

Direct Illumination

The computation of primary illumination paths can be trivially accomplished in the rasterization pipeline. In its most basic form, this lighting technique is performed through *forward rendering* where for each light source the entire scene is submitted to the graphics pipeline and shaded, typically during the fragment shader stage. Under the assumption of geometric optics that lights are additive, the contribution of each light source is accumulated to the frame buffer using a blending operation. Forward rendering is simple but can quickly become a computational bottleneck. A reason for this is due to depth-testing being performed after the expensive fragment shading stage.

While various optimizations exist, a more efficient approach is through deferring all lighting computations after primary visibility has been resolved. There, the scene is rendered once and apart from depth, various other surface properties are stored in color buffers, such as normal vectors, material attributes, etc. The aggregation of these attribute buffers that results from this scene rasterization pass is called the geometry buffer, or *G-buffer* [ST90]. Then, and for each light source, the scene is shaded based on the screen-space information stored in the G-buffer and the results are additively inserted to the frame buffer, using the same blending operation as before. Another important benefit of deferred rendering is that various post-processing effects can be performed using this single-layered screen-space information available on the G-buffer. The main disadvantage is increased memory cost and the inability to handle transparent surfaces. The latter requires order-dependent visibility and can be accomplished either through a separate forward rendering pass, or by exploiting multi-fragment approaches.

Global Illumination

Single-layer screen-space techniques can be efficiently employed to generated various effects, even global illumination phenomena. An efficient way to compute shadows is based on rasterizing the scene from the light source [Wil78]. All visible surfaces, as seen from the light's point of view, are stored in the depth map in a separate, preceding rendering pass. Any surface parts sampled and stored in this depth map, called the *shadow map*, represent surfaces that are not in shadow. The shadow map is used during rendering direct illumination effects in order to detect whether a visible surface sample from the camera's point of view is lit or not. This is accomplished by transforming each point to the light's post-projective space and testing it against the depth values stored in the shadow map.

Similarly, a G-buffer can be generated from the light source. From a global illumination perspective, primary visibility from the light source is essentially the first bounce of light. Therefore, a G-buffer from the light source has resolved visibility for the first indirect bounce, which can be exploited to perform single-bounce indirect illumination [DS05].

A typical multilayer extension of the G-buffer are the *deep G-buffers* [Mar+14]. There, instead of storing only color information in multiple layers, a G-buffer is generated per layer, consequently increasing the quality of the captured information. However, multifragment techniques can quickly become prohibitive for real-time purposes, since the total number of generated fragments depends both on the frame buffer resolution as well as the scene complexity.

More advanced techniques need global information with respect to the entire scene, regardless of the number of captured layers. For example, reflections and ambient occlusion (see Section 2.3.2) require the tracing of rays in the vicinity of a point of interest until the closest intersection is found. For real-time purposes, these phenomena are approximated by ray marching in screen-space, i.e., iteratively moving in object- or pixel-space increments until a ray-fragment collision is detected based on the information stored in the G-buffer (see Section 3.2.5). In general, screen-space global illumination methods can comprise a visually plausible approximation of the desired phenomena, as long as the needed global information is restricted to small-range effects, as their view-dependent nature can cause visual instabilities. These methods are discussed in Section 3.2.2. Alternatively, large-scale effects are usually accomplished by data structures which capture the entire scene, such as regular grids. There, a high-detailed, but view-dependent discretization (image-space), is replaced with a crude, but view-independent approximation of the scene instead. These *volume-based* methods are usually combined with some form of (irradiance or radiance)

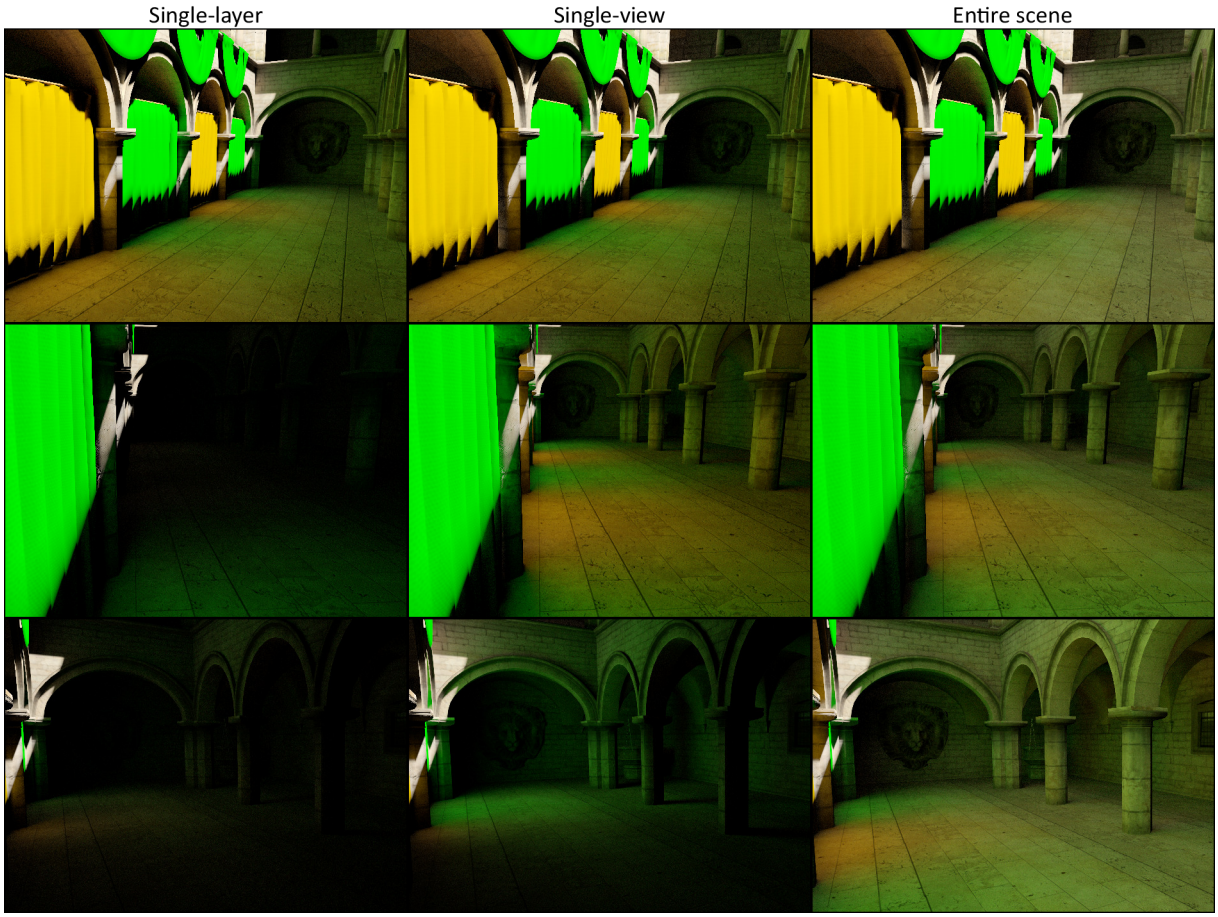


Figure 2.26: Single-bounce global illumination rendered using a small distant area light under three different viewpoints. Top row: A single-layer representation (left column) can provide acceptable results as long as the viewpoint contains adequate coverage of both geometric information and direct light energy. Middle row: In cases where the indirect light depends on energy gathered from nested layers (the non-visible yellow and green curtains) the entire view must be considered (middle column). Bottom row: In cases when the direct light energy is dominant outside the viewing frustum, the entire scene information must be considered in order to properly capture the indirect light (right column). Image rendered with XEngine [VP], using the method presented in Chapter 6.

caching, which can efficiently compute global illumination effects, as long as the represented phenomena are of relatively low frequency, e.g., the scene consists of diffuse or glossy surfaces. A discussion on these methods is offered in Section 3.2.3.

2.6.2 General Discussion

Regarding rasterization, the efficient scan conversion procedure imposes some very severe limitations in the graphics pipeline. The process of rasterization *approximates* the underlying geometry; a continuous primitive is represented by discretized samples at specific (usually regular) locations on the image plane. This results in three main sources of spatial aliasing: (i) polygon edges as well as (ii) oblique geometry are sparsely sampled, while (iii) geometry parallel to the view is entirely skipped. To mitigate (i) and (ii), the number of samples can be increased through various *multi-sampling* algorithms [AMH08]. However, this is a sampling problem involving the reconstruction of discontinuous functions, such as polygon edges, and according to sampling theory aliasing in these cases is inevitable [PH04; Mav13].

An alternative approach is conservative rasterization [HAO05], which guarantees that a fragment will always be generated as long as there exists some partial coverage between a pixel and a primitive. The

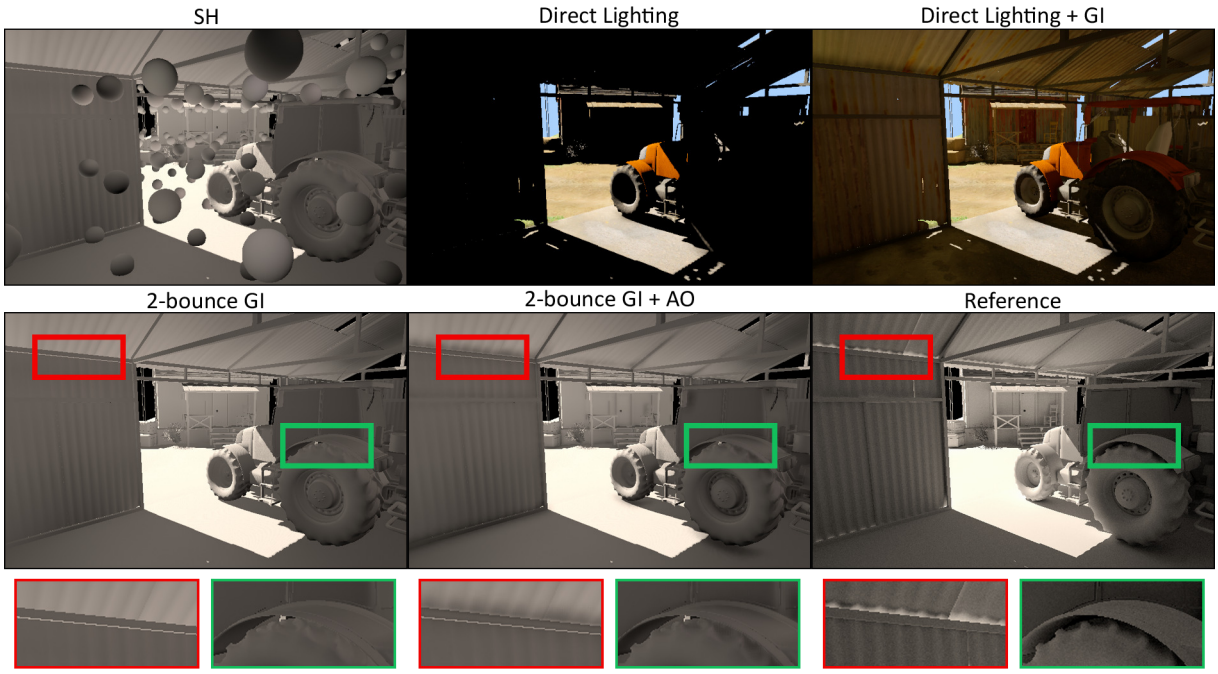


Figure 2.27: 2-indirect volume-based global illumination, where energy exchange occurs between representatives of the high-detailed geometry (top left). While the indirect light contribution of these methods is adequate for real-time purposes (top middle/right), the energy contribution of small-scale geometry cannot be properly captured (green and red rectangles in middle and bottom rows). To mitigate this, volume-based GI methods are usually complemented with AO techniques (middle row/column). However, the "light leaking" cannot be fully remedied. The correct solution is demonstrated in the reference ray-traced image (middle right). For clarity purposes, only the lighting buffer is shown in the majority of the images, i.e., stripped of any textures and albedo is set to 1. The real-time images have been rendered with XEngine [VP], using the method presented in Chapter 5.

problems with this technique are two: (i) zero area projected primitives will not generate fragments, and (ii) the sampling location might lie outside the primitive, causing attribute extrapolation problems, i.e., the fragment's attributes will be the result of interpolation with a location outside the primitive's extents. As such, conservative rasterization is usually employed for visibility algorithms rather than illumination problems.

A second important problem lies in the very heart of screen-space techniques. Their results are always based on both a partial (layered) and view-restricted representation of the environment, regardless of the quality of the operating algorithm. This, in conjunction with the sampling limitations mentioned before, will always result in view-dependent, approximated results (see Figure 2.26). On the other hand, by employing a volume-based data structure, important contact details are always missed (see Figure 2.27).

It is evident that in order to improve the quality of the rasterization pipeline, we must also improve the quality of the algorithmic input. Unfortunately, in order to produce results similar, or even identical, to the ray tracing pipeline we would require modifications on the construction procedure of the underlying data structure on a very basic level. Given the constraints of real-time applications, this is not a viable option and approximations are unavoidable. However, a major benefit in image space techniques can result by elegantly increasing the number of layers (up to a limited extent), the generated views, or a combination of both. On the other hand, volume-based methods can be improved by more careful and efficient representations. For interactive applications, however, the time constraints are more relaxed and therefore, these kind of modifications are more possible. An investigation on the aforementioned problems was the main motivational factor behind our research.

Chapter 3

Related Work

3.1 Data Structures

In this section we discuss the various data representations that can be employed as the basis for the techniques presented in Section 3.2. Note that while this thesis focuses on data structures that are directly related to the rasterization pipeline, which are discussed in Section 3.1.1, we also provide for completeness, an overview on spatial data structures commonly employed in CPU and GPGPU architectures in Section 3.1.2.

3.1.1 Image-based Data Structures

As discussed in the previous chapter (see Section 2.6.1), screen-space techniques rely on the use of view-dependent information, where calculations are performed on a discretized subset of the geometry (i.e., discrete samples with respect to the viewing frustum) instead of the entire scene.

In their simplest form, the data structures employed in image-based methods store only a single-layer of depth information, with respect to a particular viewpoint. Due to their efficiency, the most basic algorithms on the rasterization pipeline have been based on these data structures in order to perform primary visibility determination [Cat74] and shadow mapping [Wil78]. Extending the Z-buffer to a G-buffer, where color, normal and other screen-space information can be stored along with depth, has resulted in the development of screen-space variants of various global effects, such as ambient occlusion [McG+11], directional occlusion [RGS09] and single-bounce indirect illumination [DS05].

Multi-layer rendering techniques extend the stored information to more than one layer, resulting in fewer view-dependencies and higher quality renderings. However, this comes at the cost of increased computational complexity and memory requirements. Capturing all generated fragments in a single geometry pass is accomplished with memory unbounded algorithms, such as the A-buffer [Car84]. The original method is based on variable-length lists and correct depth order is ensured by a Z-sorting operation. Yang et al. [Yan+10] implemented a per-pixel linked list variant of the original method in the GPU, by exploiting the capability of graphics hardware to perform atomic operations (see Figure 2.25). The initial implementation suffered from performance issues due to thread contention caused by the shared counter attempting to access the same location in memory from multiple threads, an issue which has been resolved with the integration of the L2 memory cache in later generations of GPU hardware.

Instead of a linked-list, Liu et al. [Liu+10] proposed FreePipe, a CUDA-based implementation of the rasterization pipeline. Their approach allowed the storage of all fragments in fixed-size per-pixel arrays. Compared to a linked-list, the need to maintain node pointers is removed, offering much higher efficiency in complex scenes. However, the size of each array is fixed for all pixels, leading to excessive memory requirements where large blocks of memory space are allocated and never used. A similar approach was also implemented in the rasterization pipeline [Cra10].

Combining the advantages of fixed-size arrays and linked-list structures, memory friendly variants have also been proposed [Mau+12; VF12]. These methods allocate per-pixel information in continuous

regions and without overestimating memory space. However, they require additional passes for fragment accumulation and the generation of per-pixel memory offsets via a prefix sum operation.

A major bottleneck of GPU-variants is that order-dependent operations, such as the blending of transparent objects, require a post-sorting step. Typically, sorting occurs in a fragment shader by rendering a full screen quad in an additional pass. The fragment data is read from global memory, sorted locally and written back. Sorting can be very expensive for high-depth complexity scenes [KLZ12], due to the high latency of global memory, the low percentage of active thread utilization (occupancy) due to a large amount of resources allocated in local memory, which also increases latency, or even the sorting algorithm itself. Knowles et al. [KLZ13] proposed the grouping of pixels into powers of two based on their complexity, in order to increase occupancy. Each group was processed and sorted separately using precompiled shaders with different, but constant, local array sizes. Knowles et al. [KLZ14] combined the previous approach with a register-based scheme, where sorting was enforced in registers rather than local memory, for pixels of both shallow and deep complexity. The sorting algorithm could be modified based on the depth complexity, as it has been shown to provide significant performance benefits [KLZ12]. Vasilakis et al. [VF13] exploited the bucketed depth-peeling of Liu et al. [Liu+09a] to generate multiple per-pixel linked-lists, improving the overall performance of the post-sorting operation.

Memory-bounded algorithms also exist in the literature, keeping the k-closest generated fragments (with respect to the camera viewpoint) instead of the entire information. As such, memory overflows and performance bottlenecks in the post-sorting operation are avoided. On the downside, they require additional rendering passes and, typically, impose increased computational cost to ensure correct k-fragment ordering. *Depth-peeling* approaches have been traditionally employed to resolve visibility of transparent surfaces, operating under multiple geometry passes. Front-to-back depth peeling [Eve01] captures, or "peels", one layer per geometry pass in an iterative manner. The first pass operates identical to the Z-buffer, storing only directly visible surfaces. In each subsequent pass, the already stored fragments are discarded based on the depth value of the previously computed layer and the next nearest surfaces are stored. Depth-peeling results in a sorted sequence of fragments, eliminating the need for a post-sorting operation. Performance, however, can degrade quickly for complex environments, as k passes are required in order to extract the k^{th} closest fragments. Dual depth peeling [BM08b] reduces the amount of rendering passes in half by applying front-to-back and back-to-front depth peeling simultaneously, extracting both the nearest and furthest k^{th} fragments in each pass. Liu et al. [Liu+09a] concurrently perform depth-peeling into n subintervals by exploiting depth subdivisions and multiple render targets, reducing the total runtime of prior approaches. Single-pass depth peeling variants have also been proposed, but either require a CUDA implementation [Liu+09b] or limit the stored information to two layers only [Mar+14]. Multiple geometry passes are computationally intensive in real-time rendering of complex virtual worlds. Therefore, the problem of single-pass order-correct capturing of k-closest fragments in the rasterization pipeline has received important interest from the real-time research community. These approaches are generally based on the idea of the *k-buffer* [Bav+07]. The original method was able to properly capture the k-nearest fragments, but required modifications to the hardware pipeline in order to avoid read-modify-write hazards. To this end, various *k*-buffer variants have been proposed in the literature, exploiting stencil routing operations [BM08a], hardware-dependent spin-lock mechanisms [VF14], or even employing both a k- and an A-buffer for the purposes of hair rendering [Yu+12]. We refer readers to the recent work of Vasilakis et al. [VPF15] for a more general overview in *k-buffer* rendering.

3.1.2 Spatial Data Structures

The research community has extensively focused on the development of efficient spatial-based acceleration data structures in order to improve both the construction and the traversal performance of ray tracing algorithms. Since this is a vast field of research, we only describe previous work relevant to this thesis, i.e., on interactive ray tracing of dynamic environments. For more information, we refer the interested reader to the excellent book by Cormen et al. [Cor+09], the overview by Vinkler et al. [VHB15] and, finally, a comprehensive survey on animated scenes by Wald et al. [Wal+09], which sketches the general problem as well as covers the competing goals that even a present ray tracer has to face.

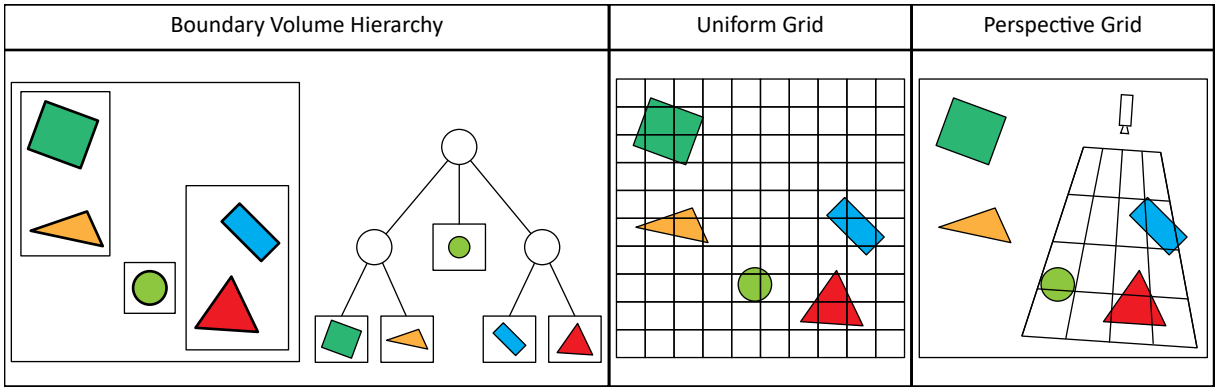


Figure 3.1: Illustration of spatial data structures. Left: BVH represent regions of space containing geometry in the form of a tree structure. Middle: Spatial subdivision structures, such as uniform grids, partition the environment by employing regular subdivisions. Right: Perspective grids operate similar to uniform grids but subdivide the scene based on the viewing frustum, in order to increase coherence in primary visibility rays.

Spatial acceleration data structures provide an approximative primitive-prioritized ordering in order to optimize ray-object intersection queries. This is accomplished by exploiting object hierarchies, such as boundary volume representations [WHG84] (BVH) (left of Figure 3.1), spatial subdivisions, such as kd-trees [Ben75; Kap85; Zho+08; HKH11], uniform grids (middle of Figure 3.1), octrees [Gla84; LK10], or a combination of both [Hav00].

With non-interactive applications in mind, where geometric content is fixed or rarely modified, the choice of the acceleration data structure can be determined solely based on its traversal stage complexity since the resource-demanding construction stage can be performed either as an offline preprocessing step, or infrequently. To support dynamic geometry, various refitting and updating strategies have been proposed to accommodate new geometry into the previous frame's acceleration data structure. However, most of them suffer from excessive deterioration and require the animation sequence to be either restricted to coherent motion types and fixed connectivity or known a priori [YCM07; WBS07; Gar09; Kop+12; BM15]. The highly complex environments employed in modern applications, however, can pose significant constraints on the acceleration structures since geometry can be potentially modified (tessellated or deformed) in an unexpected way in every frame. As such, the support of real-time animations dynamically generated from interactive manipulation tools, streamed from a shared distributed virtual environment or even procedurally generated, boils down to the simple solution of completely rebuilding the acceleration structure in every frame [Kar12]. The massive amount of research effort that has been devoted in the field of ray tracing, however, has resulted in an impressive performance speedup of the construction stage of recent approaches, with the most popular being the hierarchical linear BVH (HLBVH) [PL10; GPM11] and the treelet BVH (TRBVH) [KA13], both implemented on the GPGPU OptiX ray tracing engine [Par+10b]. Still, the construction process remains computationally intensive and most likely will remain so, as the increase in computational power consequently increases the desire for even larger and arbitrarily complex dynamic virtual worlds.

Directly applicable to interactive applications, uniform grids [Gla84] are able to achieve low construction times through regular subdivision of the environment (middle of Figure 3.1). While easy to implement, the lack of empty space skipping, which is available in hierarchical data structures, can significantly increase traversal times, especially for incoherent rays. Therefore, they are mainly applicable to scenes with uniform distribution of primitives, where the probability of early ray termination as well as ray coherence is increased. However, traversal performance in environments containing moderately non-uniform geometry distribution can be increased by employing a two-level level adaptive subdivision [KBS11]. Finally, the use of perspective grids, i.e., a perspective regular subdivision with respect to a particular viewpoint (right of Figure 3.1), can improve coherence on primary as well as on shadow rays [GN12].

It should be noted that the main advantages of regular grids are their efficient construction and their geometry-independent nature. As such, these data representations are not only restricted to algorithms related to the ray tracing pipeline. They are also highly applicable to the rasterization pipeline and real-time rendering, where they form the basis of volume-based illumination methods. These methods are discussed in the next section.

3.2 Illumination Techniques

This section describes previous work on real-time and interactive global illumination techniques. These approaches attempt to provide efficient, albeit approximate, solutions to the light transport problem, computed in either a few milliseconds or less than 1 second, based on the theory discussed in Section 2.2 and on the methods and algorithms presented in Sections 2.3 and 2.4. Interactive illumination is an elusive goal, since solving the rendering equation even without the presence of participating media is inherently complex, not only interactively, but also for offline renderers. As already explained in the previous chapters, there are many different parameters that affect both the accuracy of the result and the time constraints: the number and type of light sources, the diversity of the light paths (which depends on the complexity of the virtual world and the underlying materials) and the need to support dynamic, large environments. Therefore, a solution supporting fully dynamic scenarios would need to approximate the rendering equation continually, without any precomputations. As a result, interactive algorithms make simplifying assumptions to several of the above parameters and usually aim at producing a plausible result instead of an accurate one.

We start our discussion on the empirical ambient occlusion method presented in Section 2.3.2, both on image- and object-space techniques. This previous work is directly related to the method discussed in Chapter 4. The following sections focus on more complete solutions to global illumination, related both to real-time and interactive rendering. Section 3.2.2 discusses methods operating in image-space, which are mainly based in the idea of Instant Radiosity (see Section 2.4.4), and Section 3.2.3 focuses on illumination techniques operating in volume-space. These methods essentially lay the ground for our work presented in Chapter 5. Finally, Sections 3.2.4 and 3.2.5 discuss prior work in interactive ray tracing with respect to the rasterization pipeline, providing a background to the methods presented in Chapters 6 and 7.

3.2.1 Ambient Occlusion

The non-physically-based model of ambient occlusion attempts to simulate the "openness" of the environment above a shaded point (see Section 2.3.2). In general, this is accomplished by exploiting sampling techniques to estimate the visibility of a point of interest with respect to its surrounding geometry.

AO methods are distinguished between object-space and image-space ones. Object-space methods produce high quality, stable results, but depend on the scene complexity and are typically more expensive to compute. Image-space methods produce reasonably convincing results and offer bounded rendering times as they do not depend on scene complexity. They use information stored in the camera *G-buffer*, a series of textures containing information (depth, normal, reflectivity, etc.) from a specific viewpoint, as sampled for image generation. These methods can be executed in real-time and support fully dynamic environments, but are prone to undersampling, view-dependent artifacts and accuracy limited to near-field geometry.

3.2.1.1 Object-space AO

The early work on accessibility shading by Miller [Mil94] and the introduction of ambient obscurance by Zhukov et al. [ZIK98] were the key motivations for this area of research.

Bunnell [Bun05] proposed a two-pass method to calculate per-vertex AO for dynamic scenes by modeling the geometry as a collection of discs centered at point samples (surfels). The amount of shadowing between each pair of disks is computed analytically. The discs are stored in a hierarchical structure in order to support large environments, efficiently.

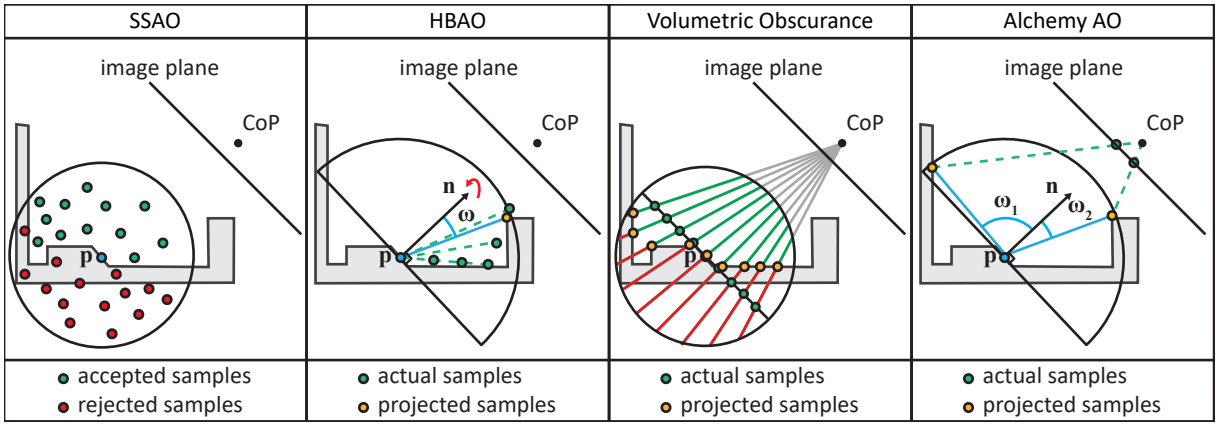


Figure 3.2: Illustration of some popular screen-space AO techniques. Left: SSAO estimates the occlusion based on random point sampling and visibility tests. Middle Left: HBAO uses a solid angle estimation by first sampling along a circular sweep in the azimuthal direction (red arrow) and then by linearly marching along the heightfield stored in the depth buffer. Middle-right: VO estimates occlusion by selecting samples on a unit disc and measuring the exact distance between the projected samples and the hemisphere. Right: Alchemy AO employs a solid angle estimation of the occlusion integral, similar to HBAO, but samples the domain of integration based on an image-space unit disc, similar to VO.

Hoberock et al. [HJ08] modified the original algorithm to remove disk-shaped and linear interpolation artifacts. Instead of disk elements, a number of methods [Ren+06; SA07; CK10] approximate the geometry as a collection of spheres, mainly for distant occluders.

Papaioannou et al. [PMP10] use a volumetric approximation of the scene geometry generated in real-time and perform ray-marching using the GPU. The accuracy of the method depends on the granularity of the volume grid, but it can also capture occlusion from semi-transparent surfaces as well, if a non-binary volume representation is utilized.

3.2.1.2 Image-space AO

Luft et al. [LCD06] simulated AO in image space by filtering the depth buffer with a high-pass kernel in order to increase visual contrast of surface discontinuities. Screen-Space Ambient Occlusion (SSAO) was introduced by Mittring [Mit07]; point samples are chosen from within a sphere centered at the shaded fragment and tested against the depth buffer. The ambient occlusion is approximated using the weighted result of the depth tests to estimate the open hemispherical volume above the shaded point. In this method, hidden samples also contribute to the occlusion of the estimator.

The popularity of the original SSAO method has sparked an enormous amount of research interest, which has been extensively focused on reducing the undersampling artifacts of the image-space AO estimator. The concept of horizon mapping [Max86] was exploited by Bavoil et al. [BSD08] in Horizon-Based Ambient Occlusion (HBAO). They perform a circular sweep of the image space near a shaded point, by ray marching along the radial directions up to an r_{max} range. Each sample is tested against the depth buffer in order to find the highest elevation (horizon) in the point's tangent space in every search direction. The final set of horizon angles is then used to estimate the solid angle subtended by the horizon, which is considered to contribute to ambient occlusion.

Loos and Sloan proposed Volumetric Obscurance (VO) [LS10], which improved the convergence of the open volume estimator of the SSAO technique. Samples are chosen uniformly on an image-space disk, projected on the depth buffer, and line segments are constructed by measuring the exact distance between the hemisphere and the projected samples. The occlusion above a point is then estimated by integrating over these line segments instead of traditional point sampling.

The Alchemy Ambient Obscurance algorithm by McGuire et al. [McG+11] combined the sampling scheme of Volumetric Obscurance with a statistical horizon-based approach to the open solid angle

estimation; samples in an image-space disc above the shaded point are projected to the depth buffer and a horizon is determined directly for each one of them according to their apparent elevation, thus dispensing with the radial samples of the HBAO algorithm. Finally, a user-adjustable attenuation factor based on the projected sample distance from the shaded point similar to Filion et al. [FM08] is applied to each partial obscurance estimate. This technique was further improved later on to efficiently capture occluders in a wider range, in a GPU-friendly and multi-scale fashion [MML12]. The aforementioned screen-space AO techniques are also illustrated in Figure 3.2.

Timonen [Tim13] proposed Screen-Space Far Field Ambient Obscurance, a method to sample the entire depth buffer without producing sampling errors due to far-field sampling.

Multi-layer/view Approaches

Despite the improvements in the AO estimator, view-dependent artifacts produced by single-layer image-space methods were still evident in all approaches (see Figure 4.2). The efficient injection of missing geometric information however, has received only a small amount of research interest, which we describe next.

Bavoil et al. [BS09] use several depth-peeled screen-space layers generated with an enlarged field of view for accessing visibility information behind the nearest depth layer and outside the screen bounds. They calculate AO for each depth layer and keep the highest occlusion which improves the AO estimator, but can overestimate occlusion. They also propose a dual resolution adaptive sampling approach to increase performance. The use of depth peeling is able to complement the geometric information but still misses occluders due to polygons that remain parallel to the view direction. Furthermore, it requires additional passes to generate the required visibility information.

Recently, Bauer et al. [BKB13] proposed the use of A-buffer linked-lists for rendering multi-layer opacity-based ambient occlusion. The use of an A-buffer, however, can pose important memory constraints for real-time purposes and still misses geometry parallel to the view.

Relevance to this Thesis

In Chapter 4, we build on the concept of multiple viewpoints [RGS09] for screen-space visibility sampling, but propose a weighting scheme for combining arbitrary views, such as the shadow maps which are already generated in the rendering pipeline in typical real-time applications, such as games. The exploitation of depth information from existing views translates to no additional computation cost or extra passes. The weighting scheme balances the contribution of each view, without overestimating occlusion or the need of a special view setup. Furthermore, we show how to utilize the view weighting function in order to perform importance sampling and an adaptive sample reduction, thus significantly reducing the rendering time, while maintaining high quality. Multiview occlusion can yield better results than simple depth peeling but can also include importance-sampled depth-peeled views as a special case. Finally, user-defined (phantom) views like the ones used in [RGS09] can be exploited and seamlessly blended according to their importance. We propose such a camera configuration to address view-dependent artifacts in hard cases, such as confined, indoor scenes with high occlusion, where previous approaches fail. We apply our multiview method to ambient occlusion/obscurance calculation based on the Alchemy AO, SSAO and HBAO algorithms, but the method can be applied to other similar techniques.

3.2.2 Image-based Global Illumination

Real-time image-based GI methods focus on approximating illumination effects based on the geometric information available in the G-buffer. This results in techniques that are independent of scene complexity, and can be rendered very fast. However, all screen-space methods are prone to view-dependent artifacts, due to missing geometric information in the depth buffer and undersampling issues.

The *Reflective Shadow Maps* (RSMs) [DS05] algorithm is the most GPU-friendly implementation of instant radiosity (see Section 2.4.4) for single-bounce indirect illumination. RSMs exploit the rasterization

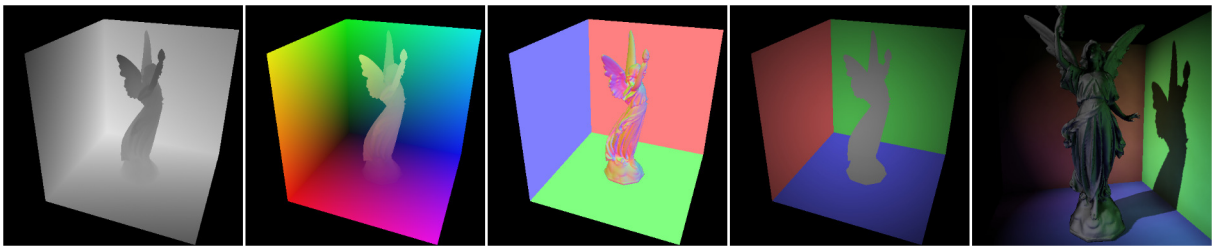


Figure 3.3: RSMs store multi-channel information with respect to the light source, resolving visibility for the first indirect bounce of light. From left to right: the different components stored in the light G-buffer (depth, position, normal, flux) and the final image. Image source: [DS05].

procedure for the generation of shadow maps to position the VPLs, thus avoiding tracing them in the scene. RSMs are implemented similarly to shadow maps but each *pixel light*, aside from depth, stores information such as reflected flux, normal vector and world space position in a multichannel G-buffer (see Figure 3.3). The final indirect illumination is accomplished by a simple gathering procedure for each point to be shaded. In general, RSM methods suffer from high variance due to the large number of samples required to reproduce the final image and do not take indirect visibility is into account. The same authors [DS06] replace the gathering procedure with an image-space splatting approach to increase performance and reduce variance. Nichols et al. [NW09] used multi-resolution splatting and eye-space interpolation to reduce the shading computations and illumination artifacts respectively.

To mitigate the indirect visibility problem, Laine et al. [Lai+07b] generate shadow maps for each VPL, but only a few VPLs are created each frame by exploiting temporal coherency. The VPLs from previous frames are kept if they are still visible to the light source. Ritchel et al. [Rit+08] created an approximate point-based representation of the scene in a preprocessing step and use this information to generate many low resolution (*imperfect*) shadow maps, in order to approximate indirect visibility for each VPL in the scene. Their approach was able to approximate indirect visibility at interactive rates, but the point-based representation can miss indirect visibility from fine-grained geometry.

A more efficient VPL placement and a view-adaptive distribution of the point-based scene approximation without precomputations is proposed by Ritschel et al. [Rit+11]. Dong et al. [Don+09a] extended VPLs to Virtual Area Lights (VALs), which contain groups of VPLs with similar information. Indirect visibility is checked against VALs, while illumination is perform using the VPLs.

Ritschel et al. [RGS09] proposed screen-space directional occlusion (SSDO) which calculates near-field directional occlusion and indirect illumination by generating samples in a hemisphere above the shaded point and testing them against the camera depth buffer. The direction of each visible sample contributes to the direct illumination, while blocked samples are projected on the depth buffer and the contribution from the corresponding point samples from the direct lighting buffer is estimated. Due to its dependence on a single depth check per sample, this technique cannot guarantee the visibility of the projected sample, especially for large radii.

McGuire et. al.[ML09] introduced Image Space Photon Mapping where GPU rasterization is used for the primary light bounce and the CPU is used for the remaining photon bounces, using traditional ray tracing. Instead of performing gathering to compute the final indirect illumination, photons are *splatted* in screen-space using the GPU.

Caustics mapping techniques [WD06; HQ07; SKP07] operate in image-space to simulate caustics at interactive rates. Typically, these methods render the environment from the light's point of view and create a caustic map, which is projected onto the scene geometry on a later step. The photon gathering process is commonly replaced either by an image-based nearest neighbor search [WD06] or by a splatting procedure [HQ07; SKP07]. Employing fixed splat sizes, however, introduces noise and aliasing artifacts which is reduced by varying the splat size [WD08], hierarchical methods [Wym08] as well as by adaptive sampling through deferred shading [WN09].

Multi-layer Approaches

The previously discussed SSDO technique can be also augmented with the use of multiple views, in order to account for the missing geometric information in the camera frame buffer. The authors employ the use of additional depth-peeled views, positioned relative to the user's camera target position, and render the scene from these viewpoints as well as the main camera's. However, this configuration requires several additional passes and does not guarantee that important direct lighting is present in any of the generated views, especially in environments with high occlusion.

Mara et al. [Mar+14] described several algorithms to achieve two-layer depth peeling in a single pass, demonstrated in various real-time applications such as reflections, diffuse global illumination and ambient occlusion. Compared to prior work, which employed depth peeling for order-independent transparency, each layer is viewed as a G-buffer element containing additional surface properties for global illumination computations. Instead of naively selecting the second layer, the authors forced a minimum separation between layers via a fixed view-based metric, thus skipping geometry with potentially similar contribution to the GI due to proximity with fragments already sampled in the previous layer. However, there is no guarantee that important geometry will be missed this way as surfaces within the separation distance may have similar geometric terms of energy exchange but can have significantly different material properties.

Nalbach et al. [NRS14] proposed a deep screen space approach to overcome the drawbacks of single-layer image-space illuminations methods. Instead of approximating the scene using the depth buffer in order to resolve visibility and perform illumination computations, they approximate it using a view-dependent surfel cloud of the triangle mesh using hardware tessellation. Each surfel is created so that its radius is the same in screen space independent of world space position and orientation. The surfels are then splatted to an array texture of different resolutions levels, so that pixels that are located near a surfel use the full resolution image, while distant pixels use their subsampled versions. Their technique was able to achieve interactive rates and was demonstrated in ambient occlusion, single-bounce indirect diffuse and glossy illumination, directional occlusion and subsurface scattering.

3.2.3 Volume-based Global Illumination

Evaluating the light transport integral for each point is a costly operation and still prohibitive for dynamic environments and interactive applications. Therefore, a large amount of effort has focused in the development of algorithms, which use some form of caching. Instead of estimating global illumination for the entire environment, it is more efficient to compute GI only for a number of representative points in space and use some form of interpolation for the rest of the scene (see Figure 3.4).

Irradiance and Radiance Caching

The idea of storing irradiance values at spatially indexed data structures was initially introduced by Ward et al. [WRC88; War94] for offline rendering. The original irradiance caching was used in scan-line style renderers; values already computed were cached and used for the subsequently evaluated points via interpolation, while irradiance gradients were employed to better predict the missing data and improve the quality of extrapolation [War92]. Krivanek et al. [Kri+05] extended Irradiance Caching to Radiance Caching to compute the indirect glossy and diffuse terms of low frequency BRDF's using hemispherical harmonics at non-interactive rates.

Greger et al. [Gre+98] introduced the Irradiance Volume to compute diffuse global illumination to speed up computations on semi-dynamic environments. The key idea is that instead of storing irradiance values at surfaces locations, irradiance was stored in a bilevel volumetric grid, where each grid location contained directional irradiance values in the form of a radial function, as an approximation of the irradiance of that point in the environment. The Irradiance Volume was built in a preprocessing step for static objects from a radiosity solution and surface irradiance for dynamic objects was then approximated by interpolating the stored directional irradiance values in the lattice vertices, speeding up the computations. However, occlusion information from dynamic objects was not taken into account.

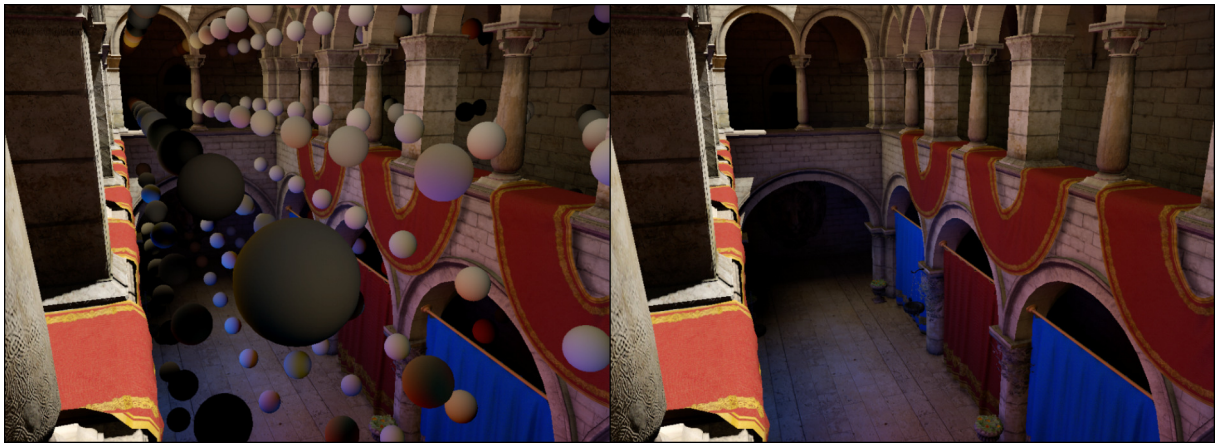


Figure 3.4: Volume-based caching methods store radiant energy on a sparse representation of the scene. The directional distribution of incoming (or outgoing) light is commonly represented through projection to a radial basis function. The rendering equation is evaluated for a point by reconstructing and interpolating the irradiance integral from these stored samples. Left: The spheres represent the indirect illumination (stored at their centers) using spherical harmonics. Right: the final scene. Images rendered with XEngine [VP], using the method presented in Chapter 5.

In order to store radiometric information in space, the radiance field needs to be estimated over all incoming directions at a particular location in space. In interactive applications, where the indirect illumination is mostly limited to diffuse surfaces, this is commonly achieved by representing the spherical function of the radiance field as the coefficients resulting from the projection of the radiance field on a set of orthonormal basis functions, such as Spherical and Hemispherical Harmonics [Sil+91; RH01]. Their most important property is that integrating two functions that have been projected to such a spherical basis, is equivalent to taking the dot product of their coefficients (see Section 2.1.2). As a result, the simplified version of the rendering equation for diffuse illumination (Equation 2.50) can be replaced by the dot product between the two projected functions, $L_i(\mathbf{x}, \omega_i)$ and $\cos\theta_i$. Other forms of representations exist as well, such as Spherical Radial Basis Functions [Leu+06] or Wavelets [NRH03; NRH04; Kon+06], but are less commonly used. Note that the Spherical Radial Basis Functions do not obey orthonormality and thus break the dot product property and Wavelets are not particularly efficient to compute and manipulate.

Similar to the above, Nijasure et al. [NPG05] use a spherical harmonics representation to sparsely encode the radiance field at the vertices of a 3D grid. The estimation of the radiance field however is performed via cube map rendering. Cube maps are rendered at the center of each cell and the incident radiance is encoded using spherical harmonics coefficients. The final indirect illumination at surface points is approximated by interpolating the radiance from the closest grid points. Their method is view-independent, supports multiple bounces and indirect occlusion, but requires a large number of draw calls for the generation of cube maps.

Light propagation volumes [Kap09; KD10; KED11] use a Discrete Ordinates Method scheme for indirect illumination. VPLs are injected into a 3D grid using samples from RSMs and they iteratively propagate light from cell to cell. Indirect occlusion is handled by using fuzzy volumetric representation of the scene stored in a separate volume, which contains view-dependent blocker information injected from the RSMs and the camera. View-dependencies can be mitigated through depth peeling. The method has been successfully used in games, mostly due to its compatibility with volumetric scattering effects and controllable (and bounded) calculation time. On the other hand, the energy transport model used is very approximate and tends to over-blur the resulting irradiance. Furthermore, the result depends on the number of iterations employed (propagation distance measured in cells), potentially excluding contribution from geometry outside this range. The hierarchical version of the described method, the cascaded light propagation volumes [KD10], removes this restriction but drastically coarsens the granularity of the solution for middle- and far-range geometry.

Mavridis et al. [MP11] create a sparse point-cloud of the scene geometry and store occupancy and diffuse illumination at each voxel of a 3D lattice. A camera and RSM-based reprojection refinement scheme is employed to remedy any missing holes introduced by the voxelization.

Papaioannou [Pap11] combines the grid-based radiance caching of Nijasure et al. [NPG05] with RSM sampling to generate the radiance field at each cache location, thus dispensing with the cubemap rendering. The key idea is that all light distributed in the scene is contributed from the light sources and thus the reflective shadow map VPLs alone represent the non-zero radiance in the scene. Incoming radiance from RSM samples is therefore directly injected to a 3D grid and encoded in spherical harmonics. First bounce indirect shadowing is supported optionally using a view-dependent probabilistic attenuation scheme exploiting the depth buffer information. For multiple bounces, a geometry-less energy exchange scheme is proposed that exchanges energy among the cache points and secondary bounce visibility is statistically approximated.

Relevance to this Thesis

In Chapter 5, we identify three problems existing in prior techniques, affecting both the performance and quality of the indirect illumination. First, when expressing the radiance field as a truncated series of spherical harmonics, a large number of coefficients is needed to sufficiently approximate the original signal, increasing the bandwidth requirements due to texture fetches during the reconstruction operation. For this, we introduce a compression scheme based on the idea of chroma subsampling (see Section 2.5). Second, computing the radiance field for the entire grid imposes an unnecessary computational overhead when participating media are absent. There, we employ an optimized positioning scheme of radiance cache points which stores information only near surface locations. Finally, indirect shadowing is either expensive to compute (through cubemap representations), computed probabilistically, or is view-dependent (based on the depth buffer and the RSMs). In our method, we employ a binary geometry volume to provide view-independent indirect shadowing.

Ray marching and Cone tracing

Thiederman et al. [Thi+11] store the surface information of the objects in a scene in *geometry images*, i.e., bijective texture maps, which encode the position, normal and material attributes of the geometry. These texture atlases are subsequently efficiently injected as point samples in a volume grid to create a view-independent volume representation of the scene. Then, indirect illumination is estimated using ray marching on the generated volume. Ray-intersections are accelerated through variable interval marching within the volume data and indirect illumination is gathered from RSMs.

Crassin et al. [Cra+11] proposed a volume-based cone tracing technique to efficiently approximate single-bounce indirect diffuse and specular illumination. Direct illumination is injected in a sparse voxel octree [CG12] and gathered at the shaded points by cone marching. Each cone is sampled by stepping along the cone axis and selecting the appropriate mip level based on the current cone radius. Their method is generic and capable of detailed results at interactive rates, provided that a small part of the volume is dynamically updated. Furthermore, this method is memory intensive as a large number of attributes have to be encoded per voxel for illumination computations.

Sugihara et al. [SRS14] improved upon the original algorithm by decoupling occlusion from illumination data to reduce the memory overhead. Instead of using prefiltered multichannel voxels, the voxelization holds binary occlusion information and the lighting information is stored in a prefiltered RSM structure, called layered RSMs (LRSM). The injection operation of the lighting data is replaced by a gathering operation in the LRSMs. Their method reduces the total memory consumption, but does not scale well with the number of light sources and is limited to single-bounce indirect illumination.

3.2.4 Rasterization-based Ray Tracing

Ray tracing can quickly become a computational bottleneck since it spawns millions of incoherent rays in multiple directions and calculates intersections with every object in the scene. While a large amount of research effort has been devoted towards improving the quality and performance of the spatial data structures in GPGPU approaches (discussed briefly in Section 3.1.2), the needs for high-quality and interactive content in large and complex environments have also resulted in recent attempts to perform and improve ray tracing through the rasterization pipeline, essentially fusing the advantages of these two pipelines.

The use of global ray-bundles; sets of rays with a global, common direction for the entire environment [SS96; SP98] has been exploited in the rasterization pipeline to simulate full global illumination solutions, such as path tracing [Hac05; Her+10; Tok+13] and bidirectional path-tracing [TO12]. Despite their acceptable global illumination approximation output, however, these methods are not able to sample at arbitrary locations, leading to various aliasing artifacts.

Zirr et al. [ZRD14] performed object-order ray tracing by exploiting a coarse voxel grid that stored links to all rays intersecting each cell. Their deferred scheme is able to perform analytic intersection tests, but required an undetermined number of geometry passes and was limited to single-bounce indirect illumination.

Hu et al. [Hu+14] proposed a full ray tracing solution using only the rasterization pipeline. The scene is projected orthographically onto three A-buffers along with a low resolution volume for fast ray-fragment intersection tests. The volume, which contains the fragments' average normal, is traversed with a 3D Digital Differential Analyzer (DDA) algorithm to quickly identify a general hit location, and the A-buffers are then traced to locate the intersected fragment. However, the authors assume that each voxel contains one smooth surface and their method is therefore applicable to scenes of limited extent and granularity, in order to sufficiently capture the underlying geometric detail.

Ganestam et al. [GD15] use a hybrid approach to render perfect reflections and refractions. A boundary volume hierarchy is created to store the geometry near the camera and a cubemap of G-buffers, each with a field of view of 90 degrees, for the rest. Their method, which requires both GPGPU (CUDA) and the rasterization pipeline, can achieve interactive frame rates but is still prone to view-dependent artifacts due to the single layer information stored in each face.

Widmer et al. [Wid+15] utilize their adaptive acceleration structure on a cubemapped variation to offer multilayer reflections by off-screen objects at interactive frame rates, but their results are limited to two-layered representations, opaque geometry and diffuse surfaces.

Relevance to this Thesis

While some of the aforementioned methods are able to support a full global illumination algorithm, none of them is capable of performing ray tracing computations without posing any restrictions on the geometric representation or on the effects that can be reproduced. Contrary to these approaches, the techniques presented in this thesis are far more generic, as they are not based on discretized direction sets, are not restricted to a particular type of surface or effect and are able to capture in high detail, multi-layer information residing both inside and outside the viewing frustum. As a result, view dependencies are either minimized (Chapter 6) or eliminated completely (Chapter 7).

3.2.5 Image-space Ray Tracing

Screen-space techniques are widely popular in the real-time community due to their capability to compute post-processing effects as well as visually pleasing, yet approximate, global illumination phenomena. Regarding the latter, visibility between points is commonly determined by performing ray tracing in image-space, e.g., using the G-buffer.

The most widely used method for image-space ray tracing in the games industry, is the object-space linear ray marching method as described by Sousa et al. [SKS11]. To identify a hit, a ray's position in 3D

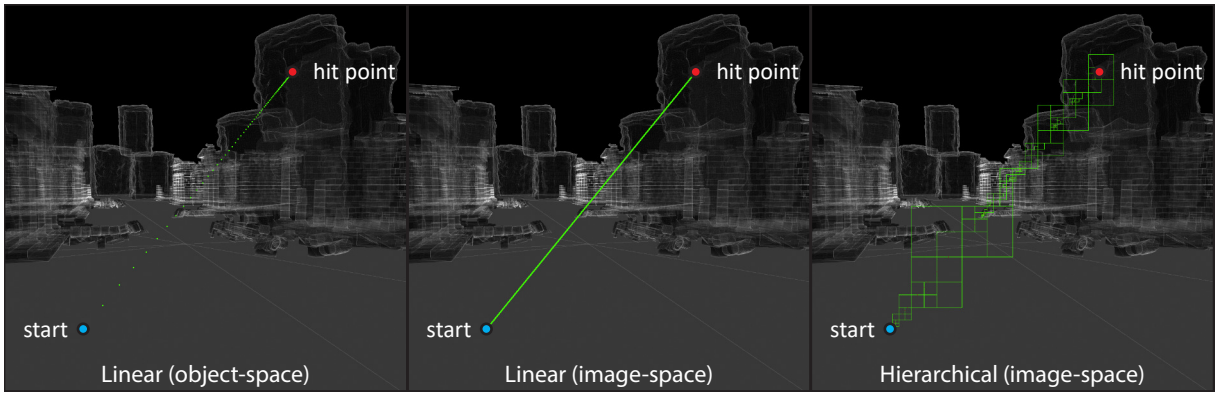


Figure 3.5: Example of screen-space ray tracing, where the ray is directed towards the hill at the top left. Left: 3D ray marching. A ray is incremented in fixed-size object-space increments (2 meters in this case) and tested for intersections only at these locations (green pixels). Under- and over-sampling artifacts occur due to perspective projection, resulting in pixels being skipped close to the near plane (near the start position) and pixels being oversampled as the ray approaches the far plane (near the hit point). Middle: 2D ray marching. The ray is incremented in pixel-space increments, avoiding incorrect hits due to under-sampling artifacts. Right: Hierarchical 2D ray marching. The ray is sampled hierarchically, reaching the intersection location in fewer steps. The colored rectangles indicate intersection queries against different mip levels of the hierarchical depth buffer.

space is incremented in fixed-size steps for a bounded distance. At each step, the current location of the ray is projected in screen-space and tested against the current value stored in the depth buffer (see left of Figure 3.5). Despite its simplicity, this method suffers from efficiency and performance issues. Performing a large number of steps might result in looking at the same pixel multiple times (over-sampling), wasting computational resources. On the other hand, employing a small number of steps will eventually skip pixels representing correct intersection locations, resulting in skipping intersected geometry.

McGuire and Mara [MM14] proposed an efficient 2D DDA GPU implementation by moving in pixel-space increments instead of object-space (see middle of Figure 3.5). As such, they are able to capture both contact details and distant geometry as well as to address the under- and over-sampling limitations of the previous method. The authors demonstrate screen-space ray tracing against not only single, but also multiple depth layers. However, the proposed scheme can exhibit erroneous behavior in multi-layer schemes, as we discuss in Section 6.2.3.

Single-layer screen-space ray tracing can be also performed hierarchically through a mip-mapped depth buffer representation [Ulu14]. Essentially, this reduces the 2D traversal complexity from $O(n)$ to $O(\log n)$ (right of Figure 3.5). In this approach, the ray is tested against the values contained in the mip hierarchy, instead of only the high-detail depth buffer. For each successful intersection test, the mip level is iteratively refined in order to determine the precise hit location. If an intersection test fails, the mip level is increased, where each pixel represents a larger area to be tested, and the ray moves to the next block, effectively skipping large areas in screen-space.

Relevance to this Thesis

Regardless of the sampling approach taken for screen-space ray tracing, all aforementioned methods are based on ray-fragment intersection tests, i.e., rays are tested against the discretized representation of the geometry stored in one or multiple depth buffer layers. As such, even with accurate pixel-based sampling strategies (linear or hierarchical) rays can still pass through sparsely sampled geometry and incorrectly intersect with the wrong one. A common approach to mitigate these artifacts is to employ a tolerance factor during the intersection tests, essentially representing each fragment as a frustum-shaped voxel of non-zero thickness [GD15]. Still, correct intersection cannot be ensured, as explained in Chapter 7 (see also Figure 7.7). Furthermore, all current methods are mainly designed to properly handle single-layer

scenarios. Apart from the quality issues, there has been no investigation on the performance considerations with respect to performing intersection queries against arbitrarily complex multi-layer representations.

In our work, we investigate all these problems and generalize screen-space ray tracing in terms of both performance and quality. More specifically, our contributions with respect to the quality are:

- A modification of screen-space ray tracing (linear and hierarchical) to properly handle fragment-based intersection tests when applied to multiple layers/views (Chapter 6).
- A generalization of screen-space ray tracing (linear and hierarchical) to properly handle analytic ray-primitive intersection queries (Chapter 7).

Also, we achieve significant improvements on the performance of ray traversal by applying various culling optimizations. Here, our contributions are:

- Empty space skipping is applied in both the near and far sides of the frustum as well as in depth-space, by exploiting per-pixel depth boundaries as well as a uniform depth-subdivision strategy (Chapters 6 and 7).
- Hierarchical screen-space ray tracing is augmented by increasing the finest mip level in the hierarchy, essentially trading traversal overhead in image-space with larger lists in the depth domain. This modification provides significant performance benefits in certain scenarios in our tiled-based ray tracing architecture (Chapter 7).

Chapter 4

Multiview Ambient Occlusion with Importance Sampling

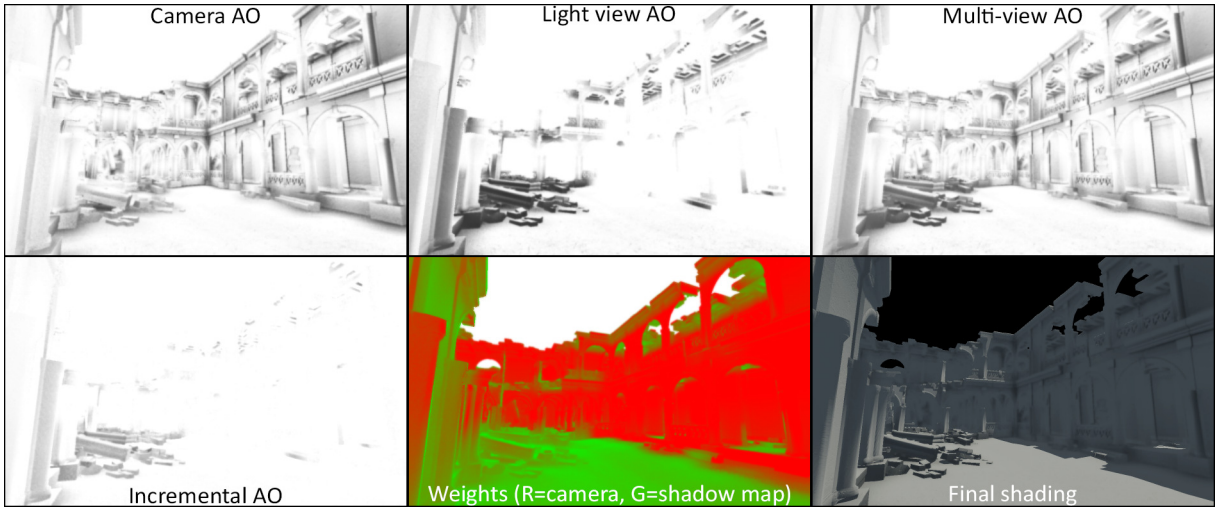


Figure 4.1: Multiview ambient occlusion combines depth information from any available views to fill in missing occlusion.

Screen-space ambient occlusion and obscurance techniques have become de facto methods for ambient light attenuation and contact shadows in real-time rendering. Although extensive research has been conducted to improve the quality and performance of AO techniques, view-dependent artifacts remain a major issue. In this chapter, we introduce Multiview Ambient Occlusion, a generic per-fragment view weighting scheme for evaluating screen-space occlusion or obscurance using multiple, arbitrary views, such as the readily available shadow maps. Additionally, we exploit the resulting weights to perform adaptive sampling, based on the importance of each view to reduce the total number of samples, while maintaining the image quality. Multiview Ambient Occlusion improves and stabilizes the screen-space AO estimation without overestimating the results and can be combined with a variety of existing screen-space AO techniques. We demonstrate the results of our sampling method with both open volume- and solid angle-based AO algorithms.

The chapter is organized as follows: Section 4.1 discusses the view-dependency problem of single-space ambient occlusion techniques and our contribution. Sections 4.2 and 4.3 presents the outline and implementation details of our algorithm, respectively. Section 4.4 reports on the efficiency and robustness of our method and discusses limitations and, finally, Section 4.5 offers conclusions and discusses future research directions.

4.1 Overview and Problem Description

Ambient occlusion is a non-physically based method which tries to approximate the amount of indirect light that reaches a point based on its surrounding occluders, without taking into account interreflections (see Section 2.3.2). Two main approaches are taken when approximating ambient occlusion and obscurance of complex, dynamic scenes; object- and image-based methods. Object-space methods produce stable, high-quality results, but the proposed techniques are not suited for applications with limited per-frame time budget for global illumination effects. On the other hand, image-based methods introduce a trade-off between quality and performance. They produce reasonably convincing results and offer bounded rendering times as they are decoupled from scene complexity, making use of information already stored in the camera G-buffer.

In general, image-space techniques use buffers from available views, such as the camera G-buffer or shadow maps to obtain information related to a world-space point and its surrounding geometry. Typically, samples in the vicinity of a point are projected onto the buffers and the resulting image-space locations derived through the view's depth map constitute an approximate reconstruction of the geometry at it. The recovered information, is used in calculations such as ambient occlusion, obscurance and indirect lighting. Despite the stable and fast performance of image-space methods, they all suffer from view-dependent artifacts. This is mainly due to the absence of occluders, unseen in the image buffers of the current view (inside and outside the view frustum). These artifacts are manifested in the form of shadows that pop up as the camera moves and haloing effects.

Our Contribution

In this work, we present a single-pass generic method which addresses and improves the view-dependent inconsistencies of current screen-space ambient occlusion techniques by exploiting geometric information from other view points, such as the shadow maps, already generated as part of the rendering process. A weighting scheme for balancing the contribution of arbitrary views is proposed, so that our technique can be applicable to environments with high occlusion and geometric complexity without erroneously estimating occlusion (see Figure 4.1). Furthermore, we utilize our weighting functions to distribute our samples based on their importance which, along with an adaptive sample reduction, reduces the computational cost significantly while being able to maintain very high quality. Finally, for very hard cases such as confined, indoor scenes with high occlusion, we propose a camera configuration based on user-defined (phantom) views, similar in spirit to the ones used in SSDO [RGS09]. The views can be exploited and seamlessly blended according to their importance. We apply and demonstrate our multiview method to ambient occlusion/obscurance algorithms such as Alchemy Ambient Obscurance [McG+11], Screen-Space Ambient Occlusion (SSAO) [Mit07], and Horizon-Based Ambient Occlusion(HBAO) [DBS08], but the method can be applied to other similar techniques.

To summarize, our contributions in this chapter are:

- A *generic* method for reducing view-dependent artifacts of screen-space ambient occlusion techniques, which estimates the occlusion importance of an arbitrary number of views and reduces errors in the final occlusion estimation, even for the main camera view.
- An importance-driven and adaptive-based sampling algorithm for the distribution of samples among the views, able to reduce rendering times significantly without sacrificing the final quality.
- An extensive experimental study of the most common ambient occlusion techniques, as well as their applicability to our a multiview weighting scheme.

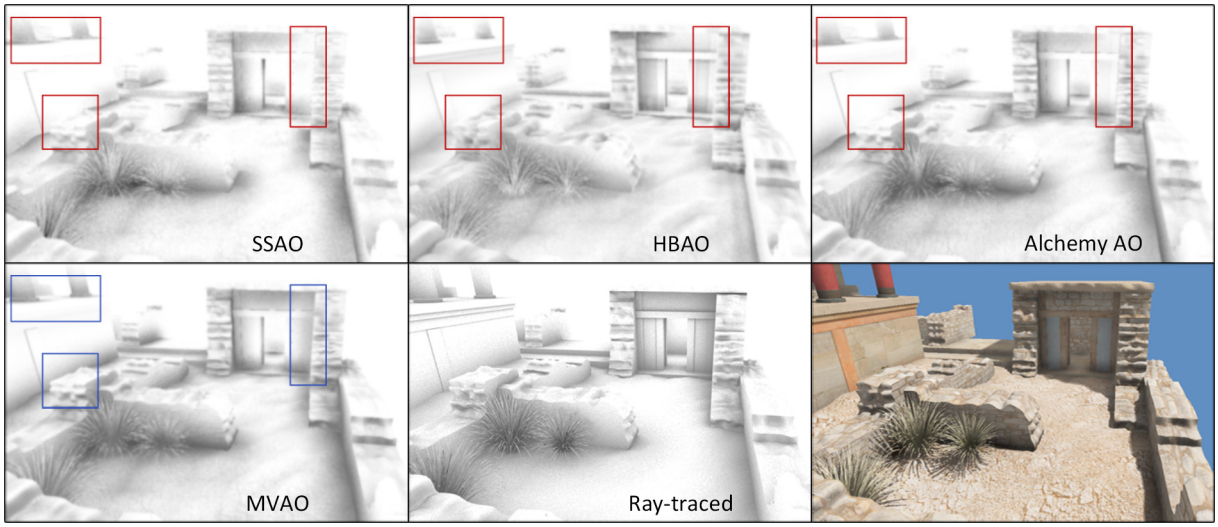


Figure 4.2: View dependencies of screen-space AO algorithms. Marked regions indicate erroneous occlusion due to parallax.

4.2 Method Description

As screen-space AO techniques evolved, they attempted to improve mostly issues related to sampling quality, memory access efficiency and faster and more accurate convergence of the occlusion integral estimator. A prominent example of this constant refinement is the Scalable AO by McGuire et al. [MML12]. However the "traditional" problems of screen-space methods, such as view-dependent occlusion fluctuations and erroneous shadowing, still remain (see for example Figure 4.2). View dependency commonly manifests itself either as a gradual darkening at the edges of the frame buffer, as new depth values start contributing to the occlusion estimation, or as erroneous occlusion due to lack of geometric information in the depth buffer. The first problem is remedied either by fading in the occlusion at the image boundaries or by virtually extending the image buffer beyond the viewport. The second problem, i.e., the lack of depth information within the image, is the root of severe artifacts in many cases, where geometry parallel or nearly parallel with respect to the camera viewing direction causes discontinuities in the depth buffer and makes samples fall on different surfaces than the shaded point lies on. If these samples are not culled or attenuated (usually with respect to the maximum AO range r_{max}), they cause a darkening halo at the depth discontinuities. If on the other hand they are rejected, the result is an unnatural, view-dependent brightening of normally shadowed crevices. The latter is also demonstrated in Figure 4.2.

Multiview Ambient Occlusion (MVAO) takes advantage of image buffers from secondary views, such as shadow maps, which are already available at AO calculation time, in order to improve and stabilize the latter, when screen-space techniques are used. For each fragment to be shaded in the camera frame buffer, the occlusion or obscurance is estimated using any number of available views and the final AO is the combination of the partial results. The (partial) occlusion in each one of the image buffers is calculated using a standard AO algorithm, such as the Alchemy AO method or SSAO. The combined occlusion however, is not the average or maximum occlusion of all partial results, but rather a weighted sum of them, since we need to favor views with better visibility of both the shaded point and the samples drawn in its neighborhood and subsequently projected in the corresponding depth buffer. For instance, in Figure 4.2, AO samples for a point on geometry visible just above the broken wall in the foreground are more reliably projected in the shadow map of the overhead light in the scene rather than in the camera view. Since we allow multiple arbitrary views, the shaded points and near-field samples may be occluded in any other view but the camera buffer, and therefore the importance of these views must be adjusted per fragment accordingly. The weighting functions for combining the partial AO results from the multiple views are discussed in the following section.

4.3 Algorithmic Details

Given a number of n views we calculate the occlusion for a point $\mathbf{p}$ as:

$$O(\mathbf{p}) = \frac{\sum_{v=1}^{n} O(v,\mathbf{p})w(v,\mathbf{p})}{\sum_{v=1}^{n} w(v,\mathbf{p})}, \tag{4.1}$$

where $O(v,\mathbf{p})$ and is the occlusion of view v at point $\mathbf{p}$ and $w(v,\mathbf{p})$ is a weighting function describing the *contribution* of the occlusion of view v at point $\mathbf{p}$.

4.3.1 View Weighting Function

Any screen-space sampling implementation assumes that the point of interest $\mathbf{p}$ is always visible in the image buffer and that the same is true for the samples taken within its near field. Additionally, the eye-space normals at all sampling locations, including the central point $\mathbf{p}$, always point in the direction of the center of projection. However, when projecting a camera buffer point to an arbitrary view, any of the above assumptions may not hold. Furthermore, even if $\mathbf{p}$ and the projected samples in view v comply with the above conditions, the view-dependent projection of the samples, especially at grazing angles, makes certain view points unreliable for a given point $\mathbf{p}$.

The importance of a particular view v in the calculation of the occlusion $O(\mathbf{p})$ at any point $\mathbf{p}$ visible in the camera frame buffer depends on three factors: the visibility of $\mathbf{p}$ in view v, the orientation of the surface at $\mathbf{p}$ with respect to the view direction and finally, the proximity of the projected samples on the v-th depth buffer to $\mathbf{p}$. These three factors affect the overall contribution of the partial result $O(v,\mathbf{p})$ to the final occlusion in the form of a weighting function $w(v,\mathbf{p})$:

$$w(v,\mathbf{p}) = bw_d(v,\mathbf{p}) + (1-b)w_n(v,\mathbf{p}), \tag{4.2}$$

where $w_d(v,\mathbf{p})$ is the sample distance weighting function, $w_n(v,\mathbf{p})$ is the directional weighting function and b is a blending coefficient to bias the view weight in favor of the distance or the orientation importance weight. The visibility indirectly affects the weighting function through the distance weight. Directly accounting for the visibility of a point in a given view would cause discontinuities in the weighting function, resulting in visible banding, for instance at shadow boundaries, if light views are used. The blending coefficient value depends on the AO method used and is discussed in Section 4.4.3.

Distance Weight

To avoid sampling occlusion at depth discontinuities and across depth layers or image regions where geometry is obliquely projected, we favor views where the projected samples land closer to the central (shaded) point $\mathbf{p}$ (see Figure 4.3). This is achieved by keeping track of the *average distance* from $\mathbf{p}$ to the projected samples $\mathbf{s}_i$ on view v (clamped to r_{max}):

$$w_d(v,\mathbf{p}) = 1 - \frac{1}{N_s r_{max}} \sum_{i=1}^{N_s} \min(||\mathbf{p}-\mathbf{s}_i||, r_{max}), \tag{4.3}$$

where N_s is the total number of samples per view. Visibility of the central point $\mathbf{p}$ is handled indirectly through the above weighting function; for an occluded point $\mathbf{p}$ in the current view, many of the projected samples $\mathbf{s}_i$ will lie on the occluder, thus increasing the average distance and lowering the importance of view v. However, the decrease of $w_d(v,\mathbf{p})$ will happen gradually, as more and more samples land on the occluding depth layer when $\mathbf{p}$ moves behind an occluder, thus avoiding any visible banding.

Note that the calculation of the distance weight incurs no additional cost, since $\mathbf{s}_i$ are already available and $||\mathbf{p}-\mathbf{s}_i||$ is needed for the distance-based obscurance attenuation function anyway.

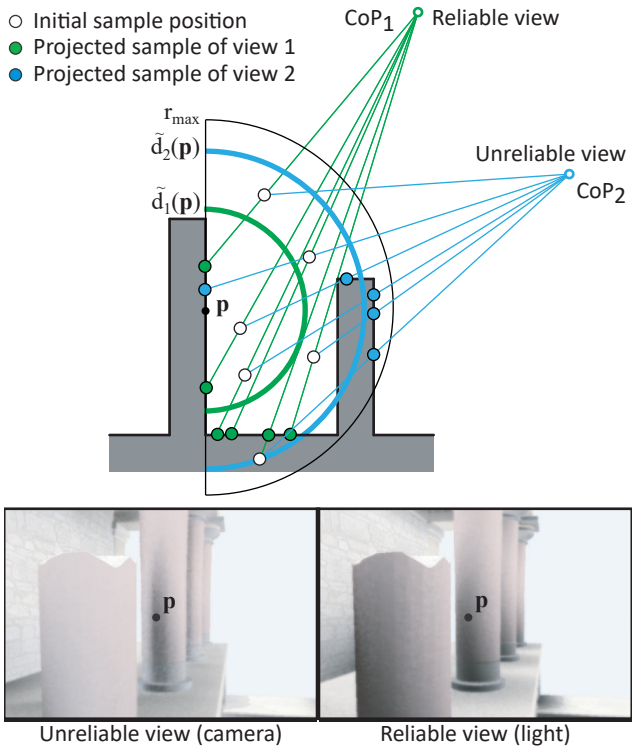


Figure 4.3: Distance weighting of a view with respect to the average distance of projected samples from a given shaded point. Intense parallax in the depth buffer can cause many false positive samples in the projection of the sampling pattern on the depth buffer. The thick green and blue lines visualize the distance weight for each view, i.e., the average distance of the projected samples from $\mathbf{p}$.

Directional Weight

A common problem in screen-space AO techniques is that, when the normal at a shaded point $\mathbf{p}$ diverges from the viewing direction, the probability that a sample is projected on a disjoint or distant patch of geometry, as registered in the depth map of the view, increases. Worse, in the case of multiview projection, where the normal $\mathbf{n}$ at $\mathbf{p}$ may point away from the view, the projection of the sample has a significantly higher probability to land on a disjoint region than near $\mathbf{p}$, causing erroneous occlusion. Figure 4.4 demonstrates this issue; in a highly oblique view, samples on an image disk or in a hemisphere above $\mathbf{p}$ exhibit a strong directional bias towards one side of the sampling pattern and are prone to be projected out of the near-field range r_{max}. Although the effect of this issue varies according to the screen-space algorithm used, the problem may occur in all views, including the camera view (e.g., at grazing angles with respect to the geometry). A simple criterion to favor views facing along the normal vector at $\mathbf{p}$ is the dot product of $\mathbf{n}$ and $\mathbf{l}_v$:

$$w_n(v, \mathbf{p}) = max(0, \mathbf{l}_v \cdot \mathbf{n}), \tag{4.4}$$

where $\mathbf{l}_v$ is the direction from $\mathbf{p}$ to the center of projection of view v. Views residing behind $\mathbf{p}$ are highly unreliable, therefore their contribution is set to zero by clamping the dot product accordingly.

4.3.2 Importance Sampling

We can use the weighting function of Section 4.3.1 in order to direct more samples towards views with higher contribution to the final AO. However, since the distance-based weight $w_d(v, \mathbf{p})$ depends on the yet unknown position of the depth samples, we initially generate a first estimate of the occlusion $\tilde{O}(v, \mathbf{p})$ and the corresponding weights $\tilde{w}(v, \mathbf{p})$ with a small number of samples N_{init} for each view v, for example, 5-7. The choice of N_{init} depends on the specific AO algorithm used (see quality comparison in Figure 4.8).

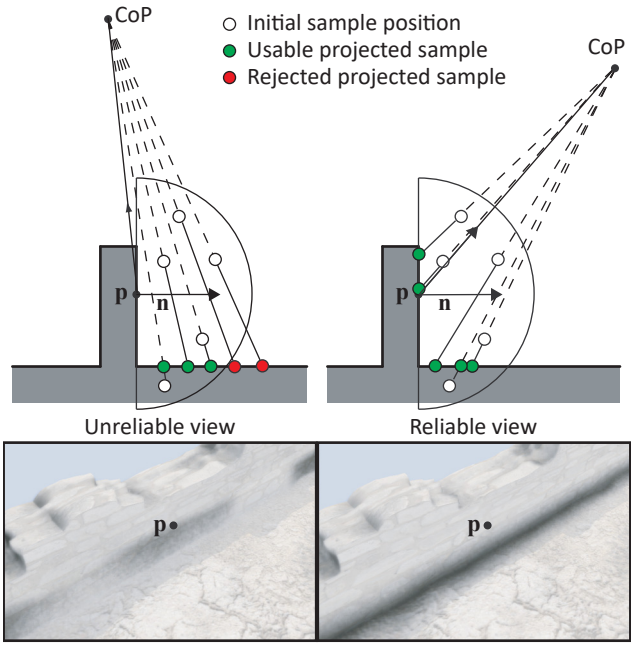


Figure 4.4: Directional weighting of a view with respect to the normal vector at the shaded point. Oblique projections or viewpoints facing away from the normal can cause severe artifacts in view-dependent near-field geometry recovery from depth.

$\tilde{O}(v, \mathbf{p})$ and $\tilde{w}(v, \mathbf{p})$ are used to perform importance sampling on the image buffers in a second iteration, with $N(v, \mathbf{p})$ samples for each view v.

The importance of a particular view v and consequently, $N(v, \mathbf{p})$, is directly proportional to the initial weighting function estimate $\tilde{w}(v, \mathbf{p})$. Additionally, since we have a first estimation of the occlusion at $\mathbf{p}$ from view v, $N(v, \mathbf{p})$ can be biased against regions with low occlusion to avoid wasting samples on uninteresting regions. This way we combine importance-driven sample distribution among the views and occlusion-driven adaptive sampling in a single, *adaptive step*. The proposed heuristic for determining $N(v, \mathbf{p})$ is directly proportional to both the initial estimate of the occlusion and the current view weight:

$$N(v, \mathbf{p}) = (N_s - N_{init})\tilde{w}(v, \mathbf{p})\frac{1 + \tilde{O}(v, \mathbf{p})}{2}. \tag{4.5}$$

The new importance-sampled occlusion $O_{\text{is}}(v, \mathbf{p})$ of each view v from the adaptive step is added to the initial estimate $\tilde{O}(v, \mathbf{p})$ and the updated occlusion becomes:

$$O(v, \mathbf{p}) = \frac{O_{\text{is}}(v, \mathbf{p})N(v, \mathbf{p}) + \tilde{O}(v, \mathbf{p})N_{init}}{N(v, \mathbf{p}) + N_{init}}. \tag{4.6}$$

This second, adaptive step also calculates new weights $w_{\text{is}}(v, \mathbf{p})$ for each view v, which are used to update the weighting function estimate $w(v, \mathbf{p})$, similar to Equation 4.6:

$$w(v, \mathbf{p}) = \frac{w_{\text{is}}(v, \mathbf{p})N(v, \mathbf{p}) + \tilde{w}(v, \mathbf{p})N_{init}}{N(v, \mathbf{p}) + N_{init}}. \tag{4.7}$$

The example in Figure 4.5 demonstrates how a significant reduction in the total number of samples and the respective rendering time of the AO buffer is achieved, without significantly affecting the image quality. Since more samples are shifted towards more reliable views in terms of depth variance and view orientation, occlusion variance due to view selection is minimized. The occlusion-driven adaptive reduction of the number of samples on the other hand, introduces some noise (see Figure 4.5(g)), which is however suppressed in the post-filtering stage. In this particular example, the Alchemy AO algorithm was used with a 15-tap rotating kernel and time measurements correspond to rendering 1 megapixel of the

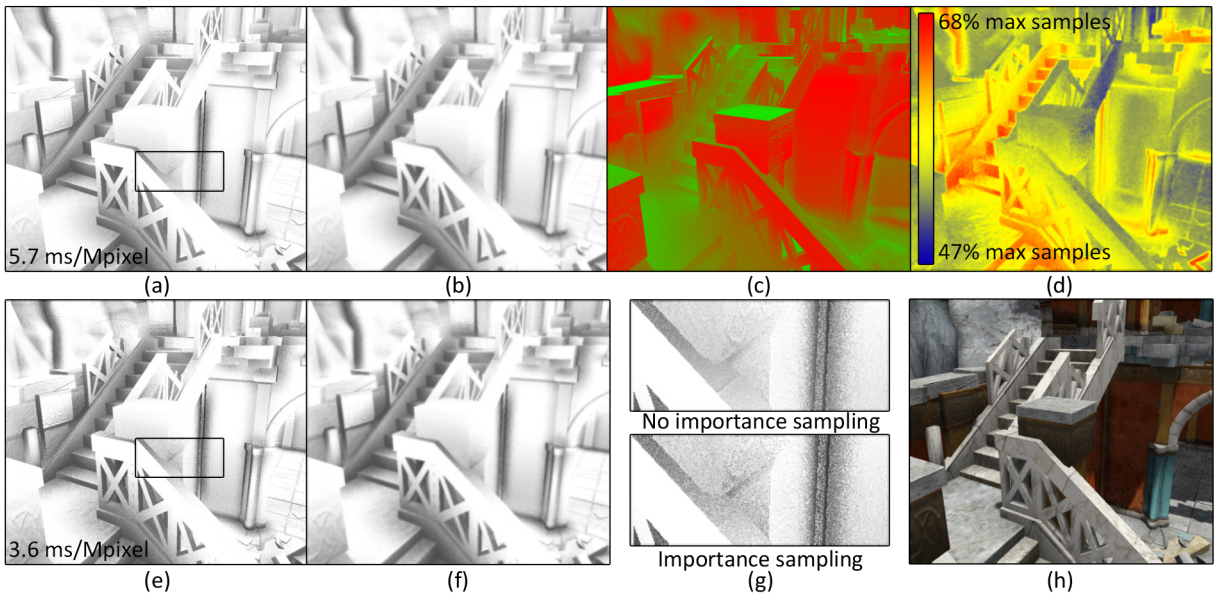


Figure 4.5: MVAO with importance sampling (IS) using the Alchemy AO variation. Example using the camera and the main light shadow map. (a,b) Raw and filtered obscurance without IS. (c) view weights: red denotes camera weights, green corresponds to light view. (d) Number of samples with respect to maximum samples. (e,f) Raw and filtered obscurance using IS. (g) Close up view of the highlighted region in (a) and (e). (h) The final rendered view.

AO buffer with 100% geometry coverage. The joint bilateral filtering time to produce the smoothed AO buffers is not included. Both the filtered and unfiltered buffers are provided for comparison (a,b,e,f,g). The heatmap of inset (d) indicates the percentage of the total samples used with respect to the maximum number of samples for both views, i.e., 2×15 here.

The partial occlusion results $O(v, \mathbf{p})$ can be estimated and combined in a single fragment shader for the camera view using an image-space AO algorithm $AOEstimation$:

Algorithm 1 MVAO Fragment Shader

1: **for all** views v **do** ▷ init step
2: $\tilde{O}_v(v, \mathbf{p}), \tilde{w}(v, \mathbf{p}) \leftarrow AOEstimation(N_{init})$
3: **end for**
4: **for all** views v **do** ▷ adaptive step
5: Estimate $N(v, \mathbf{p})$ using *Equation* 4.5
6: $O_{\text{is}}(v, \mathbf{p}), w_{\text{is}}(v, s\mathbf{p}) \leftarrow AOEstimation(N(v, \mathbf{p}))$
7: Estimate $O_v(\mathbf{p})$ using *Equation* 4.6
8: Estimate $w_v(\mathbf{p})$ using *Equation* 4.7
9: **end for**
10: Estimate $O(\mathbf{p})$ using *Equation* 4.1

4.4 Experimental Study

Implementing MVAO in a deferred lighting scheme is easy, since the method is a drop-in replacement for a single screen-space AO calculation shader. Since our method is generic, it is applicable to a variety of AO algorithms. In this study, we start with a description of our experiments setup and a discussion on the provided supplemental material of this chapter, in Sections 4.4.1 and 4.4.2, respectively. We continue with an extensive qualitative and performance analysis with the most popular AO techniques,

in Section 4.4.3, and a discussion regarding the generation of *bent normals*, which is an estimate of the average open direction above a shaded point, in Section 4.4.4. Since our method is generated based on readily available views, positioning in small, confined environments is not always trivial. As such, we have also experimented with an efficient positioning of user-defined views for hard scenarios, which is discussed in Section 4.4.5.

4.4.1 Experiments Setup

We ran our tests on an NVIDIA GeForce GTX 570 and report times in milliseconds per megapixel of an AO buffer with 100% geometry coverage. Timings for multiple resolutions are not provided, since our experiments showed that they scale linearly with the pixel count, as expected. Furthermore, since this algorithm operates in image-space, there is no dependence on the geometric complexity of the scene.

4.4.2 Supplemental Material

We provide as supplemental material the shader source code of the multiview version of Alchemy AO [McG+11] as well as a video that showcases the benefits of our method in both open and enclosed environments of arbitrary complexity.

4.4.3 Multiview Ambient Occlusion Variations

We have integrated MVAO with representatives of both open volume and solid angle genres of screen-space algorithms. Figure 4.6 provides a comparison of the AO produced by a camera-only screen-space technique and its multiview implementation, as well as the resulting bent normals and final shading. In these implementations, the distance weighting function is the predominant view selection criterion, while the directional weighting function is required only in techniques, which use the surface normal for the occlusion estimation. Figure 4.7 includes relative timings of the MVAO variations for 2 and 3 views with respect to our single-view algorithm implementations, including importance sampling where applicable. We also report the absolute AO rendering times for our single view versions for completeness. These are only indicative as the relative performance of MVAO to the single-view methods matters.

4.4.3.1 Solid Angle Algorithms

Alchemy AO

First, we have implemented a variation of the recently developed Alchemy Ambient Obscurance algorithm, due to its efficiency and stochastic rather than explicit horizon estimation. The MVAO version of this algorithm produces stable results and the view-dependent artifacts are reduced significantly. In terms of performance, Alchemy AO scales proportionally with the number of views, as shown in the top chart in Figure 4.7. Additionally, in all test scenes, importance sampling was particularly effective in drastically reducing AO buffer rendering times, which in some cases were lower than the single-view Alchemy AO, for the same maximum number of samples (see the “2 + IS” column).

HBAO

Horizon-Based AO produces more accurate radial horizon measurements in exchange for more samples and increased computation time, but for the same reason tends to enhance errors when depth information is inaccurate due to high occlusion. When applied to MVAO, it produces banding in the form of silhouettes in regions, where the variance of the distance weights is high, such as the shadow boundaries, if light views are used. In the latter case however, this does not perceptually affect the final image, since the occlusion transitions coincide with the shadow boundaries. Performance-wise, our HBAO variation is the heaviest algorithm, so the stochastic horizon construction of Alchemy AO should be preferred instead.

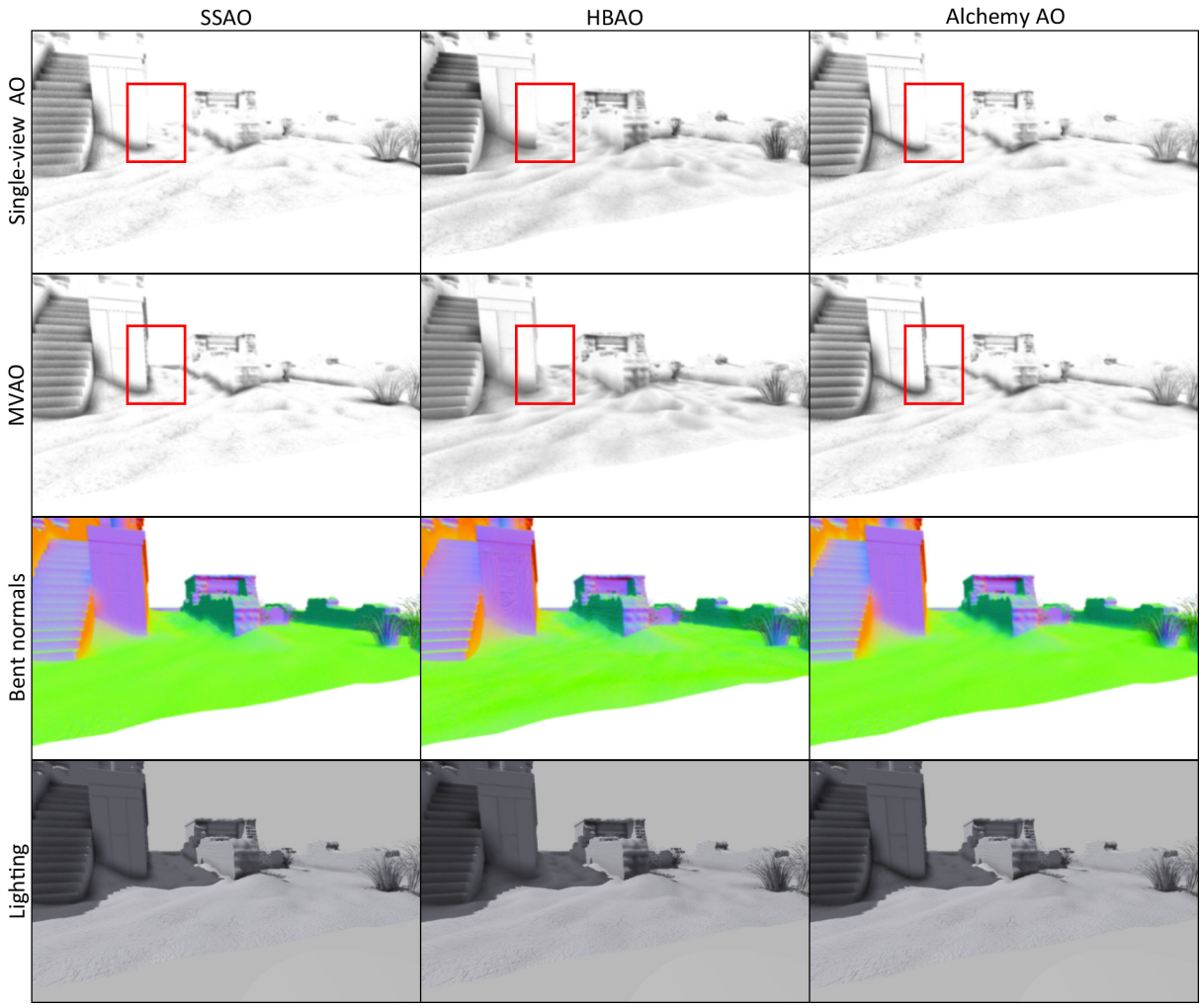


Figure 4.6: Multiview implementations of various AO algorithms. In this example, MVAO uses the camera, the shadow map and a fixed phantom view. The red regions denote areas where our multiview approach has the highest contribution

In our tests, we have found that both solid angle techniques need to favor the distance weighting function ($b \in [0.8, 0.9]$).

4.4.3.2 Open Volume Algorithms

Screen-space Ambient Occlusion

In our implementation of sample-rejection occlusion, we sample the hemisphere above $\mathbf{p}$, instead of the entire sphere centered at $\mathbf{p}$. The depth test mechanism used for the occlusion calculation performs well when adapted to the MVAO scheme, but two modifications are needed. First, the directional weighting function is not necessary as SSAO does not use normals for evaluating the occlusion integral ($b = 1.0$). Second, points on surfaces facing away from the view tend to produce shadowing instead of occlusion. Relying on the distance weighting function to lower the significance of the problematic view would not be sufficient however, since both the shadowed and visible regions with respect to the view can have similar average sample distances. On the other hand, if for all samples that are not visible in the view we assign a distance equal to r_{max} when computing $w_d(v, \mathbf{p})$ we effectively bias the weight against regions in shadow and therefore eliminate the erroneous shadowing.

In terms of quality, multiview SSAO results in stable and plausible images. The performance of multiview SSAO is comparable to that of Alchemy MVAO. However, importance sampling is less effective

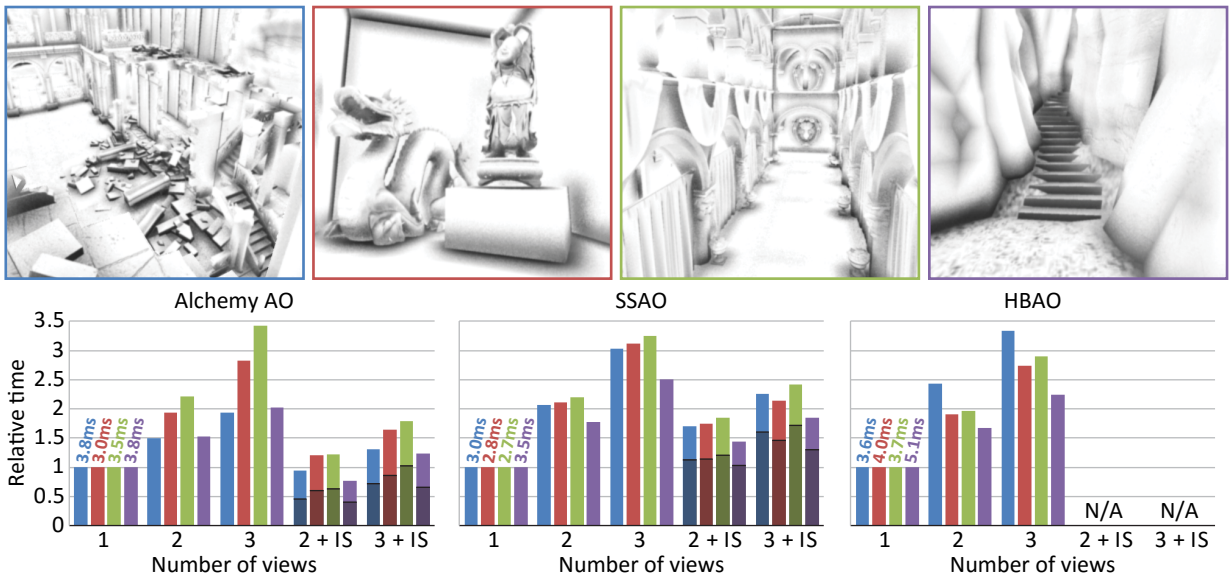


Figure 4.7: Relative timings of MVAO for various scenes (color-coded insets) with respect to the corresponding single view AO. IS denotes importance sampling and the dark part of the bars corresponds to the initial samples taken (5 and 7 samples for the Alchemy AO and SSAO respectively). IS is not available in HBAO, due to its fixed sampling strategy. Alchemy AO and SSAO both use maximum 15 samples, while HBAO uses 6 slices with 4 radial samples per slice.

compared to Alchemy AO, mainly due to higher variance in the initial weight and occlusion estimates. Figure 4.8 demonstrates this fact; both the visual comparison and the RMS difference of the importance sampling results relative to the basic MVAO method reveal that the slower convergence of SSAO increases the deviation of the initial estimates for the occlusion and the view weights, which in turn lead to a slight degradation of the overall occlusion (1.3%) with respect to the Alchemy AO.

Volumetric Obscurance

VO improves the convergence of the occlusion integral by measuring the exact distance between a point on a hemisphere and the projected sample. However, VO is incompatible with our multiview algorithm, since it relies on distance measurements rather than point sampling; if the sphere of radius r_{max} around $\mathbf{p}$ is below the depth horizon, i.e., when all samples in the vicinity of $\mathbf{p}$ are invisible in the current view, VO produces shadows instead of occlusion. When the extra views are light sources, the calculated obscurance accentuates deep shadows and softens the shadow boundaries. Since the results deviate from the intended effect we do not present relative timings for this method. Still, VO can be used as an alternative integrated obscurance/shadow generation technique.

4.4.4 Bent Normals in Multiview Ambient Occlusion

Multiview AO also supports the generation of bent normals using the weighted results from different views. A uniform way to do this with negligible overhead is to accumulate for every view v the vectors from the central point $\mathbf{p}$ to every original sample (not projected on the depth buffer) that is not obscured in v, i.e., lies in front of the recorded depth and thus signifies an open direction. The result is blended with the normal at $\mathbf{p}$. The partial results from all contributing views are weighted in the same manner as the AO values. Figure 4.6 includes bent normal calculation results for the multiview implementations described above.

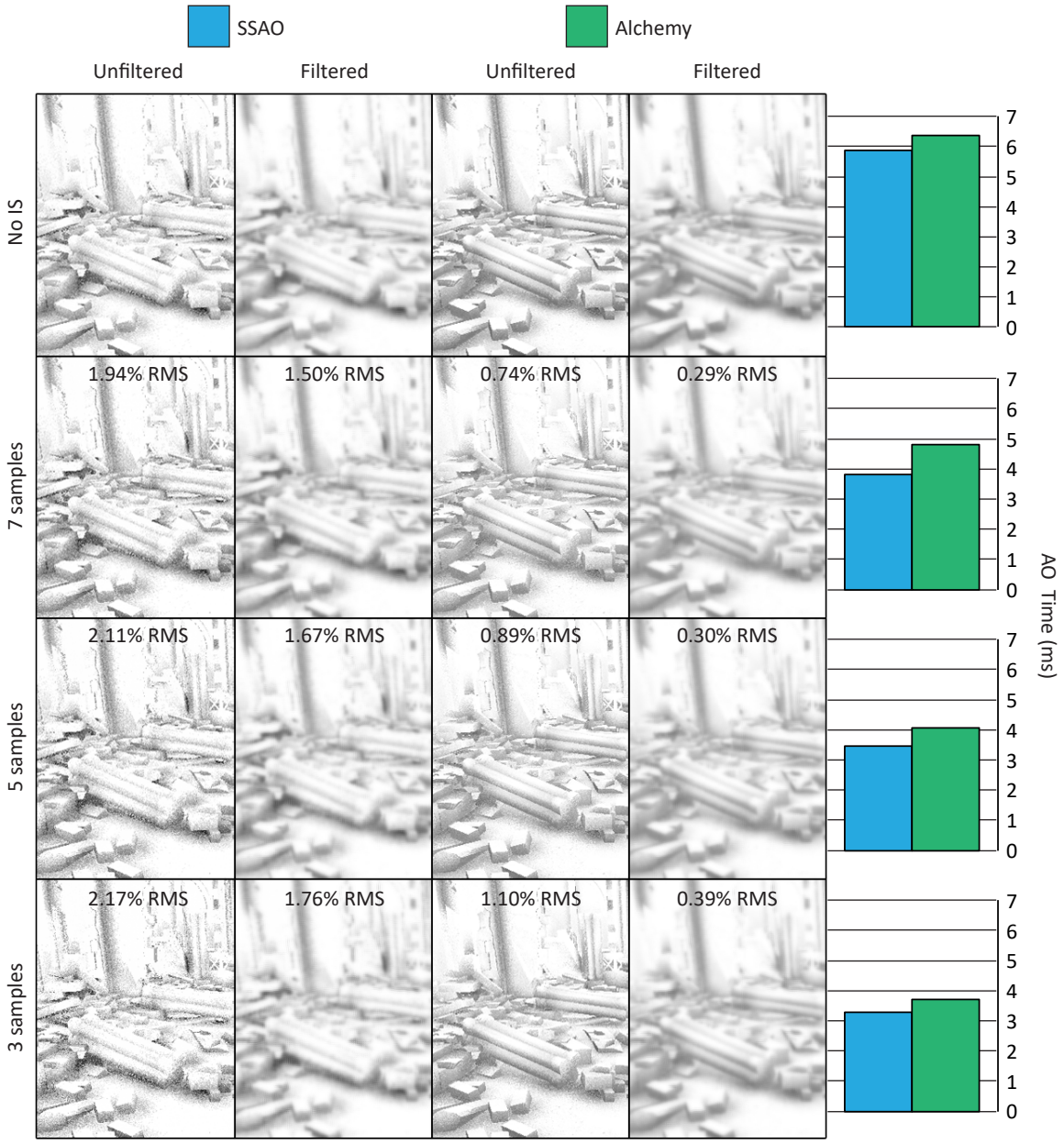


Figure 4.8: Comparison of AO quality for the multiview Alchemy AO and SSAO methods, using importance sampling with different N_{init} number of samples for the first estimate of the occlusion and weights. Corresponding AO rendering time is shown for the two methods for a 1Mpixel AO buffer and 1Mpixel shadow map.

4.4.5 Discussion on View Positioning

MVAO can account for any number of views, but our tests have shown that one or two additional views are usually sufficient for improving significantly the occlusion estimation, even in highly occluded parts of the environment. Adding a large number of views to MVAO is impractical, since the AO computation time could become prohibitive for real-time applications.

For open environments, or parts of indoor environments where shadow maps provide a significant coverage of the geometry, the latter can be effectively used as secondary views (see Figures 4.1 and 4.5). For example, in Figure 4.1, the obliquely projected ground and left wall in the camera view do not contribute significantly to the estimated occlusion in a single-view approach. However, by using MVAO and including the overhead light's shadow map, significant geometric information is introduced and

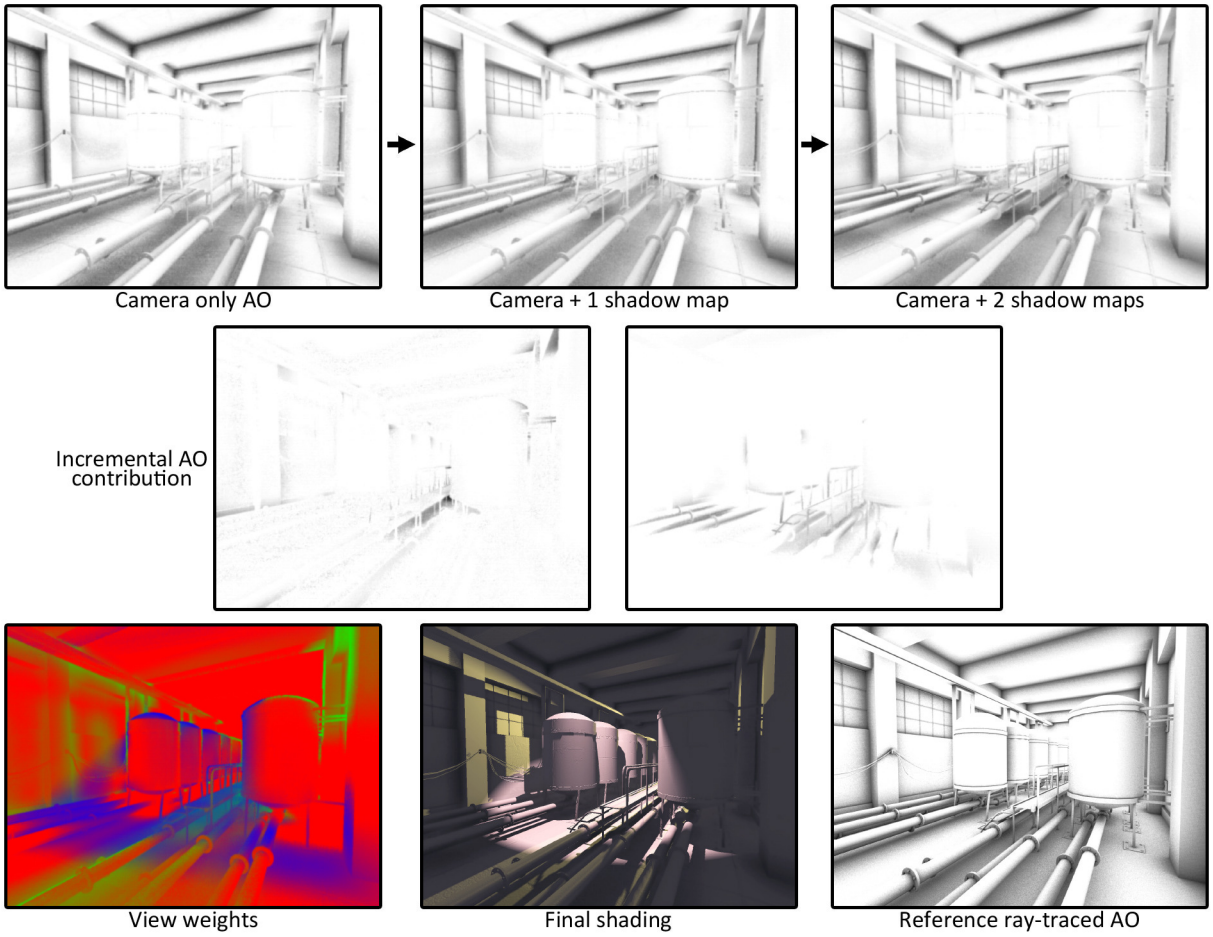


Figure 4.9: The two shadow maps of the scene's light sources are successively added into the MVAO calculation (top row) and the corresponding incremental AO contribution is shown in the middle row. The bottom row includes the weights of the 3 views (R: camera, G and B: shadow maps), the final shaded scene and the reference ray-traced ambient occlusion.

exploited, as demonstrated in the bottom left inset, where the occlusion difference of the single and multiview approaches is shown. It is important to keep in mind that views that do not reveal complementary geometric information are unlikely to improve the result compared to a single-view AO method, as expected. Therefore, such view points should not be included into the MVAO calculations.

In the example of Figure 4.9, a difficult case is provided; most of the information comes from the camera view, while the two shadow maps used do not provide a wide coverage of the camera frustum. Still, accounting for the two additional buffers, helps include missing occlusion and improve its constancy, bringing the final result closer to the reference AO solution.

Two more cases are illustrated in Figure 4.10, where MVAO does not improve the occlusion in parts of the camera view due to lack of coverage of the corresponding regions in the secondary views (shadow maps). The example in the top row uses the shadow map of a light source positioned on the left of the camera view. As expected, missing geometric information at the area on the ground just beside the wall on the right is filled in and occlusion is corrected. However, that was also expected for the occlusion at the corner on the right of the doorway, but the corresponding geometry does not appear in either the camera or the shadow map, since that part of the wall is in shadow. Note that Figure 4.2 demonstrates MVAO in the same scene under similar lighting conditions, but the final occlusion is substantially closer to the reference AO, even in the shadowed regions. This is because in this example, we exploit two light sources with substantially different fields of view, one main overhead light (causing the high contrast shadow) and one fill light. The bottom row of Figure 4.10 demonstrates a hard case where, for the most parts of the

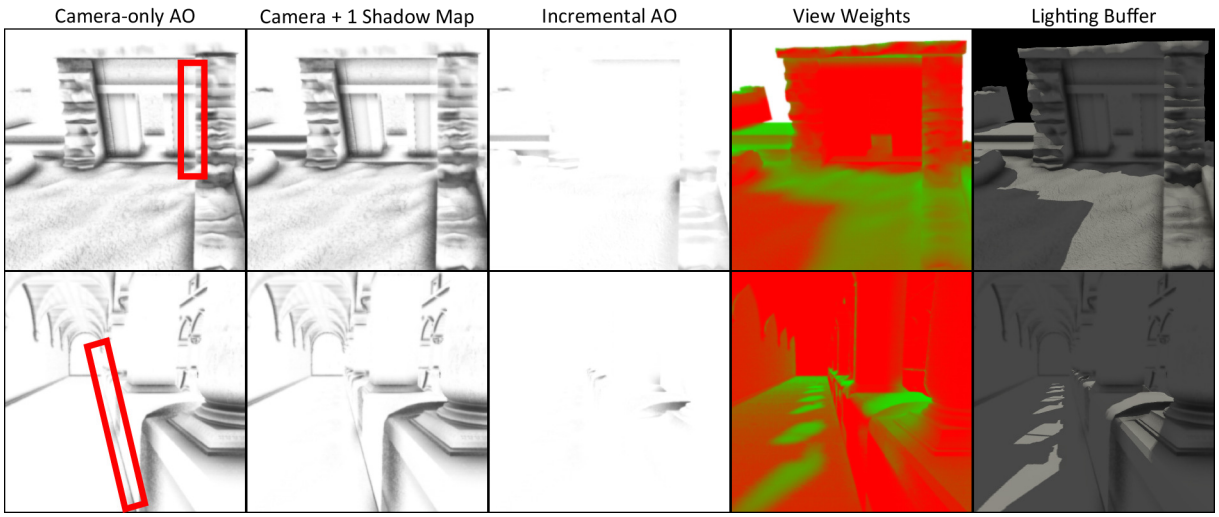


Figure 4.10: The view-dependent artifacts (red rectangles) are not removed since they belong in the shadowed region of the light. The Incremental AO is almost empty and the View Weights rely heavily on the camera depth buffer (red color).

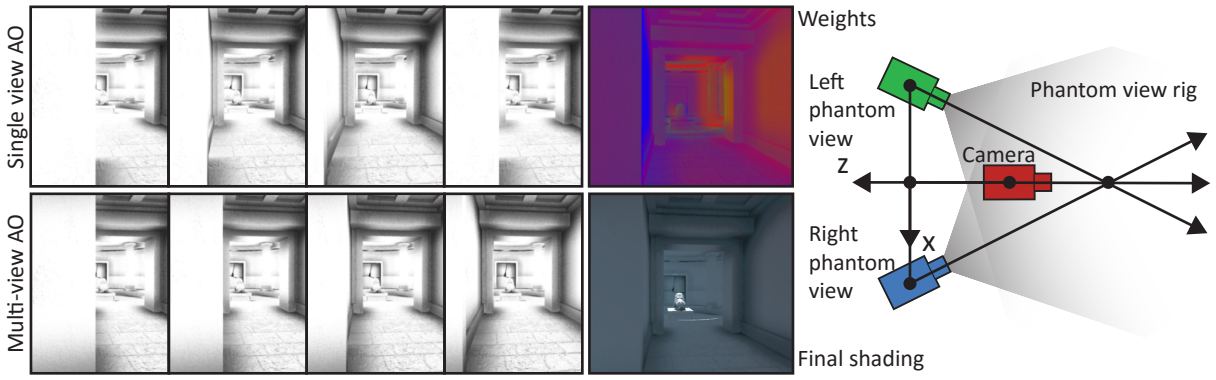


Figure 4.11: MVAO in confined spaces with heavy occlusion. Using an additional pair of cross-eyed low-resolution "phantom" depth buffers eliminates most view-dependent artifacts.

environment, direct illumination from an overhead light source is blocked. Using the shadow map of that light source in the MVAO results in a rather negligible contribution to the overall effect.

4.4.5.1 Phantom Views

For confined spaces and parts of a scene that are not present in a shadow map, instead of using the shadow map(s) as additional views, one may opt to add *phantom* views and render the scene at a low resolution from there. Phantom views can also be set up to follow the user from a different relative view, similar to [RGS09], complementing the visible geometry of the camera viewpoint. We have tested the four-view configuration described in the above paper and found that it is ineffective in a typical indoor environment, where high occlusion from walls and other pieces of geometry obstruct the visual range of additional cameras pointing towards the user's point of interest. Furthermore, the latter is ill-defined in most cases and thus unreliable.

We propose the use of a stereo pair of phantom views arranged as shown in Figure 4.11; the additional low-resolution cameras render the scene from positions behind the user, but at a fixed, close distance so that the possibility of one of the views being obstructed is minimized. Having the look-at directions of the additional cameras cross each other in front of the user, allows for high coverage from significantly different vantage points near the camera, where accuracy is needed the most. The parallel-to-the-ground

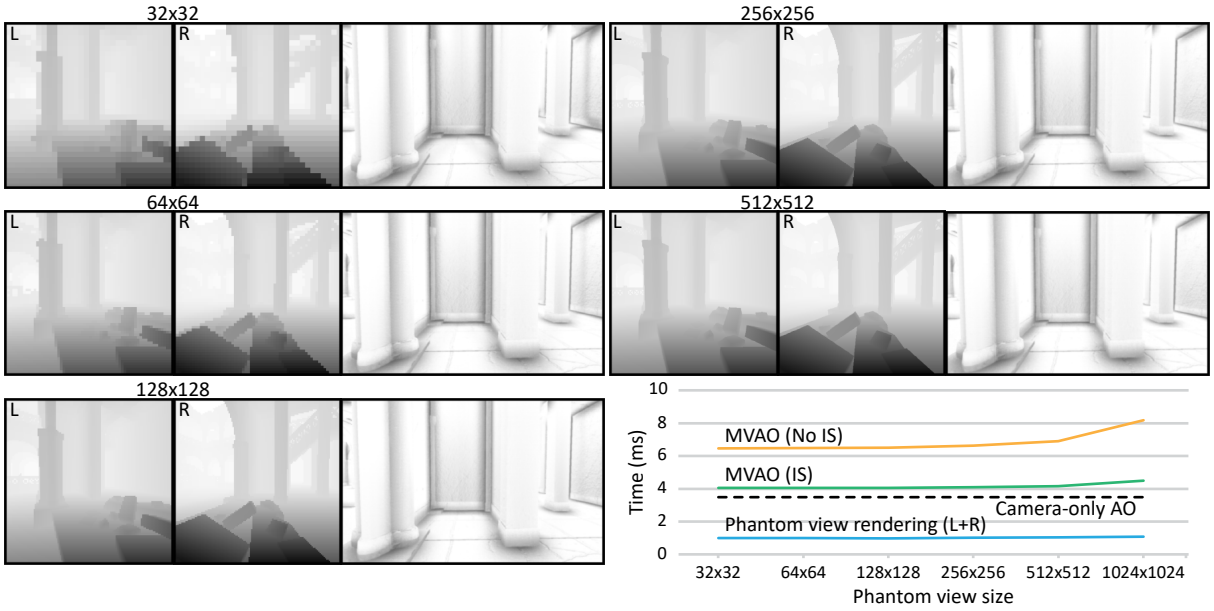


Figure 4.12: Quality comparison between different phantom view shadow map resolutions. The left and center columns of the insets correspond to the shadow maps of the left and right phantom views, respectively. The rightmost column insets refer to the MVAO buffer. Bottom right: Timings for phantom views and MVAO rendering compared to a single-view AO approach.

configuration was chosen having in mind that in most practical scenarios, the view is obstructed by objects whose vertical extents are dominant (e.g., walls, trees, people, etc). Placing the phantom views behind the camera prevents clipping the geometry in front of the camera against their near planes.

For typical scenes, we have found phantom view resolutions between 256×256 and 512×512 to produce satisfactory results (see Figure 4.12). Buffer resolutions above 256×256 remove the artifacts, while maintaining an almost constant rendering time. As shown in the timings included, the extra time required for rendering 2 phantom views and executing the MVAO pass with importance sampling is about 37% greater compared to the camera-only AO. In our tests, increasing the resolution further, did not improve the image quality, while it incurred a performance hit to the MVAO computation time, mainly due to the frequent texture cache misses as a result of the larger image buffers.

4.5 Conclusions and Future Research Directions

In this chapter we have presented a cost-effective solution to the view dependency issues of image-space methods. We weight the contribution of the camera and other views, such as light sources or phantom views depending on the visibility of point $\mathbf{p}$ and its samples, the direction relative to the view and the average distance between the projected samples. Importance Sampling uses the first estimate of the occlusion and weight information to adaptively reduce the total sample count and therefore minimize the overhead imposed by the AO calculations for the extra viewpoints. We show that our algorithm works reasonably well with a variety of techniques and increases the existing visibility information. However, the selection of viewpoints directly affects the completeness of the occlusion information; geometry not properly represented by any of the depth buffers will still produce erroneous, view-dependent occlusion.

Regarding future directions, it would be interesting to investigate the applicability of our generic scheme to other screen-space techniques, incorporating phenomena such as indirect lighting and color bleeding. Furthermore, in the same spirit as [MSW10], temporal coherence can be explored by using previous image buffers from different viewpoints, effectively spreading the multiple views over time instead of space. Finally, the most interesting direction is probably a partially/fully dynamic phantom view placement, taking into account factors such as occlusion by adjacent geometry.

Chapter 5

Real-time Radiance Caching using Chrominance Compression

Figure 5.1: Our radiance caching method compresses the radiance field by retaining the directional resolution of luminance and suppressing the chrominance (left), leading to efficient radiance cache throughput while preserving details such as indirect shadows (middle). This is combined with a reduced cache population scheme, resulting in the efficient rendering of large and detailed environments (right).

Real-time radiance caching methods typically store information about the radiance field on a lattice of cache locations within the scene in order to improve efficiency. When spherical harmonics or other basis functions are used for representing the radiance field, the coefficients of the truncated series employed to sufficiently approximate the original signal can impose an important computational penalty for real-time applications. Additionally, if in-scattering due to transport of indirect lighting through participating media is absent, storing radiance for all grid points in the volume results in wasted computations. In this chapter we introduce the idea of expressing the radiance field in luminance/chrominance values and encoding the directional chrominance in lower detail. We combine the radiance field chrominance compression with an optimized cache population scheme, by generating cache points only at locations that are guaranteed to contribute to the reconstructed surface irradiance. The computational and storage savings obtained allow the use of higher-order spherical harmonics to sufficiently capture and reconstruct the directionality of diffuse irradiance, while preserving finer intensity transitions and maintaining fast and customizable performance. We exploit this radiance representation in a low-cost real-time radiance caching scheme, with support for arbitrary light bounces and view-independent indirect occlusion and showcase the improvements in highly complex and dynamic environments. Furthermore, our general qualitative evaluation indicates benefits for offline rendering application as well.

The chapter is organized as follows: Section 5.1 discusses the general problem and our contribution. Section 5.2 explains the required background work of our approach. Section 5.3 presents the outline of our algorithm, while Sections 5.4 to 5.6 discuss each of our contributions separately. Section 5.7 reports on the efficiency and robustness of our method and discusses limitations and, finally, Section 5.8 offers conclusions and discusses future research directions.

5.1 Overview and Problem Description

Real-time global illumination methods are of particular interest in both the research and industrial community as today's applications strive for visually rich and realistic environments. Due to the hard time constraints posed by real-time rendering and dynamic environments, the costly evaluation of indirect illumination against the entire scene is usually replaced by a gathering operation against a sparsely generated radiance field. In these methods, the incoming radiance is encoded as a radial function, whose coefficients are computed through projection to a truncated series of orthonormal basis functions (such as spherical harmonics). These coefficients are injected into the cells (cache points) of a three dimensional lattice, stored in global memory, and retrieved during a gathering (reconstruction) step at a later stage to compute the indirect illumination for each point in the environment. While the performance benefit provided by these approaches is substantial, the memory bandwidth required for the representation of the radial function during the injection and reconstruction stage can impose a significant overhead in the illumination computations. This becomes even more important in today's large virtual worlds that require a large number of cache points to represent in more detail the incoming radiance. Even worse, methods that ignore participating media may spend a considerate amount of time in calculating and sampling the radiance field in empty space, i.e., in locations that are not near the geometry and do not contribute to the lighting of nearby surfaces, resulting in wasted computations or even incorrectly sampled radiance.

Our Contribution

Inspired by the recent work on chrominance subsampling for frame buffer compression [MP12], *Compressed Radiance Caching* (CRC) introduces the conversion of incident radiance to the YCoCg color space (see Section 2.5) before projecting the directional signal to the spherical harmonics basis. The chrominance components are encoded using lower order spherical harmonics than the luminance channel without significantly compromising the visual quality (see Figure 5.1). This compression scheme is able to minimize the storage, bandwidth requirements and the number of texture fetches, as well as improve the texture cache coherency for real-time illumination computations. Furthermore, in contrast to previous volume-based caching methods, we store incident radiance only near those surfaces, where indirect illumination is gathered for the final shading, but still benefit from the data access and interpolation mechanisms of a volumetric representation. Finally, we exploit the storage and texture access savings of the radiance field compression by implementing a very fast and stable diffuse indirect lighting approach that extends the *Radiance Hints* method [Pap11]. The improved method supports full view-independent indirect shadows in all indirect interreflections and in combination with a geometry-driven cache point selection, achieves real-time performance for arbitrarily complex scenes.

More specifically, our contributions in this chapter are the following:

- Introduction of *directional chrominance subsampling* for radiance field compression, reducing the cache storage and access cost (see Section 5.4). We apply this to real-time diffuse global illumination and demonstrate the resulting performance improvement.
- Optimized positioning of radiance cache points in a volumetric grid, storing the radiance field only near locations where the irradiance is going to be evaluated (see Section 5.5). For single bounce indirect illumination, this corresponds to enabling cache points only near points visible to the camera. Due to a *occupancy volume dilation* procedure, no view-dependent artifacts occur.
- *View-independent* approximate indirect shadowing for all light transport events, based on a binary geometry volume, which is already constructed for the purpose of the cache point occupancy determination (see Section 5.6).

Note that while our last contribution essentially improves the specific indirect lighting method (Radiance Hints), both the chrominance compression and the optimized cache point positioning are generic contributions and can be applied to other radiance caching techniques.

5.2 Background Work

The Radiance Hints method (RH) [Pap11], which this work extends, combines the grid-based radiance caching of Nijasure et al. [NPG05] with the use of RSM sampling to generate the radiance field at each cache location, thus dispensing with the cubemap rendering.

A reflective shadow map is built for each light source that contributes to indirect lighting. The incoming radiance from RSM samples is measured and an approximate radiance field is encoded as spherical harmonics coefficients and stored at each cache point laid out on a 3D uniform grid. This process is continued in an iterative manner for all contributing lights, accumulating the respective radiance field SH coefficients.

First-bounce indirect shadowing is supported optionally using a view-dependent probabilistic attenuation scheme. In the case of multiple bounces, a second lattice is created for interleaved radiance field update and the new radiance field is estimated by direct energy exchange among the cache points. Secondary bounce visibility is statistically approximated.

RSM Sampling

The radiance field for the first bounce of indirect illumination is estimated by sampling the information stored in the RSM G-buffer of the light sources. Each cell in the shadow map represents a small area light source or a *pixel light* that illuminates the scene as if an actual light source was placed at that particular location. The radiance function (radiance field) $L(\mathbf{P}, \omega)$ at cache point $\mathbf{P}$ can be represented using the spherical harmonics basis functions $Y_l^m(\omega)$ of degree l and order m as:

$$L(\mathbf{P}, \omega) \approx \sum_{l=0}^{n} \sum_{m=-l}^{l} c_l^m(\mathbf{P}, \omega) Y_l^m(\omega), \tag{5.1}$$

$$c_l^m(\mathbf{P}, \omega) = \frac{n_{\text{texels}}}{\pi n_{\text{s}}} \sum_{k=1}^{n_{\text{s}}} \frac{\mathbf{\Phi_k} V(\mathbf{x_k}, \mathbf{P_k}) Y_l^m(\omega_k)(\mathbf{n_k} \cdot -\omega_k)}{||\mathbf{x_k} - \mathbf{P_k}||^2}, \tag{5.2}$$

where n_{texels} is the total number of texels in the RSM, n_{s} is the number of samples drawn from the RSM, $\mathbf{x_k}$, $\mathbf{n_k}$ and $\mathbf{\Phi_k}$ is the position, surface normal and stored flux at the k-th sample, respectively and ω_k is the direction $\mathbf{P_k} \rightarrow \mathbf{x_k}$. Finally, the function $V(\mathbf{x_k}, \mathbf{P_k})$ is responsible for the visibility estimation between $\mathbf{x_k}$ and $\mathbf{P_k}$.

The parametric-space positions of the RSM samples are chosen based on a precomputed rectangle of low-discrepancy Halton sequence. The total number of samples n_{s} can either be split uniformly among the light sources or allocated based on any importance sampling scheme. For each RSM sample $\mathbf{x_k}$ drawn, a corresponding radiance "reception" position $\mathbf{P_k}$ at an arbitrary location inside the cache cell is chosen and incoming radiance is estimated along the path $\mathbf{x_k} \rightarrow \mathbf{P_k}$.

Typically, 50-200 RSM samples are used. The resulting coefficients $c_l^m(\mathbf{P}, \omega)$ are stored in the RH cell render target channels and this process is repeated for all cache volume cells.

RSM Importance Sampling

For a large coverage of the scene by the RSM, the radiance can be poorly sampled because of the selection of distant and therefore insignificant samples. For this reason, similar to the original RSM method and the Radiance Hints method, we perform importance sampling by focusing the distribution close to the projection (u_P, v_P) of P in the parametric RSM space. In contrast to the disk sampling approach used in the above, however, we perform a different sampling domain selection; an $s \times s$ square window centered at parametric coordinate (u_c, v_c) is defined, where $0 \leq s \leq 1$ so that (u_c, v_c) is the closest location to

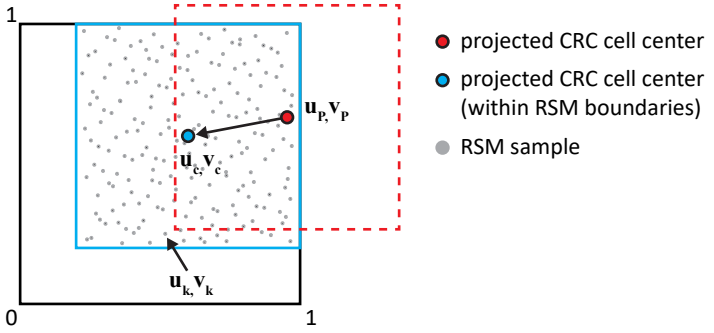


Figure 5.2: The importance sampling window regulated by the sample spread s. The red square shows the sampling area based on the projected center of P in the RSM (u_P, v_P) and the blue square the modified one (u_c, v_c) according to s (0.8 in this case) so that the RSM samples (gray color) do not cross the texture boundaries.

(u_P, u_P) without the window crossing the texture boundaries (see Figure 5.2):

$$\begin{aligned} u_c &= \min(1 - \frac{s}{2}, \max(\frac{s}{2}, u_P)), \\ v_c &= \min(1 - \frac{s}{2}, \max(\frac{s}{2}, v_P)). \end{aligned} \tag{5.3}$$

Samples (u_k, v_k) on the RSM are drawn from the parametric region $([u_c - \frac{s}{2}, u_c + \frac{s}{2}], [v_c - \frac{s}{2}, v_c + \frac{s}{2}])$ by simply scaling and shifting the precomputed Halton samples according to the "spread" s and Equation 5.3.

This alternative importance sampling approach has the following useful properties: it retains the spatial distribution of the Halton rectangle samples, it requires no special handling of samples falling outside the unit rectangle of the parametric space (i.e, rejection or domain folding), it seamlessly resolves to the initial rectangle for $s = 1$ and its PDF is simply the uniform distribution scaled by s^{-2}. Note however that this sampling selection is not unbiased, due to the fact that samples outside the square window have zero probability of being selected.

Secondary Interreflections

For each cache point $\mathbf{P}$ to be updated, n_{sec} sampling directions ω_s are uniformly chosen around $\mathbf{P}$ and the contribution of an, also arbitrarily chosen, cache point $\mathbf{P_s}$ in this direction is sampled. The radiance field is incrementally updated by accumulating the gathered radiance spherical harmonics coefficients at $\mathbf{P}$.

Let $L(\mathbf{P}, \omega)^{(i)}$ be the i-th light bounce, where $L(\mathbf{P}, \omega)^{(0)} = L(\mathbf{P}, \omega)$ from Equation 5.1. Using a uniform spherical distribution of n sampling directions, $L(\mathbf{P}, \omega)^{(i)}$ can also be approximated by a truncated spherical harmonics series, in the spirit of Equation 5.1, as:

$$L(\mathbf{P}, \omega)^{(i)} \approx L(\mathbf{P}, \omega)^{(i-1)} + \sum_{l=0}^{n} \sum_{m=-l}^{l}, c_l^m(\mathbf{P}, \omega)^{(i)} Y_l^m(\omega), \tag{5.4}$$

$$\begin{aligned} &c_l^m(\mathbf{P}, \omega)^{(i)} = \frac{4\pi}{n} \sum_{k=1}^{n} L_{\text{i}}(\mathbf{P}, \omega_k) Y_l^m(\omega_k) = \\ &\frac{4\pi}{n} \sum_{k=1}^{n} L_{\text{o}}(\mathbf{x_k}, -\omega_k) Y_l^m(\omega_k) = \\ &\frac{4\rho_{ave}}{n} \sum_{k=1}^{n} E(\mathbf{x_k}, -\omega_k) Y_l^m(\omega_k), \end{aligned} \tag{5.5}$$

where $\omega_k = \mathbf{P} \rightarrow \mathbf{x_k}$. $E(\mathbf{x_k}, -\omega_k)$ is the estimated irradiance on a virtual reflective surface positioned at the sampled location $\mathbf{x_k}$, which is oriented towards $\mathbf{P}$, i.e., $-\omega_k$. Note that in the above formulation

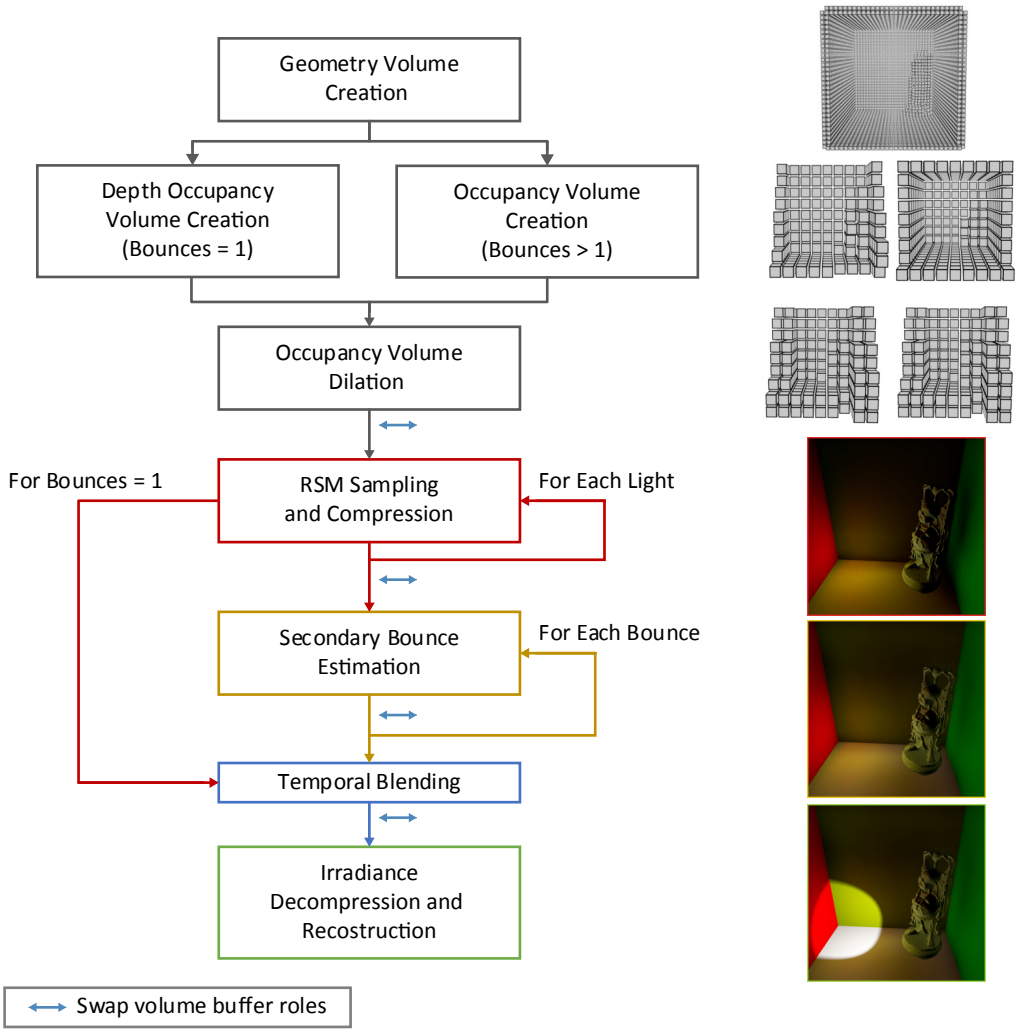


Figure 5.3: The radiance field generation and caching pipeline. The rectangles correspond to separate shader passes.

contributing points are not necessarily the nearest ones to $\mathbf{P}$, nor is occlusion taken into account. Having expressed the radiance field as spherical harmonics, $E(\mathbf{p}, \omega)$ can be efficiently reconstructed using a closed form solution as proposed Ramamoorthi et al. [RH01]. To compensate for the unknown albedo at $\mathbf{x_k}$, a fixed average scene albedo ρ_{ave} is used for the secondary interreflections.

Irradiance Reconstruction

The surface irradiance at the visible fragments is reconstructed as follows: A number of samples are selected on a normal-aligned rotating hemisphere above the shaded point. The fragment's final irradiance value is then estimated by the dot product between the interpolated radiance SH coefficients and the SH coefficients of the oriented hemisphere.

5.3 Method Description

In CRC, a uniform volume grid for maintaining the radiance cache data is created, similar to the Radiance Hints algorithm, with a predefined maximum resolution r corresponding to the longest side of the radiance field bounding box. The CRC volume may occupy the entire scene, a part of it or be centered at the user and translated in voxel-sized increments. To support more than one bounce a second volume is created,

for interleaved radiance field update.

The steps required for determining the position of the occupied cache cells in the current frame, the population of the cache, the compressed encoding of the radiance field in them, the secondary interreflections and the final irradiance reconstruction are outlined in the following paragraphs and summarized in Figure 5.3.

Cache Points Determination

The cache point occupancy information is generated per frame based on a 2D binary voxelization of the scene at a higher resolution (quadruple resolution in each dimension), the *geometry* volume, which is also maintained and used for visibility tests. Then, the *occupancy volume* is generated through a special mipmapping procedure and is also stored as a 2D bit-mask texture. For single-bounce indirect illumination, only cells visible to the camera need to be enabled, so a *depth occupancy volume* is created instead. This can reduce significantly the number of occupied cache points, without any degradation or view-dependent behavior. In a single pass, the appropriate occupancy volume is *dilated* to avoid interpolation errors during irradiance reconstruction. The process is described in more detail in Section 5.5.

First-bounce Radiance Field Estimation and Compression

The radiance field of the first light bounce is estimated similarly to the Radiance Hints method, but the computations are restricted to the occupied cache cells rather than the entire volume. Also, indirect visibility is estimated by ray-marching through the geometry volume, rather than using the depth buffer, thus dispensing with any view dependencies. The first-bounce radiance caching requires one pass and one drawing call for the entire volume per light source. The CRC volume cell fragments are generated with layered rendering and geometry instancing of a volume-sweeping quad. For each light source, the occupancy of all radiance cache volume cells is checked and for occupied cells only, the RSM is sampled and each sample is checked for visibility. The incoming linear RGB radiance from visible samples is transformed to the YCoCg space and compressed so that the luminance is always projected to a 3rd order SH basis and the chrominance values are encoded in 1st, 2nd or 3rd order spherical harmonics, depending on the quality setting of the method. The resulting coefficients are finally accumulated in the CRC cell c data. See Section 5.4 for more details on the compression method.

Secondary Diffuse Interreflections

For the energy exchange in secondary light bounces, two copies of the CRC volume are maintained. In contrast to the Radiance Hints method, where radiance field was evaluated in void space, in CRC we only consider energy exchange among mutually visible, occupied cells. For each occupied cache cell c, n_{sec} rays are marched around the location of c on the geometry volume. Once an intersection is found, the radiance from the hit cache point in the input CRC volume is sampled and accumulated along with the radiance field of the previous bounce in the output volume. For subsequent bounces, the roles of the input and output cache volumes are interchanged and the process is repeated.

Temporal Blending

For dynamic scenes, where the light sources and the geometry can change arbitrarily, flickering of the reconstructed irradiance may occur, which is a defect common to both radiance caching and instant radiosity approaches. To remedy this, a third volume is used, which, similar to the method by Kaplanyan et al. [KED11], contains the exponential moving average of the SH coefficients between the previous frame and the current one.

Irradiance Reconstruction and Decompression

The surface irradiance at the visible fragments is reconstructed similarly to the Radiance Hints method. However, since the radiance field is stored in YCoCg space, a final step is required to convert the irradiance back to the linear RGB space. To avoid interference of cache locations behind the shaded point for thin structures, all samples are shifted by half a voxel in the direction of the surface normal. This, along with the requirement that interpolated radiance coefficients must not be affected by unoccupied cells, are the reasons for the dilation of the occupancy volume by one voxel.

5.4 Radiance Field Compression

An RGB triplet can be decomposed into a luminance component and two chrominance (orange and green) offsets using the YCoCg transform (see also the discussion in Section 2.5):

$$\begin{aligned} Y &= 0.25 \cdot R + 0.5 \cdot G + 0.25 \cdot B, \\ Co &= 0.5 \cdot R - 0.5 \cdot B, \\ Cg &= -0.25 \cdot R + 0.5 \cdot G - 0.25 \cdot B, \end{aligned} \tag{5.6}$$

The lower sensitivity of the human visual system to tonal variations as compared to the luminance gradients is exploited in many compression algorithms for spatially-varying color data. We claim that for typical environments spatial variations of luminance are dominant over chrominance ones. We exploit the fact that the projection of spatial light variations (RSM samples) on the unit sphere remain coherent. This means that chrominance at cache points can be sufficiently approximated by a truncated series of spherical harmonics or other spherical domain basis functions of *lower order than the luminance*. As demonstrated by our diverse examples shown in all Figures, this assumption holds for most typical environments and lighting conditions (see discussion about quality in Section 5.7).

To this end, in CRC, before accumulating the incoming radiance at a cache point, the radiometric RGB values are converted to the YCoCg space. Then, the luminance and chrominance components are projected to the spherical harmonics basis using different orders. Due to the linear form of the transformation, all radiance accumulation and interpolation calculations still hold in the YCoCg space. During the irradiance reconstruction step, the irradiance is converted back to to linear RGB values and used for any shading calculations.

Converting the radiance values to YCoCg space and encoding the color offset components in a lower order can drop the number of coefficients down to 17 or even 11 for $l = 2$, with negligible or acceptable visual quality loss, respectively. We project the luminance in the 3rd order spherical harmonics basis and the two chrominance offsets in either 1st, 2nd or 3rd order spherical harmonics, depending on a quality setting (see Figure 5.4). The reduction of SH coefficients directly affects the number of volume textures to store them. Compressing the chrominance to spherical harmonics with $l = 1$ and $l = 0$ reduces the number of volume textures (rendering targets) to 5 and 3 respectively and has a direct performance benefit.

Specular events require much higher order spherical or hemi-spherical harmonics to be approximated in a sufficient manner. Therefore, we were not able to accommodate the required coefficients in the graphics pipeline implementation of our radiance caching scheme.

A handy consequence of using radiance expressed in the YCoCg space is the direct artistic control of GI luminance (Y) and color bleeding (CoCg) by linear scaling of the respective coefficients. These can be adjusted either in the reconstruction stage (scale factor per polygon or global) or per cache point (via volume- or area-based factor selection).

5.5 Geometry and Cache Occupancy Data

Our approach stores the radiance field only near locations, where the irradiance is going to be evaluated, by keeping track of occupancy information for each cell. This process can reduce significantly the number of occupied cache points and improve the rendering times, especially in open environments. For single

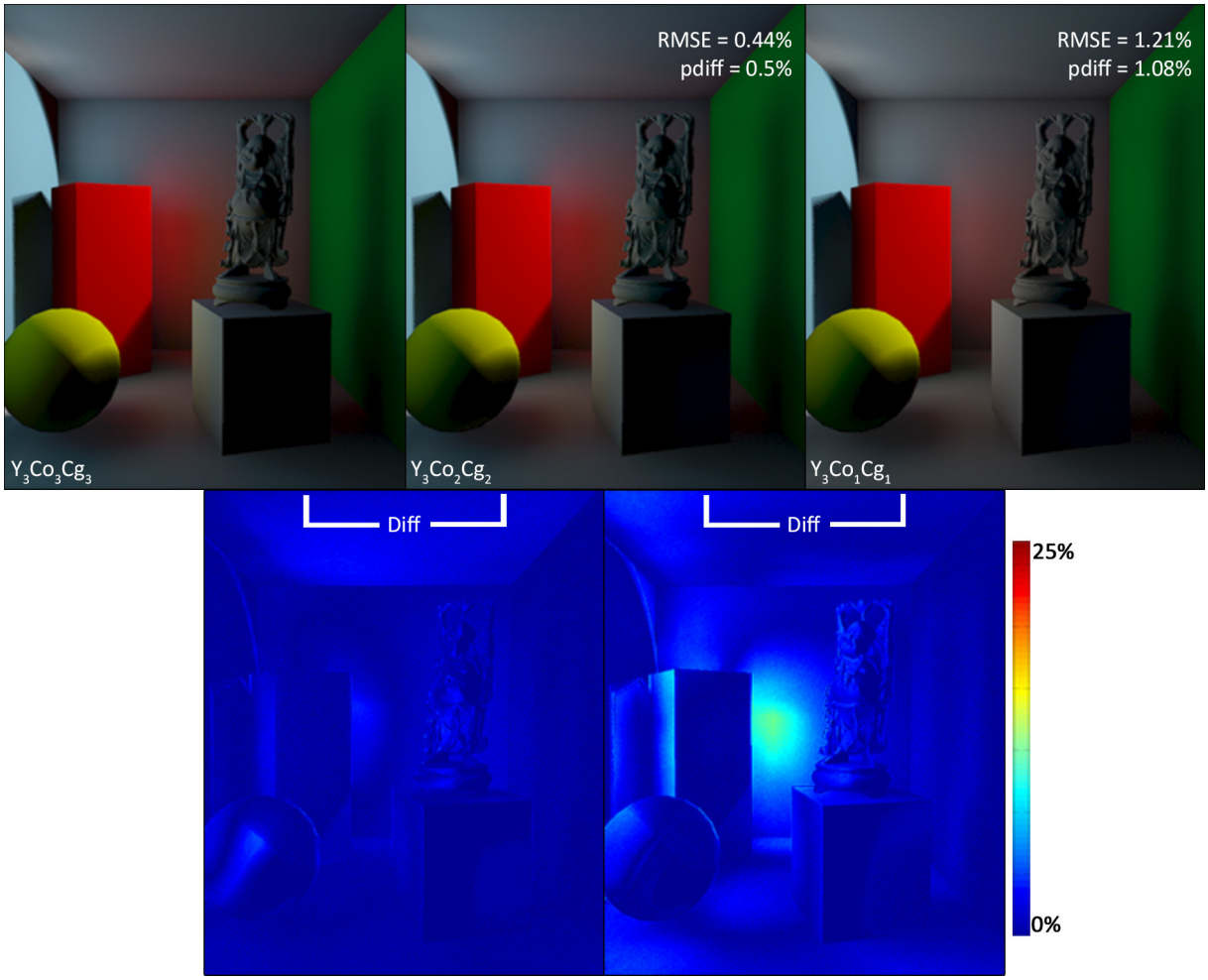


Figure 5.4: The effect of radiance field directional compression in our real-time diffuse-only GI method; luminance transitions are maintained, while suppressing the chrominance detail.

bounce indirect illumination, the cache points are further reduced to only those that are visible to the camera. Also, having this occupancy information available, allows us to reuse it for view-independent indirect visibility tests for an arbitrary number of bounces, highly improving the visual quality of the result.

Initially, the scene is voxelized in a higher resolution volume, the *geometry* volume. This resolution is four-times the CRC resolution. In a second step, the geometry volume is downsampled to create the *occupancy* volume, which determines the *active* cache points, using the same resolution as the CRC grid. To avoid any interpolation artifacts in the irradiance reconstruction step due to incorrect sampling of unoccupied cells, the set of occupied cells is further dilated by one voxel.

Geometry Volume

The geometry volume is generated by a three-way variation of the binary voxelization proposed by Eisemann et al. [ED06]; the geometry is orthographically rasterized in a single pass along the three world-space axes of the CRC volume bounding box and each view is encoded separately as a bit-plane image layer. A geometry shader selectively routes each polygon to the dominant rasterization axis layer according to its normal vector. The partial and potentially sparsely-sampled results are subsequently merged to form a single complete volume.

Our current implementation uses a single 128bit buffer, allowing a maximum resolution of 128^3 voxels. Higher resolutions are of course achievable with a multiple rendering target binary voxelization.

Volume-based Occupancy

The occupancy volume, i.e., the mask of occupied CRC cells, is bit-encoded in a two-dimensional texture, similar to the geometry volume. It is simply a downscaled version of the geometry volume and is produced by generating 2 additional mipmap levels of the latter, using the maximum (OR) operator in a low-cost additional pass. Since the maximum resolution of the geometry texture is 128 in the current implementation, the maximum CRC volume resolution is restricted to 32.

During CRC occupancy checks in the radiance caching and secondary bounce steps, the occupancy volume is considered *dilated* by one voxel. For this reason, after the initial occupancy volume is generated, a fast dilation pass also marks a cell as occupied, if any of its 26 neighbors is also occupied. Since the volume is encoded as a binary mask image, only 9 coherent texture fetches per cell are required in total. An example of the occupancy volume is shown in Figure 5.5.

Depth-based Occupancy

When only one bounce of indirect illumination is estimated, there is no need to cache the radiance field near invisible geometry, since this is never going to be exploited in any irradiance reconstruction calculation. Therefore, we can further optimize the occupancy of the cache points by marking as occupied only those CRC cells that are visible to the camera.

To achieve this, instead of extracting the occupancy from the geometry volume, we inject the camera depth buffer fragments in the occupancy volume using a fixed horizontal and vertical stride s. This separate bit mask texture, the *depth occupancy volume*, is used instead of the full occupancy volume to determine which cache points must be evaluated.

The occupied cache points in the depth occupancy volume can be significantly fewer than those of the geometry-based one (typical cache point reduction up to 90%), resulting in a sizeable speed-up of the radiance cache population.

An example of this is illustrated in Figure 5.6. The depth occupancy is tracked along a certain path as the user navigates from a high-coverage part of the scene (bottom left - similar view to the canyon scene in Figure 5.9), marked as $\mathrm{p1}$, towards the robot model at the top right, marked as $\mathrm{p5}$. As the view closes in on $\mathrm{p5}$, the number of occupied cells is reduced, with a respective drop of the cache generation time. In the current scene, the caching requires 0.54ms at the start of the path with 40% occupancy and it gradually drops to 0.37ms for a 5% occupancy ratio. It should be noted that 0.3ms is a lower limit for this step in our implementation due to the rasterization overhead and the required state changes.

Note that using the camera-based injected points introduces no view-dependent artifacts, since the state of marginally invisible voxels or even voxels missed by the injection is also flagged as "occupied" due to the occupancy dilation.

5.6 Visible Sample Determination

In the RSM sampling pass (first light bounce), visibility $V(\mathbf{x_k}, \mathbf{P_k})$ is determined between the RSM sample $\mathbf{x_k}$ and the CRC cell sample $\mathbf{P_k}$ by ray marching in the geometry volume along the segment $\mathbf{P_k} \rightarrow \mathbf{x_k}$. We offset the start and end position to avoid incorrect self-occlusion checks, and sample the segment sparsely, using a finite number of jittered locations.

In contrast to the Radiance Hints method, for the secondary light bounces, ray marching from the gathering location is used to determine the nearest visible point. Since the contribution of secondary light bounces is less perceptually significant compared to the first bounce, a smaller number of per ray samples is devoted to them. The first bounce uses 8-12 ray samples, while for the secondary bounces, 4-6 samples suffice. A downside of using sparse ray marching is that intersections may be missed. However, the impact of occasionally missed secondary occlusion in typical environments is hardly objectionable and does not justify the computational penalty introduced by increasing the (fixed) number of samples or dynamically determining the samples per ray.

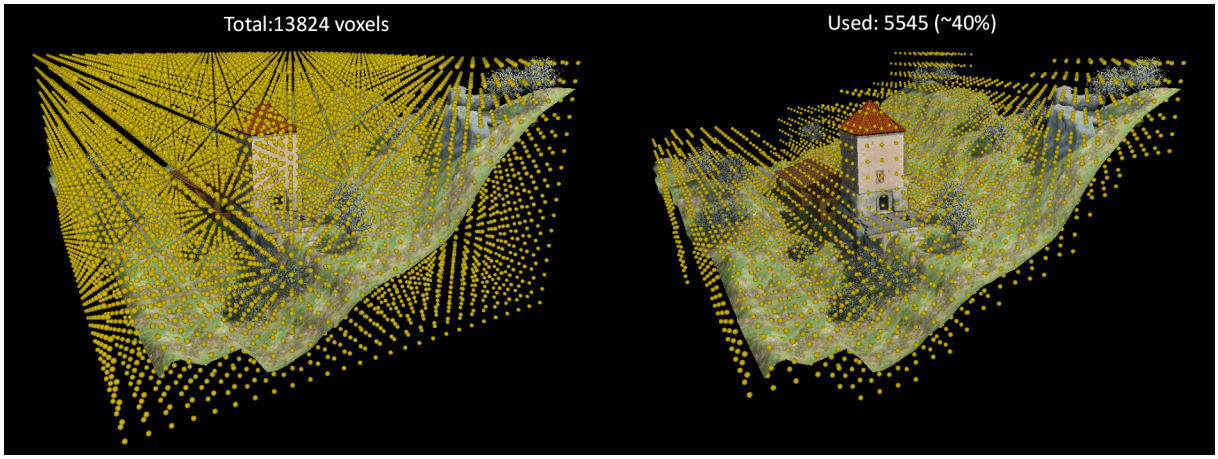


Figure 5.5: Left: A typical scene without the cache point optimization. Right: Employing the occupancy volume can significantly reduce the total number of cache points required (40% in this case).

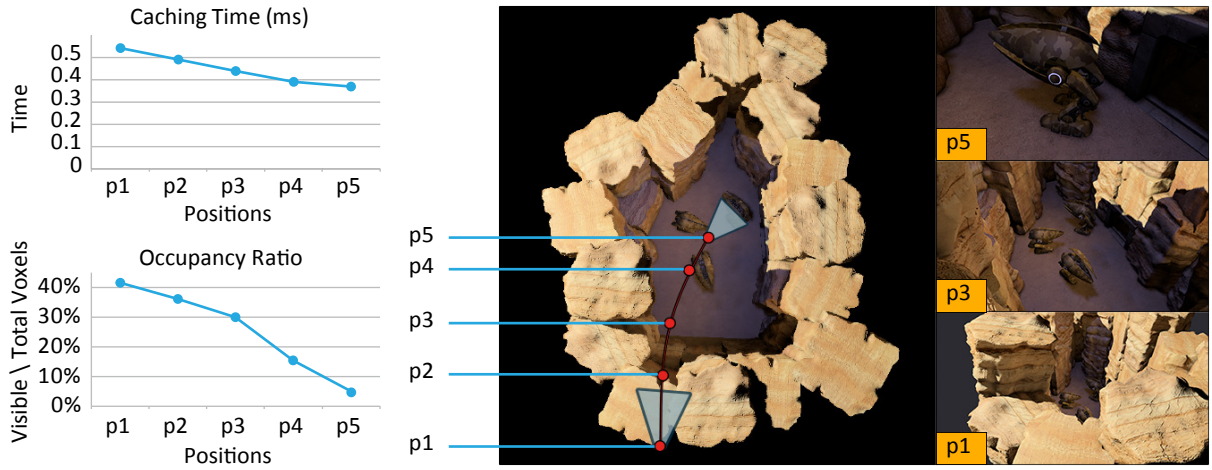


Figure 5.6: Depth Occupancy volume behavior on the canyon scene. As the user moves from p1 towards p5, a gradually smaller portion of the scene is visible. This reduces the number of occupied voxels, resulting in a smaller occupancy ratio and computation time.

Since diffuse indirect illumination is mainly of low frequency and accurate visibility calculations are not necessary [Yu+09], the granularity offered by the geometry volume is usually sufficient (see Figure 5.7). For (near-field) contact shadows, we employ ambient occlusion in some of the example scenes.

We also experimented with hierarchical empty space skipping using the already available geometry volume mipmaps. However, since ray marching always begins at an occupied high-level cell, there is no safe way to make the first leap without losing intersections. Furthermore, any hierarchical traversal would lead to unbalanced thread execution.

5.7 Experimental Study

Our method is able to provide significant performance improvements in arbitrarily complex environments. In this study, we begin with a description of our experiments setup and a discussion on the provided supplemental material of this chapter, in Sections 5.7.1 and 5.7.2, respectively. We continue with a thorough qualitative evaluation of CRC, in Section 5.7.3, where we discuss the quality of our compression scheme, the importance of view-independent indirect shadowing and also report on failure cases. Finally, a performance analysis is presented, where we report on the efficiency of CRC with respect to the

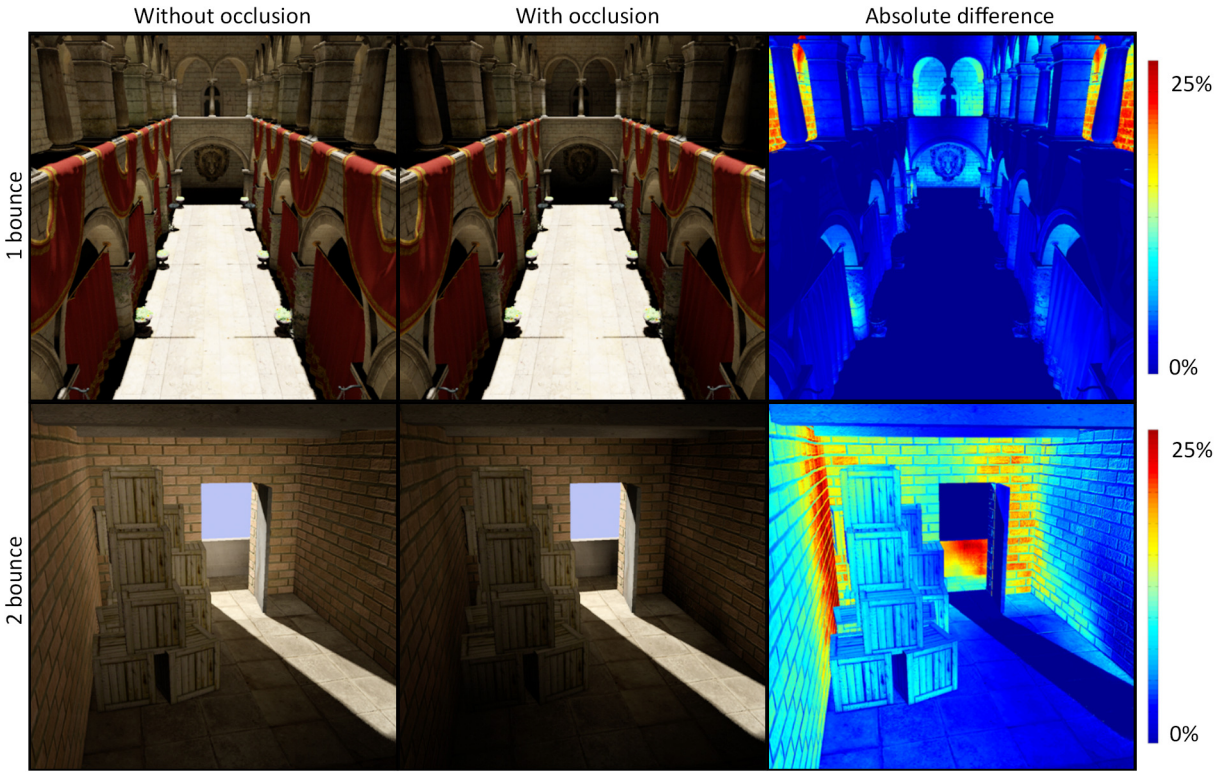


Figure 5.7: Illustration of the indirect occlusion in 1- and 2 bounce indirect illumination.

chrominance compression as well as to our optimized positioning scheme, in Section 5.7.4.

5.7.1 Experiments Setup

We tested our approach on an NVIDIA GTX 670 GPU on various scenes of varying geometric complexity, volume coverage and dimensions. Apart from example scenes with full CRC volume coverage, we have also performed testing for expansive environments, where building an all-encompassing CRC volume is impractical. Instead, GI is restricted to an axis-aligned moving volume centered at the user. The extents of the volume are updated in voxel-sized increments as the user explores the environment. At the extents of the boundaries of the volume, indirect lighting is blended with a constant ambient color. Alternatively, a cascaded approach can be utilized.

5.7.2 Supplemental Material

We provide as supplemental material a complete working demo containing the shader source code of our entire pipeline as well as a video that showcases the benefits of our real-time global illumination method in various environments.

5.7.3 Quality Evaluation

We evaluate the quality of the compressed-chrominance radiance field by both visual inspection and two metrics: the RMS error in the CIE Lab space and the PerceptualDiff metric [YN04] with respect to the uncompressed linear space RGB values. PerceptualDiff measures the percentage of different pixels from the reference image. Both measurements are shown in the RMSE and the pdiff rows in Table 5.1.

Order 2 spherical harmonics for the CoCg channel works remarkably well even in extreme lighting conditions, such as the one presented in Figure 5.8. Keeping only the constant term for the CoCg ($l = 0$) results in no noticeable color shifting of the GI in the majority of the scenes and lighting conditions (see Figure 5.9), while the rendering times are decreased by 28-44%.

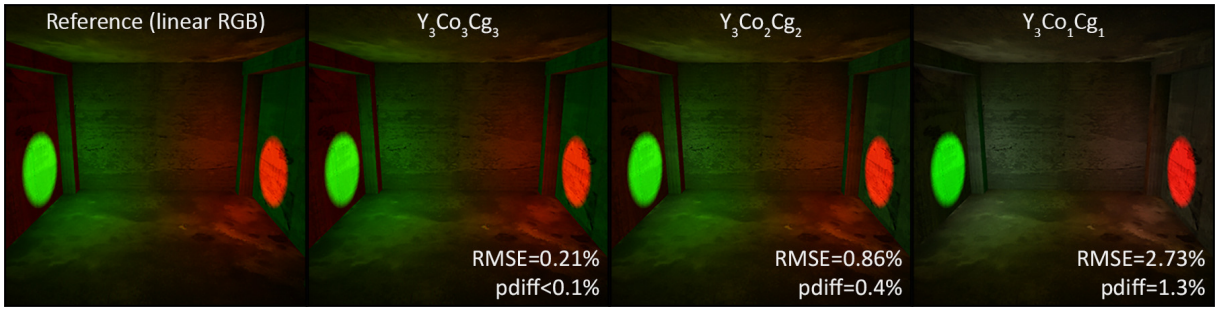


Figure 5.8: Radiance field compression. An extreme case of a red and green spotlight illuminating the scene in opposite directions. From left to right, top to bottom: linear RGB space (reference), YCoCg space with no SH order reduction, 2nd and 1st order (constant) SH chrominance. The subscripts denote the SH order.

Figure 5.9: Screenshots on several scenes. Left: (1 bounce) Canyon, Ruins, Barn. Right: (2 bounces) Factory, Level1, Temples (moving volume). The radiance field is compressed using the $\mathrm{Y_3Co_1Cg_1}$ setting for all scenes.

Corner cases, where the tonal variations in the original signal are of higher frequency, may arise. In these situations, chrominance cannot be efficiently reconstructed especially in the Y3Co1Cg1 compression, where the chrominance components are represented by a constant color. These are manifested either as desaturation of the light field at the problematic cache cells or the predominance of the strongest chrominance value of the incident radiance. The former is shown in Figure 5.8, where two lights sources illuminating the scene in opposite directions. Chrominance is neutralized at cache points in the middle of the room as they gather incident radiance of a different color from opposite directions. The color predominance can be observed in Figure 5.10. In the bottom row, where the light field consists of orthogonal hues from different directions, notice the prevailing orange tint on the nearest columns and arches (indicated by the red arrows) and the color desaturation on the ceiling in the $\mathrm{Y_3Co_1Cg_1}$ inset.

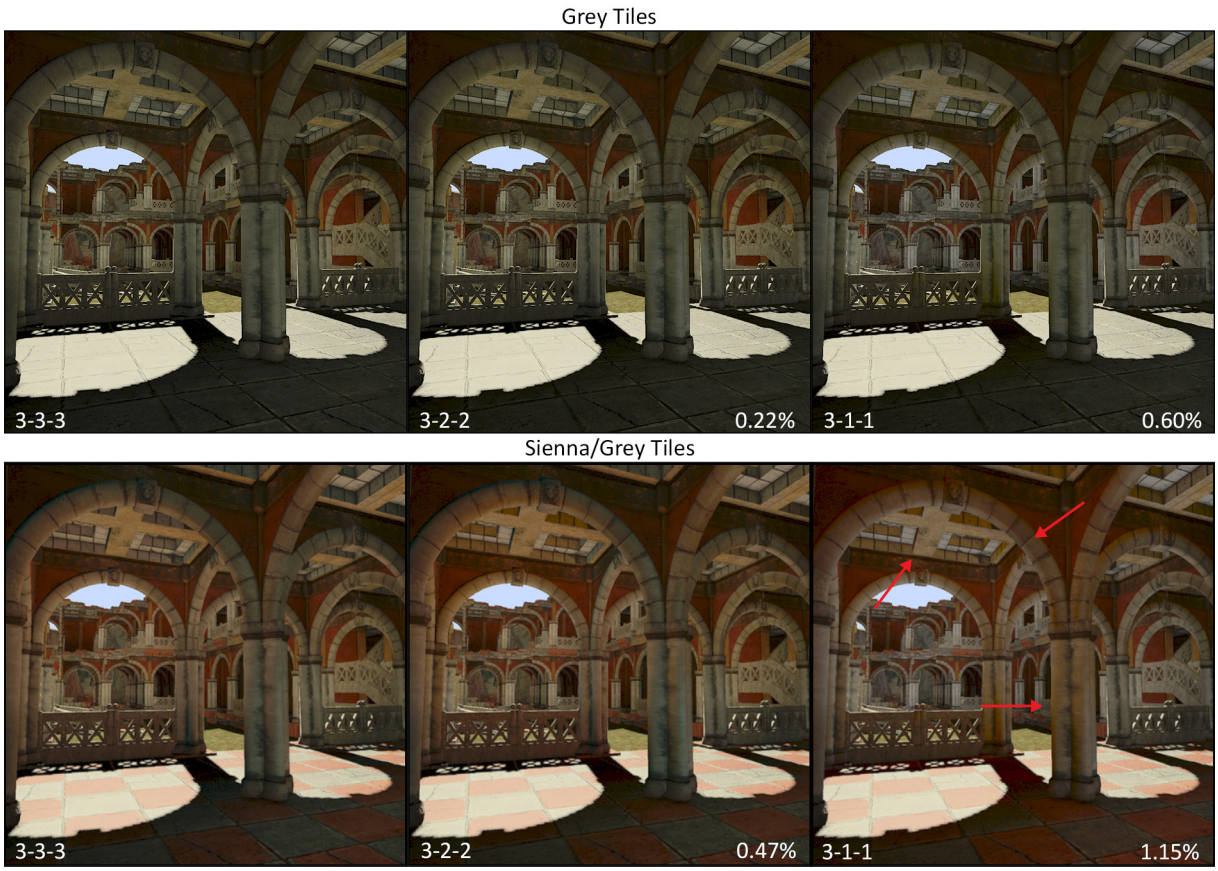


Figure 5.10: Perceptual evaluation of chrominance directionality in variable GI conditions. Percentages indicate the RMS error compared to the uncompressed lightfield. Light is reflected off the floor (top: grey, bottom: sienna) and enters the building as well, reflected off grass and stone. The red and orange arrows denote the observed orange color predominance due to radiance averaging and color desaturation respectively.

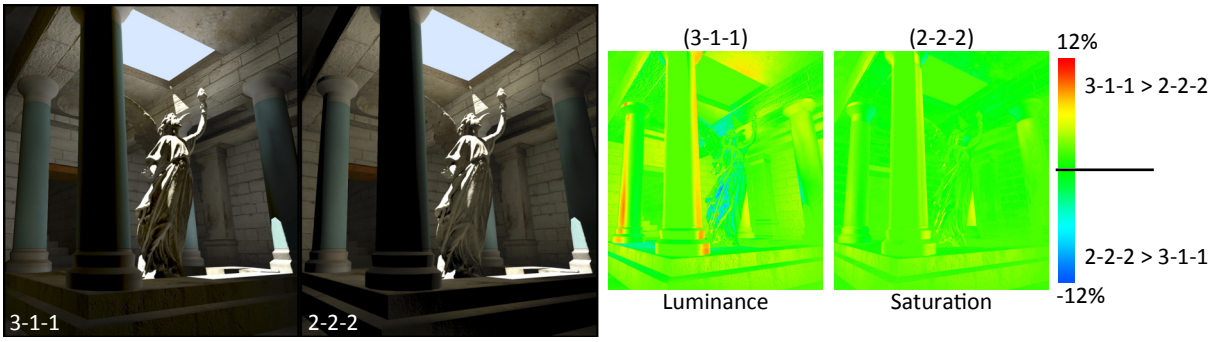


Figure 5.11: Comparison of similar budget radiance field encodings indicates more severe errors when luminance and chrominance are equally but inadequately represented.

Conversely, in the top row, where the reflected light from the floor is consistent with the rest of the environment, no visible change can be detected. A similar case can be observed on the right inset of Figure 5.8 along the protrusions of the side walls where the directionality of the green and red indirect diffuse color is lost. A higher volume resolution would mitigate the issue, but not remedy it.

For the same memory (and bandwidth) budget, we favor an unbalanced luminance/chrominance representation ($Y_3Co_1Cg_1$ - 11 coefficients) over a balanced one ($Y_2Co_2Cg_2$ or $R_2G_2B_2$ - 12 coefficients). In most scenes, such as the one in Figure 5.11, the error introduced by the 2nd order spherical harmonics in the luminance transitions exceeds the over/under saturation caused by the constant chrominance term

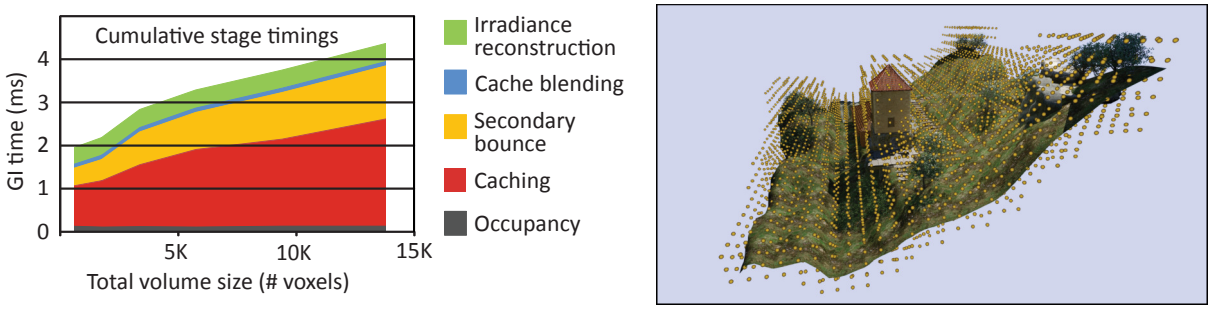


Figure 5.12: Cumulative GI timings with 2 light bounces for the hillside scene (also shown in Figure 5.1) with respect to the CRC volume size. 100 RSM samples were used for one light source.

of the $Y_3Co_1Cg_1$ encoding. Furthermore, the desaturation effect is not usually objectionable or even noticeable, while the over-saturation can be easily treated by scaling down the CoCg coefficients during irradiance reconstruction. Most importantly, on the other hand, problems in excessive smoothing of illumination or leaking caused by the 2nd order luminance SH representation cannot be remedied.

5.7.4 Performance Evaluation

Cumulative timings of the pipeline stages are presented in Figure 5.12 for the "hillside" scene of Figure 5.1 (right), a moderately populated outdoor environment. The maximum coverage of the scene by cache points is also shown above the timings. The factors that influence the performance of all stages except from the irradiance reconstruction, are the total number of RSM samples, the CRC volume resolution (which consequently affects the number of occupied cells) and the number of secondary interreflections. The irradiance reconstruction is only affected by the frame buffer size. All measurements regarding this stage are per MPixel and the rendering time scales linearly with the GI buffer size.

The cost of the occupancy and blending stages depends on the total number of voxels, but is significantly lower than that of the other stages (approx. 0.1ms each). The mipmap generation for the Occupancy Volume and volume injection for the Depth Occupancy volume have a very small cost. The first depends on the CRC resolution, but is relatively constant (0.11ms) and the second depends on the size of the frame buffer (usually between 0.47-0.85ms for stride $s = 10$ and a frame buffer of 1MPixel). It is important to note that the performance benefit of the occupancy optimization has a non-linear relationship to the actual occupancy. This is an expected behavior, since during the caching stage, the unoccupied voxels are skipped using a simple discard statement. If one of the voxels that belong to the same warp is flagged as occupied, the performance increase will be zero. As a result, the occupancy optimization does not perform as well for densely occupied volumes. In our tests, the highest performance gain was 100% in the caching stage. This was reported for occupancy ratios lower than 30%. On the other hand, only a 2 - 10% gain was measured for occupancy ratios higher that 50%.

The performance of our method for some of the scenes of Figure 5.9 is listed in Table 5.1. The voxelization times are given separately, since alternative methods can be used instead. The unoptimized GI time corresponds to no occupancy volume generation (all cells are considered occupied by default) and no radiance field compression. It essentially represents the performance of the original RH method under the current implementation. The other two timings show the rendering time as the occupancy optimization and radiance field compression are enabled and stacked. The Occupancy volume used for 1-bounce indirect illumination is the Depth Occupancy volume, which means that the timings represent the exact viewing position of Figure 5.9. In the case of 1-bounce indirect illumination, the overall speed improvement ranges from 29% to 58%, depending on the visible geometry in the camera frustum. In the case of multi-bounce illumination the overall gain is smaller, due to the fact that CRC cells are flagged as occupied regardless of their visibility. Note that we stress our tests by completely rebuilding the volumes per frame. A more realistic scenario would include action-triggered updates or amortized voxelization and occupancy data generation and dilation, spreading the cost of these passes to 5 or more frames.

Table 5.1: Performance results on some scenes of Figure 5.9. Global illumination frame buffer size is 1MP and the timings are measured on an NVIDIA GTX 670. The scenes with one indirect bounce use the Depth Occupancy Volume, so the occupancy ratio (occupied voxels / total voxels) represents the occupied voxels from the current view instead of the whole volume. Note: The unoptimized time represents the performance of the original RH method under the current implementation.

Scene (indirect bounces)	Canyon (1)	Barn (1)	Level1 (2)	Temples (2)
Polygon Count / Lights	412k / 1	230k / 2	50k / 2	1.56M / 2
Voxelization (ms)	1.44	0.87	0.26	4.96
Res / RSM Samples	16 / 100	24 / 200	24 / 150	32 / 100
Volume Generation (ms)	0.73	0.47	0.12	0.12
Occupancy Ratio	24.13%	42.72%	49.65%	72.67%
Unoptimized GI time (ms) (Occupancy & compression disabled)	2.24	3.3	4.33	2.88
Occupancy Enabled (ms)	1.65	3.12	4.25	2.65
$Y_3Co_2Cg_2$ (ms)	1.52	2.62	3.61	2.10
$Y_3Co_1Cg_1$ (ms)	1.42	2.61	3.10	1.97
Total speed improvement (over unoptimized time)	**57.75%**	**26.44%**	**39.68%**	**46.19%**
	$Y_3Co_3Cg_3$ vs $Y_3Co_1Cg_1$			
RMSE	0.45	0.25	0.45	0.54
pdiff	0.03%	0.32%	0.07%	0.80%

5.8 Conclusions and Future Research Directions

In this chapter we have presented a real-time radiance caching scheme for diffuse global illumination based on the idea of chrominance compression in the spherical domain. The proposed reduction in SH coefficients and the respective number of texture fetches are further complemented by an optimization of the cache point generation in dynamic scenes. Our method is fast, can handle fully dynamic and large environments, requires no precomputations, has no view dependencies and is thus aimed at real-time applications. It is worth noting that even though the proposed radiance field compression and cache population schemes were demonstrated on real-time rendering, they are considered generic strategies and thus, can be applied to any other interactive or offline systems.

The proposed work has opened several interesting future research directions. First and foremost, we can investigate the applicability of radial luminance/chrominance compression to other spherical basis functions. Such an investigation would most likely allow the extension of the current compression scheme to incorporate higher frequency effects, such as glossy reflections. Second, the compression scheme is currently fixed; chrominance is always downsampled. As demonstrated, there are cases that this can lead to low compression quality in environments where color variation is dominant. There, a more generic scheme can be incorporated, which adaptively reduces the required components. Finally, it would be interesting to investigate whether a light transport method including participating media can benefit from the cache population scheme.

Chapter 6

A Multiview and Multilayer Approach for Interactive Ray Tracing

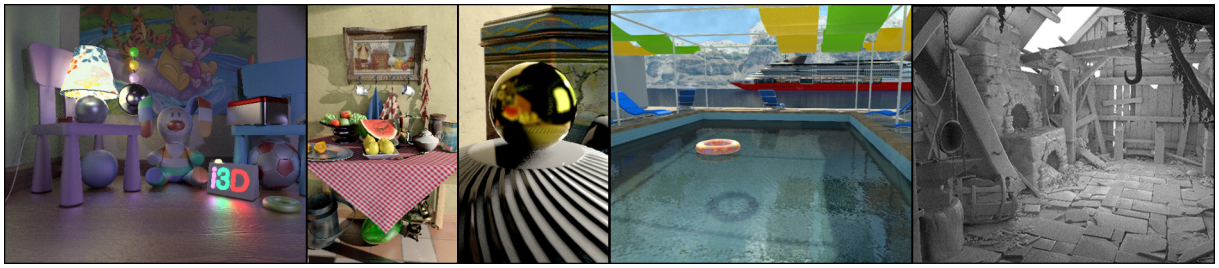


Figure 6.1: Our multiview-layered approach produces visually convincing path tracing and ambient occlusion results in complex and dynamic environments, where both near and distant information as well as off-screen geometry is properly captured.

This chapter introduces a generic method for interactive ray tracing, able to support complex and dynamic environments, without the need for precomputations or the maintenance of additional spatial data structures. Our method, which relies entirely on the rasterization pipeline, stores fragment information for the entire scene on a multiview and multilayer structure and marches through depth layers to capture both near and distant information for illumination computations. Ray tracing is efficiently achieved by concurrently traversing a novel cube-mapped A-buffer variant in image space that exploits GPU-accelerated double linked lists, decoupled storage, uniform depth subdivision and empty space skipping on a per-fragment basis. We illustrate the effectiveness and quality of our approach on path tracing and ambient occlusion implementations in scenarios, where full scene coverage is of major importance. Finally, we report on the performance and memory usage of our pipeline and compare it against GPGPU ray tracing approaches.

This chapter is organized as follows: Section 6.1 discusses the problem in image- and voxel-based ray tracing methods and our contribution. Section 6.2 describes our cubemap A-buffer architecture and generalizes ray tracing in a multilayer and multiview pipeline. Section 6.3 reports on the efficiency and robustness of our method by providing visual results on several global illumination effects as well as a comprehensive analysis and comparison between different multifragment methods and, finally, Section 6.4 offers conclusions and discusses future research directions.

6.1 Overview and Problem Description

The estimation of global illumination via ray tracing methods is a heavy computational process that requires accurate ray-object intersections for a massive number of rays that propagate through the virtual environment. As already discussed in the previous chapters, common CPU and GPGPU ray tracing methods require the construction and maintenance of acceleration data structures in order to achieve interactive framerates, a strategy still not suited for fully dynamic environments, where even the entire scene can potentially change in every frame.

Alternatively, rasterization-based methods, which commonly operate either in image-space or volumetric data, can elegantly support dynamic environments. However, image-based methods are prone to view-dependent artifacts due to the missing information inside and outside the viewing frustum, leading to significant energy loss. An illustrative example is shown in Figure 6.2(a). Front-facing surface information present in the depth buffer and its back-facing or subsequent layers can be efficiently retrieved using single layer (green sides) and multilayer (blue sides) methods respectively. However, surfaces outside the primary view (red sides) are missed.

Conversely, volume-based methods provide a view-independent representation of a bounded volume within the environment and make the assumption that a voxel uniformly represents a neighborhood of geometric elements and corresponding light interactions, limiting the detail of the stored information. As a result, energy exchange is approximate and contact details as well as small-scale surface information cannot be accurately captured (see Figure 6.2(b)), thus limiting the frequency of the properly reproduced phenomena.

A recent category of hybrid techniques attempt to bridge the gap between rasterization-based and ray-tracing approaches (see Section 3.2.4). They can provide efficient results but require multiple data representations and are limited either to tightly bounded scenes or in the type of effects they are able to reproduce.

Our Contribution

In this work, we propose an image-space ray-tracing method that effectively combines the representation completeness encountered in voxel-based approaches with the detailed capture of geometric information of image-space methods, for interactively approximating global illumination phenomena of fully dynamic environments. We perform multifragment rendering in a cubemap configuration in order to efficiently store geometric information in high-detail for the entire scene. This way, view-dependent errors are reduced significantly. Near and distant geometric information is captured by adapting screen-space ray tracing [MM14] on a multiview and multilayered scheme. By moving the computational overhead of a global acceleration structure to a per-fragment basis and relying entirely on the rasterization pipeline, which GPUs are highly optimized for, we are able to efficiently render arbitrary *dynamic* environments of any complexity. Furthermore, the realization can benefit from features such as multisample antialiasing, tessellation, displacement, instancing and primitive deformation.

To this end, we introduce a novel A-buffer that exploits bucketed double-linked lists with decoupled storage. This structure provides fast layer sorting, efficient bi-directional list traversal (with respect to the camera) and empty-space skipping. Since our method is generic, any multifragment variant can be used as a trade-off between performance and memory consumption, as demonstrated by our comprehensive measurements using several common A-buffer variants. Finally, a performance and quality evaluation is provided on path tracing and ambient occlusion implementations in a collection of complex environments demonstrating that our approach is highly suitable for rendering dynamic scenes at interactive rates and fast previewing of a scene during editing.

To summarize, the main contributions of this work are:

- A generic multilayer approach for image-space ray tracing, based on a cubemapped representation of the entire scene that lifts many of the restrictions of previous approaches.

(a) (b)

Figure 6.2: View-dependency problems manifest from missing geometric information inside and outside the viewing frustum in image-based methods (a), while a coarse scene approximation limits the quality of information that can be stored in voxel-based ones (b).

- An A-buffer method exploiting *decoupled* storage and *double linked lists* for fast construction and efficient bi-directional ray traversal suitable for a large number of layers. Per-fragment *empty space skipping* is achieved by arranging the fragments into uniformly divided depth-based buckets.
- An extensive evaluation study providing performance and memory analysis of the most popular A-buffer variants with respect to tracing arbitrary pixel locations on complex environments.

6.2 Method Description

Our algorithm operates in three main steps, as illustrated in Figure 6.3. Initially, we create 6 views, aside from the primary view frustum, using the center of projection of the latter. The near and far planes of each of the 6 view frusta are shifted to ensure full coverage of the entire scene (see Section 6.2.1 and Figure 6.4). Then, generated fragments for each one of the views are subsequently captured via multifragment rendering. Instead of utilizing one of the widely accepted multifragment alternatives, we introduce a new A-buffer structure that aims at maximizing the efficiency of the performance-critical ray tracing stage at the expense of extra memory allocation (Section. 6.2.2). In the final step, our layered multiview structure is exploited to perform accelerated computations on tracing applications by ray marching in screen-space increments and switching between different viewports, when required (Section. 6.2.3). Ray fragment intersections are computed by keeping track of the ray's eye-space Z entry and exit values at each marching step and checking whether the stored fragments are within this range.

Regarding the per-pixel fragment storage, we evolved the bucketed linked-list A-buffer [VF13] to increase the queries performance in the specific case of the ray marching paradigm: (i) An adaptive scheme is explored, where all buckets can handle an arbitrary number of rasterized fragments, avoiding potentially large and unused preallocated storage. (ii) A double linked-list is constructed for fast bi-directional list

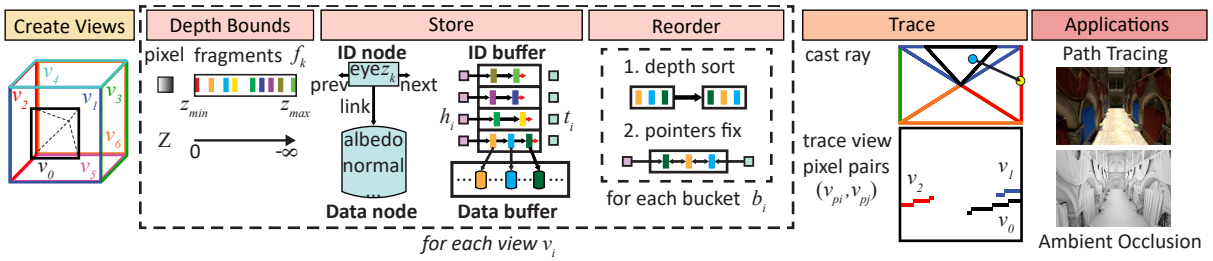


Figure 6.3: Diagram of our ray-tracing pipeline. Camera view and cubemap configuration (left). Construction of our novel A-buffer. For each pixel of every view, depth min/max bounds are computed, geometry is rasterized, uniformly subdivided, stored in decoupled fashion and finally sorted (middle). Views of the A-buffer cubemap are potentially traced for the production of global illumination effects (right).

traversal. In a generic ray tracing setup, the A-buffer is traversed in any order, according to the direction of the rays. The benefits of this modification are especially noticeable in scenes of high depth complexity. (iii) Fragments are assigned in buckets in linear instead of post-projective space. This way, overloading the last few buckets is avoided, a common occurrence due to the perspective projection for large far-over-near ratios. (iv) Depth and shading per-fragment data are stored separately in order to improve both fragment sorting and traversal by significantly reducing the latency of global GPU memory. Note that employing a bucketed scheme provides huge ray traversal benefits by performing efficient empty space skipping at arbitrary pixel locations.

6.2.1 Cubemap Configuration

The first stage of our method is responsible for configuring the necessary view frusta $v_j \in [v_0, \ldots, v_6]$. The first one, v_0, corresponds to the typical camera view with all its associated parameters, such as the field of view (FOV), near/far planes and resolution. Usually, the latter one is set to the frame buffer resolution.

The 6 additional views, $v_1, \ldots, v_6$, are distributed in an world-space axis-aligned cubemap centered at the camera position with FOV $= 90^{\text{o}}$ and square dimensions. This cubemap configuration is simple to implement and results in an overall acceptable distortion. In order to cover the entire scene, the far clipping planes are adjusted to the corresponding extent of the scene's bounding box (left in Figure 6.4).

To cover the clipped volume behind the near planes of the frusta, each cubemap frustum center of projection is slightly shifted backwards according to the near clipping distance. This creates some frustum overlap, but considering typical near clipping plane distances, this results in negligible redundancy. Figure 6.4 (right) demonstrates this modification, where the near plane distances are deliberately set far enough for illustration purposes.

The main advantage of keeping the camera view, at the cost of the extra memory required, is that it disassociates the user's FOV from those of the cubemap faces. Thus, the direct lighting pass can be efficiently executed without casting extra rays between the camera position and the first hit location (unless anti-aliasing or depth of field effects are explored). Furthermore, it allows a downgrade of the cubemap resolution, independent of the geometry captured in the primary buffer, for performance or memory savings.

6.2.2 A-buffer Structure

Once the view information has been generated, our A-buffer structure is constructed in three stages: Initially, we divide the depth range of each pixel p (of each view v_j) into B uniformly consecutive subintervals and assign each span to one $b_i \in [b_0, \ldots, b_{B-1}]$ bucket (*depth bounds* stage). Then, we concurrently store each one of the per-pixel fragments in its bucket by exploring a layered and decoupled data representation that connects nodes via GPU-accelerated double linked lists (*store* stage). A final step sorts fragments by their depth and assigns the appropriate pointers to the correct fragment nodes (*reorder* stage).

Data Structures

Our approach initially requires $2 + 2B$ textures (32-bit unsigned integers) for each view. These include (i) a *depth bounds* texture responsible for storing the min/max depth values $[z_{min}(p), z_{max}(p)]$ of pixel p as well as (ii) B *head* $h_i \in [h_0, \ldots, h_{B-1}]$ and *tail* $t_i \in [t_0, \ldots, t_{B-1}]$ pointers linking the fragments between each view's buckets. To avoid traversal performance bottlenecks that arise from GPU memory latency, fragment attributes are stored in two separate buffer objects, an *ID buffer* whose nodes are connected via double linked lists, and a *Data buffer* (see Figure 6.3). Each node of the ID buffer contains (i) a 32-bit floating point *eye space Z* value (the essential attribute for depth-ordering) and (ii) two 32-bit unsigned integers for the *next*/*prev* pointers between nodes. The index of each ID node serves as a *link*, mapping the corresponding fragment to its shading data node. The remaining attributes required for

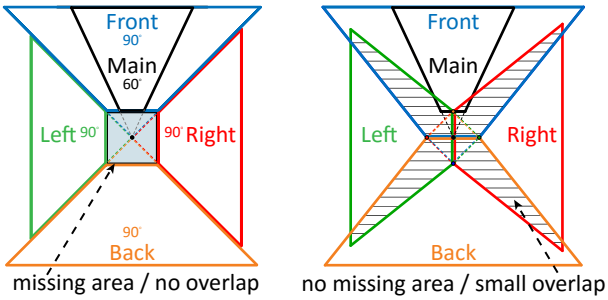


Figure 6.4: Our 7-view camera setup (5 are illustrated for clarity). The clipped volume bounded by the near clip planes (left) is eliminated by offsetting each frustum backwards (right).

shading computations (which depends on the application) are stored in each data node and are read when a ray hit has been identified. Each node (from both buffers) is placed in the next available memory address location according to a single specialized atomic counter (instead of using one counter per bucket [VF13]), resulting in an adaptive memory-friendly fragment storage scheme.

Depth Bounds Stage

Initially, the depth bounds texture is cleared and a quick rendering pass over the geometry is performed by setting depth testing off. This can either be a simplified but accurate depth-only pass over the actual geometry or an approximate and faster pass, where each object is replaced by its bounding box. During the rendering pass, each fragment assigns its eye-space Z value to the depth bounds texture using atomic min and max operations.

Store Stage

A full geometry pass is performed at this stage in order to capture all rasterized fragment information in an unsorted sequence. To avoid executing different geometry passes for each view, the geometry shader is used to emit each primitive to every view. Each fragment $f_k \in [f_0, \ldots, f_{n-1}]$ in a pixel p is mapped to its corresponding bucket by checking the pixel's depth range with the fragment's interpolated eye space Z value z_k such as $b(f_k) = \lfloor B \cdot (z_k - z_{min}(p))/(z_{max}(p) - z_{min}(p)) \rfloor$. The storage location in both ID and Data buffers is pointed by atomically incrementing the *next* counter. An atomic exchange operation sets the next pointer of the ID node to the head pointer, which is afterwards replaced by the current ID node address. Finally, the Data node is filled with the fragment shading information. At this stage, no previous node or tail pointers are assigned since they will be set after correcting depth order at the reorder stage. The head and tail textures as well as the next counter have been initially cleared to zero values.

Reorder Stage

Finally, a full-screen quad rendering pass is initiated, reordering the ID nodes for each bucket via insertion or shell sort. Then, the corrected next/head pointers as well as the newly assigned previous node/tail pointers are set for each fragment/pixel respectively.

6.2.3 Ray Tracing

Image-space Ray Tracing

Once the multilayer cubemap construction has been completed, arbitrary ray tracing is performed in a manner similar to the *single view* 2D DDA algorithm which iteratively samples pixel locations until a ray-fragment collision is detected [MM14]. In that work, given a ray r in 3D space with starting position $\mathbf{x}_r$ and direction $\mathbf{d}_r$, the ray's end point for the current view $\mathbf{y}_r$ is computed by clipping the ray against the viewing frustum. Then, the start and end pixel locations $\mathbf{x}_p$, $\mathbf{y}_p$ are computed by projecting $\mathbf{x}_r$, $\mathbf{y}_r$

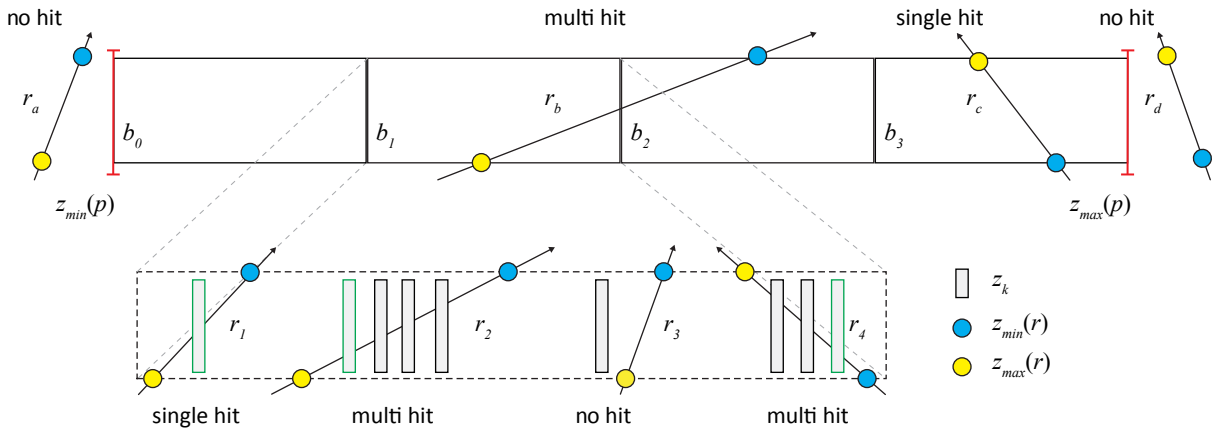


Figure 6.5: Different ray-bucket (top) and ray-fragment (bottom) hit cases. Rays passing outside the pixel/bucket boundaries can efficiently skip fragment traversal ($r_a, r_d/r_b, r_c$). Rays crossing multiple fragments (r_2, r_4) use their direction to determine the hit.

to screen space, respectively. For each sampled pixel location on the segment $\overrightarrow{\mathbf{x}_p\mathbf{y}_p}$, the corresponding eye-space Z coordinates $z_{min}(r), z_{max}(r)$ of the intersection of the ray segment with the pixel boundaries are computed and a hit is found when the Z coordinate of the closest fragment in eye space for this pixel is in the range $[z_{min}(r), z_{max}(r))$.

It should be noted that for performance reasons, a predetermined number of ray marching steps can be used instead of exhaustively visiting consecutive pixels. In this case, samples are jittered using a fixed offset per ray. Alternatively, a hierarchical approach can be explored instead [Ulu14].

For multiple layers (limited to 4 in the original paper), a sequential front-to-back traversal of the fragment list is performed. However, that algorithm does not always exhibit error-free behavior, as the correct hit does not depend only on the ray extents but also on the ray's Z direction (e.g., ray r_4 in Figure 6.5). Returning the correct hit is crucial for illumination computations as these fragments may belong to entirely different surfaces.

To correctly handle fragment lists of any length, arbitrary ray traversals and multiple views, in this work we augment and revise screen-space ray tracing, as explained below. Furthermore, decoupled storage and empty space skipping optimizations are employed in order to alleviate the performance bottlenecks that arise from the different ray-fragment collision tests.

Single-view Ray Tracing

Figure 6.5 illustrates the five different ray-fragment hit cases that arise; a ray may pass outside pixel depth boundaries in either side (r_a, r_d), it may cross one (r_c) or more (r_b) buckets, having zero (r_3), one (r_1) or multiple fragment hits (r_2, r_4). We identify two cases where we can reduce ray-fragment intersection tests; for rays that do not cross any fragment since they reside outside pixel depth boundaries (r_a, r_d), we can perform an *early skip*. Secondly, for rays that do not cross the entire pixel depth range but intersect only a bucket subset (r_b, r_c), we can perform *fragment bucket skipping*.

The fragment intersection algorithm consists of three steps: Initially, we discard rays that reside outside the pixel's depth boundaries by performing a fast interval comparison, where the ray's min/max Z values are tested against the pixel's max/min depth bounds: $z_{max}(r) \leq z_{min}(p)$ or $z_{min}(r) > z_{max}(p)$ (step a). If the ray is not culled, we compute which bucket(s) it intersects in constant time (step b). Finally, we linearly iterate through all fragments in each intersected bucket and check which fragments are within the ray interval. Rays that move towards the near plane are checked in back-to-front order (through the previous node pointer), while rays towards the far plane are checked in the opposite order (via the next node pointer) (step c). In case of multiple hits, the returned fragment is the first one encountered based on the direction of list traversal, given by the sign of the Z ray direction.

Switching between Different Views

In a single view approach, no valid hit is detected if the ray exits the frustum boundaries. In a connected multiple view setting, rays must switch frusta as they cross their boundaries until the far clipping plane of one of them is encountered. A brute-force approach would iterate through all frusta in order to find the next view the ray traverses. However, this simple approach suffers from two limitations. First, as clipping occurs within half-pixel intervals, the ray might not entirely exit the current frustum. This is resolved by snapping the ray to the outside of the viewport by a small offset. Second, if the ray exits through any frustum side (except the near clipping plane), the next view is well-determined and no iteration is required (e.g., exiting the bottom view via the bottom plane switches the ray traversal to the back view). In that case, we switch between views in constant time by keeping viewport edge connectivity information. However, in the very infrequent case of the ray exiting the near plane, an iteration through all frusta cannot be avoided. A multiview tracing example is illustrated in Figure 6.6.

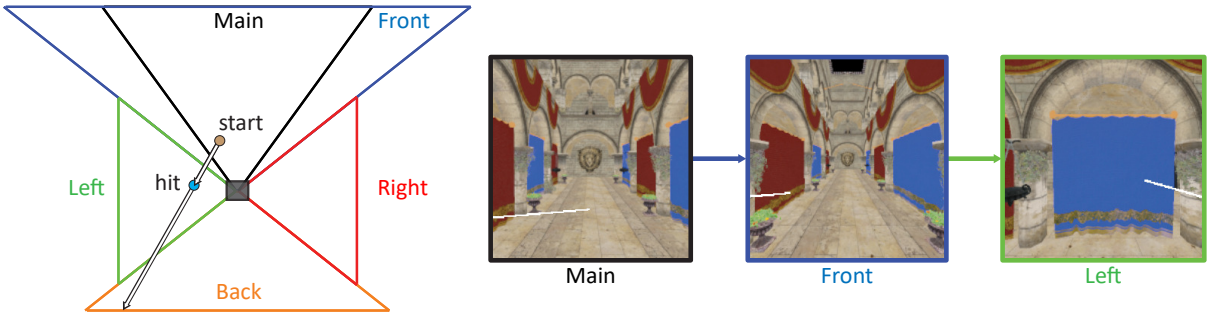


Figure 6.6: Ray tracing example on the Sponza scene. Left: Overview of the multiview frusta. Right: Screen-space ray traversal. The ray, shown in white color, starts on the main view, continues to the front cubemap face and finally finds a hit in the left frustum.

6.3 Experimental Study

We have implemented our approach entirely on the OpenGL rasterization pipeline, which is compatible with common graphics engines. Our experiments setup, as well as the provided supplemental material of this chapter, are described in Sections 6.3.1 and 6.3.2, respectively. Typical illumination algorithms that rely on ray tracing and use information of the entire scene can benefit from our approach. Instead of providing results on isolated global illuminated effects, such as reflections, refraction, etc., we demonstrate our method on a full path tracing implementation as well and as ambient occlusion, in Section 6.3.3. An experimental analysis of our method against general purpose GPU ray tracing implementations is provided, both in terms of quality and performance, in Section 6.3.4. We also compare our specialized multifragment structure against prior A-buffer variants in terms of performance and memory usage for ray-fragment collision queries involving arbitrary pixel fragments, in Section 6.3.5. To our knowledge, such a test has not been conducted so far on a multilayer method.

6.3.1 Experiments Setup

We evaluated our method with respect to performance and quality on a variety of testing scenarios. These include environments containing static geometry as well as rigid and deformable mesh animations. The images were rendered at a resolution of 720×480 for the main view and 480^2 for the cubemap faces on a Geforce GTX 780 Ti. For the stills shown, the number of samples (paths) per pixel was set to 200 and the number of sampling locations for each ray (ray marching steps) was set to 100 for each view, unless specified otherwise. This testing configuration was chosen in order to keep the rendering times as close to the real-time domain as possible. Note that all timings include the direct lighting calculations, which

come at a minimal cost due to rasterization. The number of buckets was fixed to $B = 4$, which provided a good balance between performance and memory consumption.

6.3.2 Supplemental Material

We provide as supplemental material: (i) the shader source code for the construction of our multiview A-buffer with the path tracing and ambient occlusion implementations, (ii) the single view construction of the A-buffer variants shown in Figure 6.12, and (iii) a video that showcases the benefits of our approach in various environments.

6.3.3 Applications

Path Tracing

A complete path tracing algorithm for fully dynamic environments can be easily implemented using our approach. All fragment data needed for shading are stored in the Data buffer. Therefore, material information such as normals, surface roughness, reflectivity and index of refraction are only involved and accessed during shading, without causing a bottleneck in the A-buffer construction and tracing apart. We trace a single ray per pixel using BSDF importance sampling, starting from the nearest visible fragment and trace one additional ray per light bounce to form a path. Since tracing shadow rays comes at the same cost as tracing path segments, we rely on the shadow maps instead. Multiple samples per pixel are accumulated as separate passes, progressively or at a fixed rate. The current number of samples is stored in the alpha channel of the output frame buffer so that adaptive sampling can be easily performed for each pixel individually.

Ambient Occlusion

Implementing ambient occlusion is a straightforward simplification of the path tracing mechanism. For each surface, we send ray samples around the hemisphere of directions and use our tracing method to identify a hit. Again, multiple visibility samples are gathered with multiple one-ray passes. The only change relative to the multiview construction is the information stored in the Data buffer, which requires only the normal vectors. It should be noted that since the shading information stored is minimal, a decoupled representation has a much smaller performance benefit compared to path tracing.

6.3.4 Quality and Performance Analysis

Multiple Views / Multiple Layers

The visual impact of using a layered multiview representation in path tracing is shown in Figures 6.7 and 6.8. A single view and/or single layer setting can exhibit significant energy loss, since visually important illumination is present only in certain views/layers. This becomes even more apparent in scenarios where only a fraction of the scene is available in the primary view (bottom row in Figure 6.7). Note than even using multiple views with a single depth layer may still miss important surfaces (bottom left in Figure 6.8) since the reflected/refracted rays can hit surfaces in the deeper layers. Similarly, Figure 6.9 demonstrates AO on the Smithy model from the Blacksmith environment package of Unity game engine. Note that geometry layers residing outside the viewport can only affect the result with our approach (bottom right inset).

In terms of performance, we observe that applying multiple views comes at an increased cost of $3 - 5\times$ on average, depending on the number of views required for each ray to hit a fragment or exit the scene boundaries. Figure 6.10 illustrates AO and path tracing in scenes with deformable geometry, showcasing the fact that our method is suitable for interactive preview of arbitrarily dynamic scenes.

High-quality renderings (1000spp) of our method are also shown in Figure 6.1, which demonstrate that contact details, distant geometry and off-screen information are properly captured on a variety of

environments. Note that while all scenes are illuminated by one point light source, the Bunny scene (first image) is further lit by emissive geometry. Measurements regarding the performance and memory of these images are shown in Table 6.1. The main observations are: (i) the construction times depend on the overall complexity of the scene, (ii) the tracing performance is directly related to the efficiency of empty space skipping as well as the number of buckets used and (iii) the average number of layers traversed during tracing remains relatively constant, independent of the number of indirect bounces. Regarding the number of buckets, in complex scenes such as the Smithy model, we can further improve tracing performance by increasing B to 8-12, at the expense of the extra construction cost during the resolve stage and the increased memory requirements.

Comparison with GPGPU Ray Tracing

Figure 6.11 shows a comprehensive comparison between our method and a simple but optimized GPGPU path tracer implemented using NVIDIA OptiX ray tracing engine [Par+10a], tested with both SBVH [SFD09] and TRBVH [KA13] acceleration structures. Due to inconsistencies in the lighting models between the GPGPU implementation and our method, the scenes were stripped of any textures, lit with a single point light and shaded with a diffuse-only model.

This test is critical in demonstrating the tradeoff between ray tracing performance and acceleration structure construction speed. Under the assumption of fully dynamic environments, the construction process occurs in every frame buffer update. The images show that our method exhibits comparable quality with respect to GPGPU path tracing, within the limits of the rasterization sampling rate. In

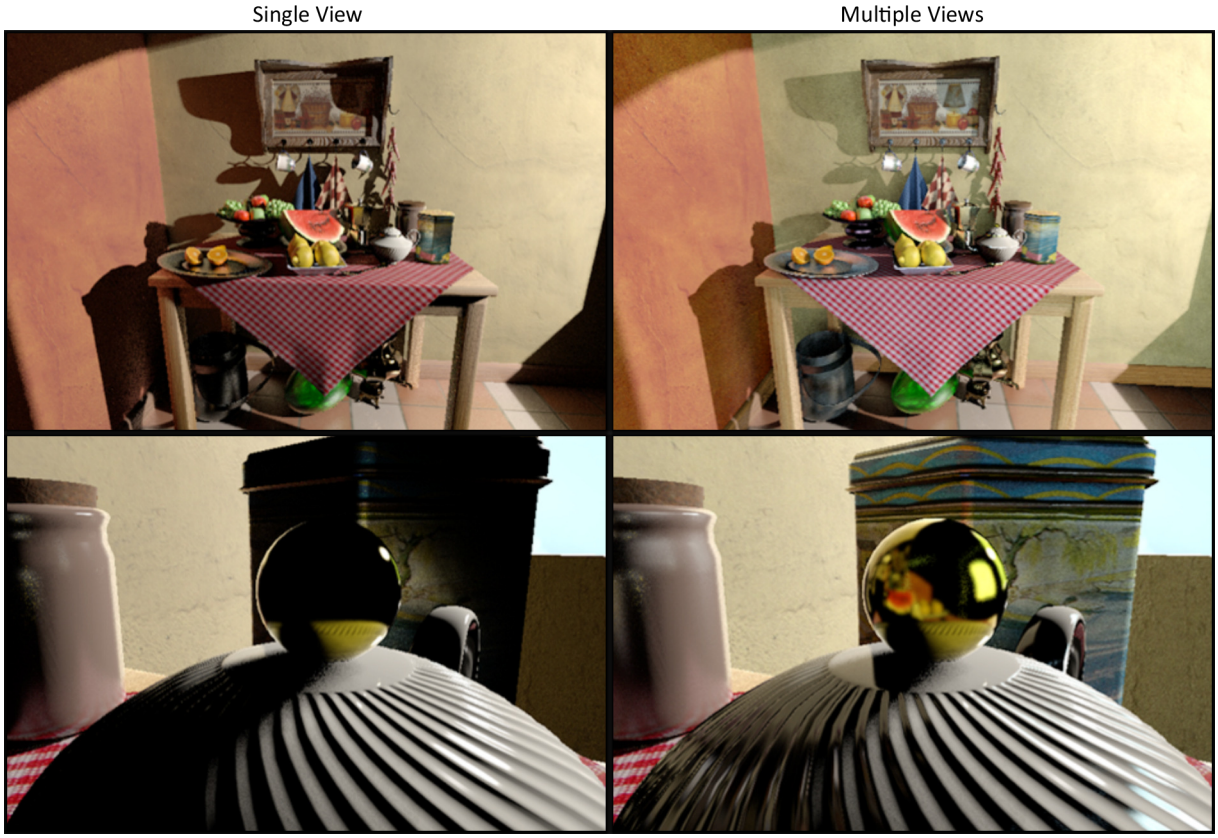


Figure 6.7: 1-indirect bounce path tracing comparison between single and multiple views on the Dead Nature scene. The energy loss in the single view case (left) is clearly shown in both view points, but becomes significant when visually important illumination comes from parts of the scene outside the primary view (bottom). Images rendered at 17ms, 37ms (top) and 20ms, 57ms (bottom) per ray path respectively.

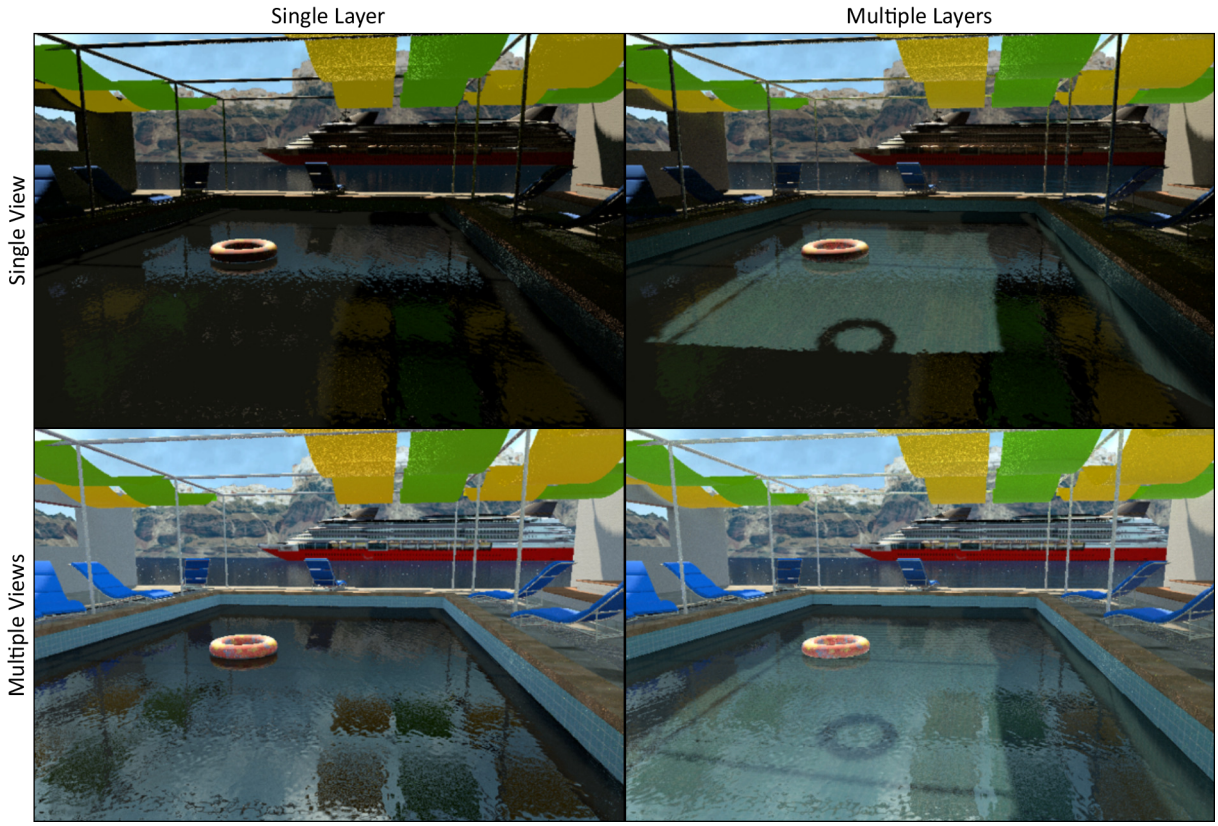


Figure 6.8: 2-indirect bounce path tracing of the Pool scene with different view/layer settings. Even though most refracted rays can be captured with a multilayer approach, reflections require a multiview-layered approach. Images rendered at 30ms, 58ms (top) and 110ms, 245ms (bottom) per path respectively.

terms of performance, a global acceleration structure is naturally faster with respect to tracing rays but takes considerably longer to construct than our method, showing a clear benefit from adopting a rasterization-based approach for applications with dynamic geometry changes (e.g., modeling software).

Table 6.1: 1- and 2- indirect bounce path tracing performance evaluation on scenes with increasing depth complexity.

Scene	Bunny	Dead Nature	Pool	Smithy
Memory (MB)	128	197	241	209
Fragments	2.4M	4.9M	6.6M	9.4M
Avg/max layers	2/17	3/30	4/20	6/49
Construction (ms)	4	12	9	20
	1 indirect bounce tracing			AO (3m range)
Avg layers/views	2/2	3/1	4/2	6/2
Empty space efficiency	98%	89%	44%	77%
1spp / skipping. off (ms)	21/121	37/85	89/141	35/77
	2 indirect bounce tracing			
Avg layers/views	2/2	3/2	4/4	-
Empty space efficiency	98%	86%	47%	-
1spp / skipping. off (ms)	54/284	96/189	245/372	-

Figure 6.9: Ambient occlusion with a range of 3m on the Smithy model with different view/layer settings. Note how the missing occlusion is rectified by including multiple layers (right) and views (bottom). Images rendered at 8ms, 13ms (top) and 28ms, 35ms (bottom) per ray path respectively.

Limitations

Since rasterization is based on discretized samples at specific locations, the information available is limited by the viewport resolution and the pixel samples. Geometry parallel to the view directions is not captured, even with the use of conservative rasterization or multisampling approaches. This leads to rays passing between fragments of sparsely sampled triangles. An example of this can be observed in Figure 6.8, where some of the water reflection rays miss the yellow/green shades and hit the sky dome instead. The problem of sparse rasterization of oblique geometry can be alleviated via triangle tessellation and point sampling (e.g., as in Nalbach et al. [NRS14]).

Furthermore, minor quality issues arise from the quantization of the geometric information and the non-conservative DDA ray traversal algorithm. This can be mitigated by assuming that each fragment is a frustum-shaped voxel of a certain thickness [MM14]. However this approximation adds a relatively small view dependency. Finally, it should be noted that while our method does not distinguish between the type of environments compared to a global acceleration structure, our overall performance is obviously resolution-dependent due to its sequential marching nature, increasing linearly both the memory budget and rendering times. To improve the latter, the hierarchical-z approach to screen-space ray tracing [Ulu14] was also implemented by computing a mipmapped representation of the depth bounds texture. However, since this method requires frequent lod changes and accurate snapping of rays at pixel boundaries, a fixed number of sparse samples is mainly preferred when speed over quality is desired.

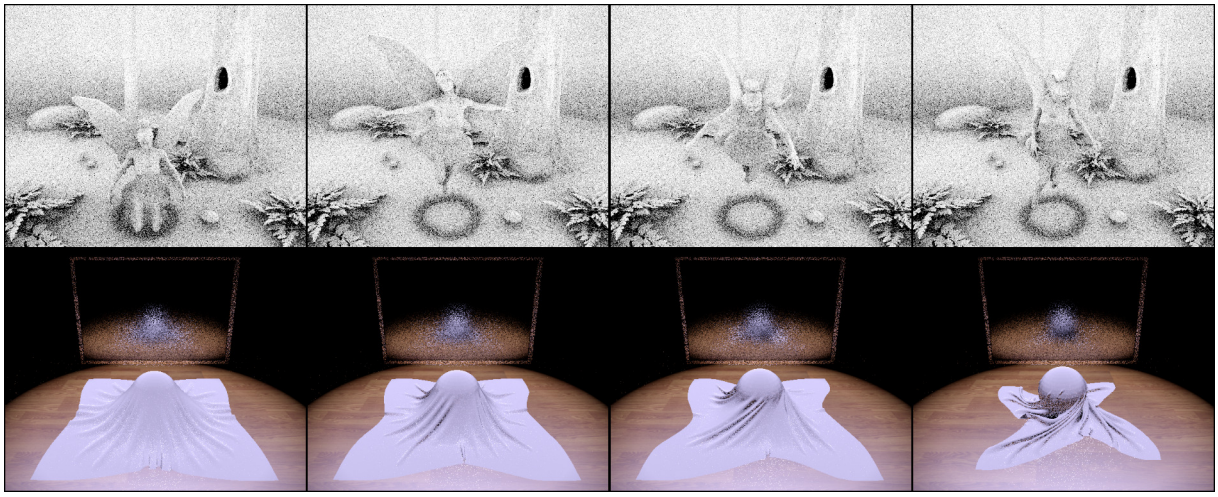
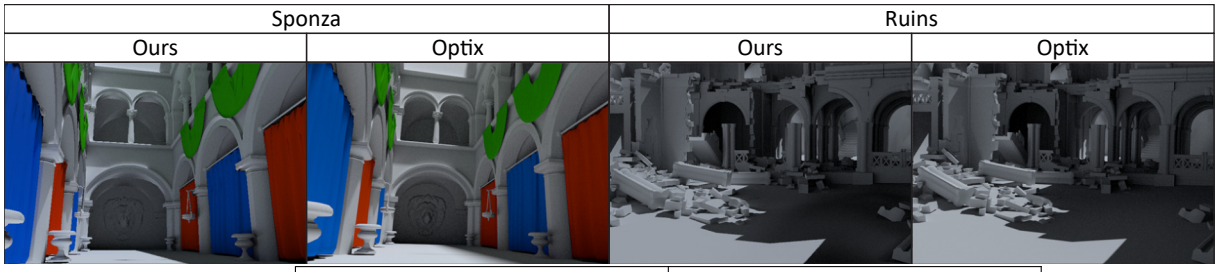


Figure 6.10: Interactive examples demonstrating object deformation in the Fairy Forest (top) and the Cloth-ball (bottom) scenes at 4spp per frame. Average construction/tracing times per frame: 6/89ms (top) and 3/97ms (bottom).



	Construction (ms)			1spp Tracing (ms)		
Scene	Ours	SBVH	TRBVH	Ours	SBVH	TRBVH
Sponza	**10**	2183	148 (15×)	36	**17** (2.1×)	18
Ruins	**18**	4103	1414 (78×)	27	**24** (1.1×)	25

Figure 6.11: 1-indirect bounce quality comparison of our method (left) with an NVIDIA Optix path tracing implementation (right) on the Sponza (top) and Ruins (bottom) scenes at 1000spp. Our method achieves comparable quality results with significantly lower construction times.

6.3.5 Analysis of Multifragment Alternatives

Performance

Figure 6.12 presents an extensive experimental performance evaluation of our approach (DLLB) against competing A-buffer variants in single view configuration when moving from low-to-high depth complexity scenes. Experimenting in multiple views proved unnecessary since the computational cost is proportional. The average/max depth complexity of the three scenes is $3/29$, $7/26$ and $16/96$ respectively. We performed 20 random-access per-pixel operations by artificially generating random rays. If a hit was detected, the stored color value was fetched from the Data buffer. We do not include a detailed comparison of A-buffer construction times, as a small difference between variants was observed. For a fair comparison against our method, we implemented our decoupled scheme and empty-space skipping optimizations on several competing A-buffer techniques, including a double linked lists A-buffer method (DLL).

While decoupling can be adopted by all cases, gaining on average 15% improved construction times, per-pixel fragment queries and empty-space skipping depend on each variant's memory layout. The SB [VF12] allocates fragments using continuous memory segments and traversal can be performed

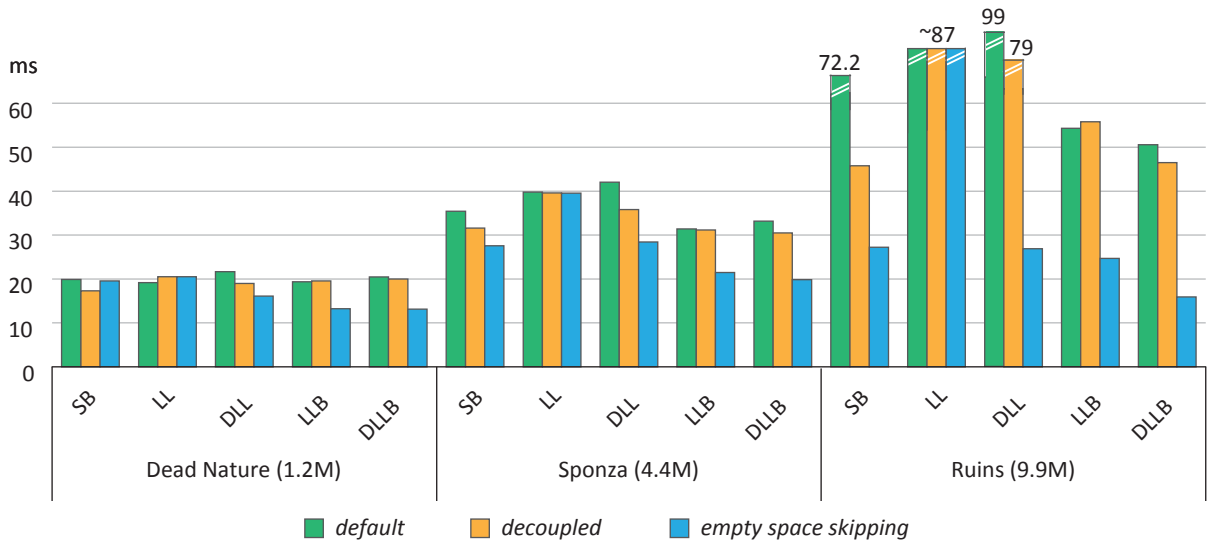


Figure 6.12: Ray tracing performance evaluation of single view A-buffer variants with and without decoupled shading data and empty space skipping on scenes with different fragment complexity at 800^2 resolution. Numbers in parentheses denote fragment count.

by either linear or binary search based on the number of generated fragments. Despite its logarithmic complexity, binary search is mostly preferred ffor traversing large per-pixel fragment sequences (> 16). On the other hand, linked-lists are limited to linear search but can take advantage of bucketing for fast fragment bucket skipping. Culling outside the depth boundaries is available in all methods, except LL [Yan+10], where the early skip on the far side is not available (e.g., r_d in Figure 6.5).

The results showed an average increase for each scene in the decoupled versions by 5, 8 and 17% respectively as well as a further increase of 21, 30 and 210% by enabling the empty space skipping. We can further make the following observations: (i) bucket-based methods exhibit much better performance compared to the rest ones due to the fragment bucket skipping, (ii) bi-directional traversal in conjunction with early space skipping outperforms all other approaches and finally (iii) the average performance improvement is exponential with respect to fragment complexity.

Memory

Table 6.2 presents memory requirements (in bytes) for a wide range of methods that implement a decoupled-based A-buffer behavior. We observe that SB, LL, and DLL outperform LLB and DLLB due to their bucket-free construction nature. Note that our method requires slightly more storage per pixel m_p (B tail pointers) as well as per fragment m_f (previous node pointer) than its predecessor, the bucketed linked-list A-buffer (LLB) [VF13].

Table 6.2: Memory formulation of decoupled A-buffer versions. n_p is the number of fragments per pixel, w_j, h_j are the dimensions of the j-th view and $data_f$ is the per-fragment size of the shading data.

Memory	SB	LL	DLL	LLB	DLLB
$m_p : \forall p \in v_j$	8	4	8	$8 + 4B$	$8 + 8B$
$m_f : \forall f \in p$	8	8	12	8	12
	$+ \text{size}\{data_f\}$				
Total	$\sum_{j=0}^{6} w_j h_j \cdot (m_p + n_p \cdot m_f)$				

Discussion

Considering completeness, we have investigated shifting bucket boundaries in order to achieve uniform fragment distribution inside bins. In this case, however, the ray-bucket intersection computation switches from $O(1)$ to $O(B)$ resulting in a noticeable reduction of the empty space skipping efficiency.

Regarding storage consumption, our implementation suffers from potentially large and possible wasted memory demands due to its strategy to allocate data nodes for fragments that may not be hit. Note that it is not mandatory to generate a complete A-buffer for the main camera view, since this information is already present in the cubemap as well. However, since the resolution of the cubemap faces can be scaled down, high frequency effects exploiting the high-resolution main view will not be captured in the best possible detail.

6.4 Conclusions and Future Research Directions

In this chapter we have presented a complete multi-fragment solution for interactive fragment-based ray tracing. Our method reduces view dependencies significantly as it captures information on the entire scene without the need for additional data structures, supports large and dynamic environments and is able to effectively capture near and distant geometry. This allows the implementation of ray tracing-based algorithms, such as path tracing and ambient occlusion, without being restricted by any surface type, BSDFs or to individual phenomena. A wide spectrum of testing scenarios has been explored illustrating the high-quality images generated using this work. While our pipeline is independent of the underlying multi-fragment technique, our analysis has demonstrated the performance superiority of our A-buffer variant compared to prior solutions. It should be noted that while we have focused on interactive applications, our tracing scheme is scalable; modifying the sampling step has a direct effect on the speed over quality ratio. In conjunction with a bounded multi-fragment method, such as a k-buffer, make our approach applicable to real-time rendering systems as well.

During this work we identified several interesting research directions in order to reduce the two main limitations of this method: the sparse rasterization of oblique geometry and the extra memory consumption required. The work presented in the next chapter is directly focused on fully addressing the aforementioned issues. Still, we can further improve on rasterization-based ray tracing by tackling several other issues: first, the suboptimal acceleration structure can be optimized by exploring adaptive depth subdivision techniques, second, coherence strategies can be explored with respect to ray traversal, and, finally we can investigate the applicability of a multi-fragment rasterization pipeline to efficiently incorporate different path space sampling strategies, such as bi-directional path tracing and metropolis light transport.

Chapter 7

DIRT: Deferred Image-based Ray Tracing

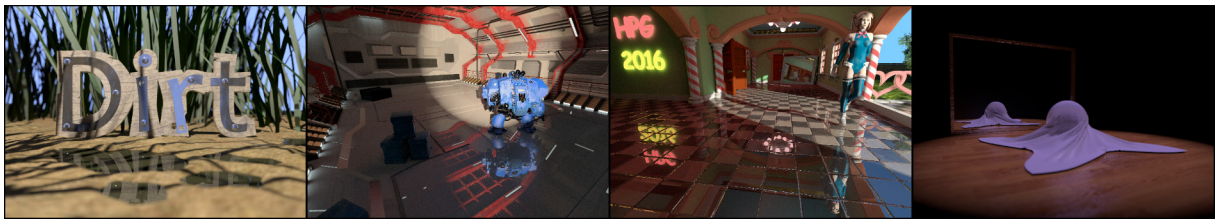


Figure 7.1: Our deferred architecture is able to produce high quality results based on analytic intersection tests through image-space ray tracing while maintaining low construction times and memory requirements, making it applicable to environments containing arbitrary complexity and motion. The last two insets demonstrate non-rigid animation. The proposed method effectively resolves the quality and memory issues that remained unresolved with the method presented in the previous chapter.

In this chapter we continue our investigation on interactive global illumination methods and introduce a novel approach to image-space ray tracing ideally suited for the photorealistic synthesis of fully dynamic environments at interactive frame rates. Our method, designed entirely on the rasterization pipeline, alters the acceleration data structure construction from a per-fragment to a per-primitive basis in order to simultaneously support three important, generally conflicting in prior art, objectives: fast construction times, analytic intersection tests and reduced memory requirements. In every frame, our algorithm operates in two stages: A compact representation of the scene geometry is built based on primitive linked-lists, followed by a traversal step that decouples the ray-primitive intersection tests from the illumination calculations; a process inspired by deferred rendering and the path integral formulation of light transport. Efficient empty space skipping is achieved by exploiting several culling optimizations both in xy- and z-space, such as pixel frustum clipping, depth subdivision and lossless buffer downscaling. An extensive experimental study is finally offered showing that our method advances the area of image-based ray tracing under the constraints posed by arbitrarily complex and animated scenarios.

The chapter is organized as follows: Section 7.1 discusses the problem of fragment-based ray tracing methods and our contribution. Sections 7.2 and 7.3 present the outline and implementation details of our architecture respectively. Section 7.4 reports on the efficiency and robustness of our method through an extensive evaluation study with respect to quality, performance as well as memory consumption, and finally, Section 7.5 offers conclusions and discusses future research directions.

7.1 Overview and Problem Description

Renderers involving ray tracing consist of two stages, an acceleration data structure (ADS) construction stage for speeding up the ray-object intersection calculations and a ray traversal loop, as shown in Figure 7.2. These data structures can be classified either as *spatial* or *rasterization-based*, depending on the approach taken for partitioning and organizing the environment.

Spatial ADS methods are based on a primitive-prioritized ordering to optimize ray intersection queries based on hierarchical representations (kd-trees, BVH, etc.) and grids. Hierarchical partitioning schemes can achieve high quality results along with efficient traversal times, by efficiently exploiting empty space skipping. However, they are mostly suited to partially static environments, where the costly construction stage is performed infrequently. As such, interactivity becomes an issue in cases where geometry can change unexpectedly way in every frame. Uniform grids, on the other hand, are able to maintain fast construction times, but traversal performs poorly even for primary rays in environments with non-uniform distribution of primitives as they cannot exploit empty space skipping. Primary ray performance, however, can be mitigated by employing more elaborate schemes, such as perspective grids [GN12].

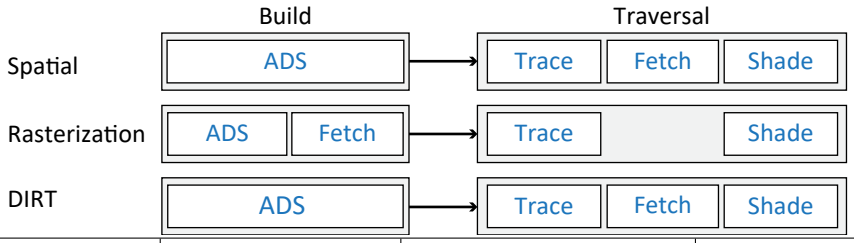
Conversely, rasterization ADS methods, which operate in *image* and *volume* domains, are able to achieve real-time construction times by exploiting the hardware rasterization pipeline. While they are well-suited to dynamic scenes, they are, in general, prone to three major issues. First, for image-space methods, the captured information is sampled in a view-dependent 2D uniform grid, which is potentially a sub-optimal acceleration structure for efficient ray traversal. Second, fragment-based approaches result in poor sampling of oblique geometry, leading to rays passing between fragments and subsequently intersecting with the wrong ones. This is illustrated in Figure 7.3. As a result, the estimated radiance is approximate. Last but not least, they suffer from over-fetching issues; the shading properties of each incoming fragment (or voxel) are fetched and stored during the construction process, regardless if they are needed during the illumination computations. For example, the computational resources required for shading of a typical unidirectional path tracer are upper-bound to the framebuffer resolution and not on the fragment complexity of the entire scene. This can have a significant impact in both the memory requirements and the performance of the algorithm, especially in large and complex environments, where the number of fragments is much higher than the image resolution. Note that this issue, while present, is not significant in screen-space methods storing only a limited number of layers. However, we do not consider such cases, as the captured information is view-dependent and severely limited for the purposes of high-quality interactive rendering.

Our Contribution

In this work, we attempt to bridge the gap between these approaches, by unifying their strengths and lifting their limitations (see table in Figure 7.2). We introduce *Deferred Image-based Ray Tracing* (DIRT), a generic solution for screen-space ray tracing, able to simulate lighting in environments of arbitrary complexity in an accurate, memory- and GPU-friendly manner. Opposite to prior approaches, we apply the deferred tracing scheme of spatial-based methods in a rasterization-based ray tracing framework. This is achieved by disassociating the ADS with the shading data, which are only required for illumination calculations. This modification makes also feasible the conversion of the ADS construction from a per-fragment to a per-primitive basis, hence, improving: (i) *quality*: ray-triangle intersections can analytically be performed in screen-space, (ii) *memory*: lighting attributes are only fetched for primitives that are intersected by a ray and (iii) *performance*: build and traversal times are reduced since the ADS can be created in a compressed representation and the excessive fragment sorting stage during construction is not required.

Note that DIRT is indirectly associated with the fragment-based method presented in Chapter 6. For clarity reasons, we directly cite it as [VVP16] throughout this Chapter.

Our ADS is initially built by performing multilayer rendering in a cubemap configuration based on the work of Vardis et al. [VVP16]. Contrary to fragment-based storage, the entire information is captured on multiple linked lists of pixel-clipped primitives. A coarser representation of this structure is also



Method	Performance		Memory	Quality
	Build	*Traversal*	*Build*	*Traversal*
Spatial	Slow	**Fast**	**Primitives**	**Accurate**
Rasterization	**Fast**	Average	Fragments	Approximate
DIRT	**Fast**	Average	**Tile-based primitives**	**Accurate**

Figure 7.2: Tracing pipelines overview. Our approach effectively combines the advantages of rasterization-based (fast construction times) and spatial-based methods (analytic intersection tests) in a GPU- and memory-friendly context using a deferred rendering pipeline.

explored through the use of conservative rasterization, providing notable improvements in storage cost and rendering times without sacrificing the final quality. During traversal, and for each light bounce, rays are concurrently traced by efficiently skipping empty space regions both in image and depth space [Ulu14; VVP16]. Concerning quality, analytic intersection tests are achieved by adapting screen-space ray tracing [MM14] to our primitive-based data structure. A deferred pass is subsequently performed, only for the pixels that contain intersected primitives, gathering material and intersection properties in an auxiliary shading buffer. Finally, a resolve pass is responsible for computing the final illumination based on the available shading information.

To summarize, the main contributions of this work are:

- A novel deferred approach to image-space ray tracing ideally suited for the efficient rendering of arbitrary animated environments by explicitly using the rasterization pipeline.
- An analytic solution for screen-space ray tracing against a primitive buffer including several optimizations for the early termination of rays, such as primitive-based hierarchical traversal, bucket storage and lossless buffer downscaling.

7.2 Method Description

Akin to any ray tracing pipeline, our image-based algorithm operates in two broad stages: the *Build* stage, where geometry primitives are stored in image-space data structures and the *Traversal* stage, where arbitrary rays are traced and the light transport is resolved in a breadth-first, iterative manner. By using geometric primitives instead of sampled fragment data, analytic tests can be performed, resulting in accurate image generation. Similar to Guntury et al. [GN12], we also exploit per-view perspective grids. However in our case, cells are irregularly sized in the depth dimension, based on the pixel's depth bounds, and hierarchically organized in the image domain, enabling early space skipping that significantly improves traversal in arbitrary environments.

To compute the contribution of each event, when a ray path is traced through a virtual environment, we use the well-known three-point light transport formulation and only allocate storage for the previous, current and next hit points encountered. This way, we overcome the over-fetching issues of prior image-based approaches. Instead of storing fragment-based shading data for the entire environment before the tracing operation, we only retain minimal information about the primitives registered in the image buffers that is required exclusively for analytic ray-primitive intersection tests. After the valid hit point of each ray has been determined, shading information is updated *only once* for each event (see Figure 7.4). For

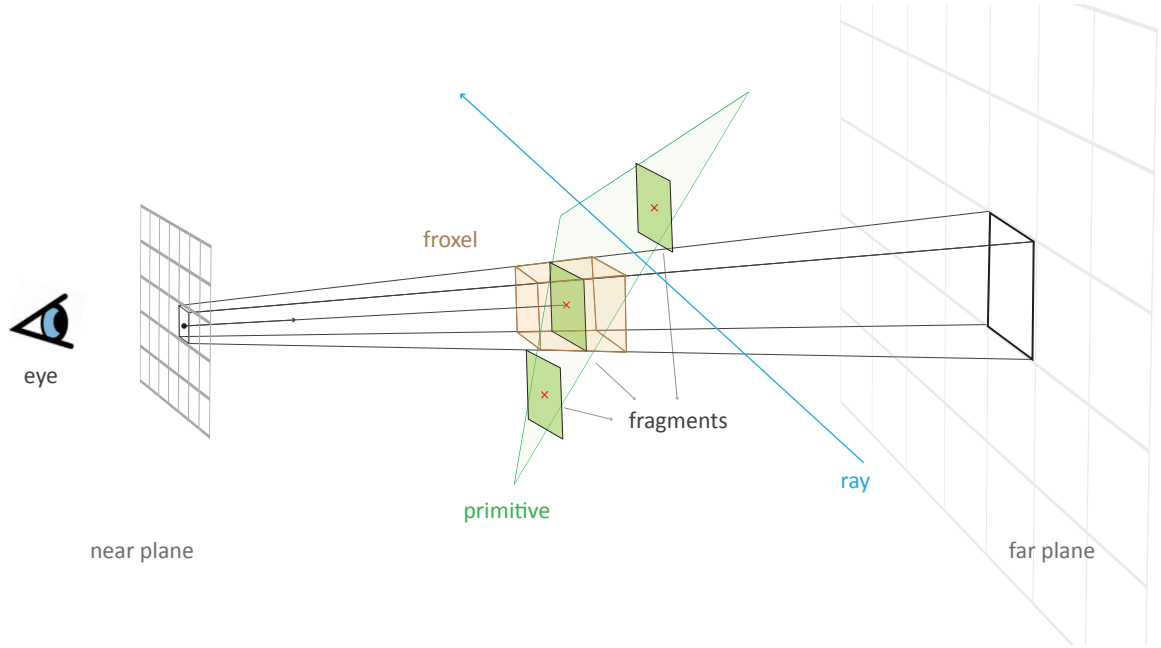


Figure 7.3: Rasterization-based methods fail to capture most ray-object collisions due to their sparse geometry discretization. While replacing fragments with frustum-shaped voxels (froxels) can improve hit-ratio, it cannot guarantee accurate intersection behavior.

multiple events, this update operation involves shifting the currently stored shading attributes of the three path points and appending the newly discovered hit attributes.

Build Stage

The construction of the ADS captures the geometric information of the entire scene by rasterizing the scene in a multiview setup. Two main and one optional steps are required. First, a *Fill & Mipmap Depth* step is employed that stores mipmapped per-pixel minimum and maximum depth values based on the incoming primitives. This operation is required for two reasons: (i) screen-space ray tracing can be performed in a hierarchical manner, significantly reducing the cost of screen-space traversal and (ii) uniform depth subdivision can be exploited, allowing for efficient empty space skipping in the depth dimension. Next, a *Fill Primitives* pass captures the detailed geometric information by storing vertex information and primitive indices. Optionally, a *Direct Visibility* pass can be executed in order to cache surface data for direct lighting calculations, when camera shading effects such as depth of field are not enabled. Note that in order to accommodate dynamic environments, the Build stage occurs in every frame. However, for the multiple view setup, if the geometry is static, only the (trivial) Direct Visibility pass needs to be executed per frame, as the captured image buffers already contain all scene primitives. The entire construction process is explained in detail in Section 7.3.1.

Traversal Stage

This stage is executed in an iterative manner, requiring three passes for each path event: a *Trace*, *Fetch* and *Resolve* pass. The key idea here is that during each event k, per-pixel material information for the three required points (G_{k-1}, G_k, G_{k+1}) is stored in a shading buffer instead of the ADS (see Figure 7.4), significantly reducing the memory requirements. Rays are generated based on the current point G_k, while any identified hits are fetched through a rasterization process and stored at G_{k+1}. At the end of each event, a left shift operation is performed to set the identified hit as the starting location of the next event (see Figure 7.5 - Shade).

Initially, in the *Trace* pass, rays are generated based on either the cached data from the *Direct Visibility* pass or the camera lens. Tracing is performed hierarchically in screen space on the mipmapped per-pixel depth bounds. When a depth-based ray intersection is found, the primitive linked lists associated with the corresponding pixel are traversed until the closest ray-primitive collision is located. At each bounce, all intersections with a particular pixel from multiple rays are stored in an auxiliary buffer (the *hit buffer* in Figure 7.5 - Trace), in the form of a linked-list associated with the hit pixel's location. In the next pass, called the *Fetch* pass, each stored hit is retrieved, rasterized and the resulting attributes are stored in the shading buffer. Finally, the *Shade* pass accumulates the lighting contribution at these points based on the properties stored in the shading buffer, according to the three-point form of the light transport equation, and prepares its contents for the next path event (iteration). Further details regarding the Traversal Stage are discussed in Section 7.3.2.

Data Structures

Our deferred approach requires six structures in total, three for the ADS construction and another three for traversal. The ADS consists of (i) a *depth bounds texture*, storing the per-pixel min-max depth information created during the Fill & Mipmap Depth pass, (ii) a linked-list structure storing primitive indices, called the *id buffer* and (iii) a *vertex buffer* holding per-primitive vertex attribute information. The last two buffers are filled during the Fill Primitives pass. Note that if no new primitives are generated within the rasterization pipeline, e.g., through a tessellation or geometry shader, the vertex buffer already available as part of the rasterization process can be used instead. During tracing, intersection and shading information is associated to two new storage units: a *hit buffer*, containing ray-primitive hit information; and a *shading buffer*, storing per-pixel material properties only for the three hit points (G_{k-1}, G_k, G_{k+1}) associated with the path event (iteration) k.

Acceleration Techniques

Similar to spatial ADS approaches, efficient exclusion of empty space is crucial for fast ray traversal. To this end, we apply several Z-culling optimizations in both the image plane and depth direction by adapting Hierarchical-Z (Hi-Z) traversal [Ulu14] and uniform depth subdivision [VF13] in our primitive-based pipeline. This is accomplished through primitive clipping operations against each pixel's frustum boundaries during the Build stage. A notable improvement in the efficiency of Hi-Z is also obtained by downscaling the ADS through conservative rasterization, where each group of shading pixels is represented as a *tile* in the ADS, reducing the total image-space steps required during ray marching in the Trace pass. Last but not least, a noticeable performance optimization is achieved by a pixel rejection scheme; all pixels with no associated intersections are marked in a *mask texture* and are subsequently discarded during the Fetch pass, where shading takes place. The entire pipeline is illustrated in Figure 7.5 and discussed in further detail in the following section.

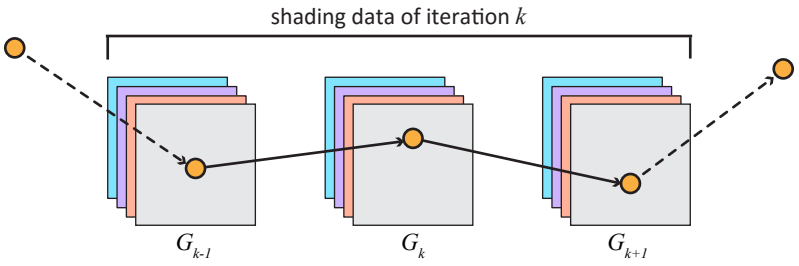


Figure 7.4: To compute the illumination for point G_k, our shading structure contains information only for the three points contributing to the current event iteration k.

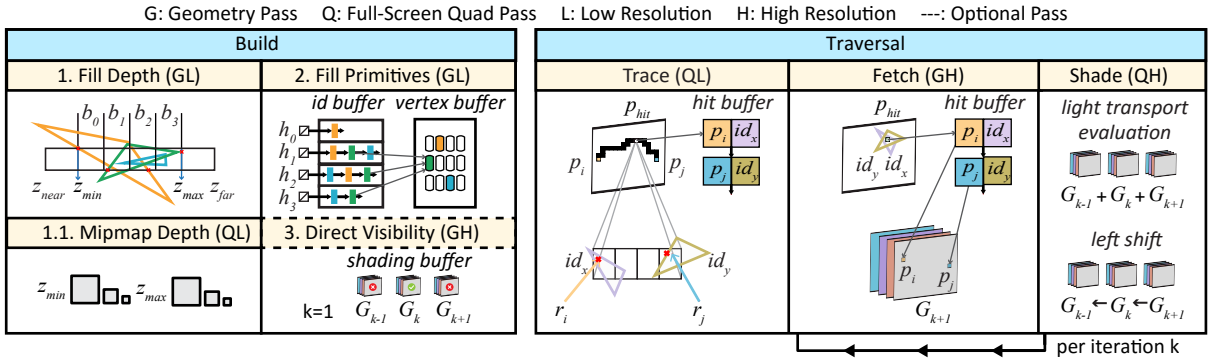


Figure 7.5: A detailed and illustrative diagram of the DIRT architecture stages. (left) Build: Geometry is clipped (i) to compute the mipmapped depth bounds and, in a subsequent pass, (ii) to perform primitive occupancy discretization. (iii) Optionally, a deferred pass is performed to resolve direct visibility. (right) Traversal: Rays (r_i, r_j) are analytically traced in screen-space (starting from p_i, p_j) storing intersection attributes at the hit pixel location (p_{hit}). A geometry pass is subsequently performed at the masked pixels (p_{hit}) to fetch the shading properties of all hit primitives (id_x, id_y). Finally, the illumination contribution at path node (iteration) k is computed and hit point data are shifted in preparation for the next event $(k+1)$. Note that several passes can be performed in lower resolution via conservation rasterization.

7.3 Algorithmic Details

7.3.1 Build Stage

Overview

Briefly, the first stage of DIRT samples the scene primitives in a user-centric manner as follows: First, 6 views $v_j, j = 0 \dots 5$ arranged in a cubemap configuration are created covering the entire scene extents, using the multiview setup proposed by Vardis et al. [VVP16]. Second, the ADS is efficiently constructed by performing two geometry passes for all views: the first one is responsible for computing the depth bounds texture that is subsequently mipmapped to retain the aggregate min/max depth values per texel. The second pass fills the vertex and id buffers. Finally, an optional geometry pass, executed only for the primary camera view, is employed to initialize the shading buffer with the direct lighting results. This pass can be omitted when camera shading effects such as depth-of-field are present, increasing the Traversal stage steps by one.

ADS Downscaling

To reduce the redundant data generated at neighboring pixels corresponding to the same rasterized primitive, a more compact lossless representation of the ADS is constructed via conservation rasterization. This mechanism, exposed as an OpenGL extension for the NVIDIA Maxwell architecture, allows rasterization to generate fragment samples for all pixels intersected by a primitive. Thus, the ADS can be constructed (Fill & Mipmap Depth and Fill Primitives passes) and traced (Trace pass) at a lower resolution $R_l = R_h/S$ by associating each high-resolution pixel p_h, that belongs to the frame buffer's resolution R_h, to a pixel tile p_l with size $S \times S,\ S = 2^m,\ m \in \mathbb{Z}_{\geq 0}$. While this data structure holds the same primitives as the high-resolution ADS due to conservative rasterization, a smaller number of fragments are generated. This is because although a tile S intersects more primitives, only a single copy of a pixel-clipped primitive is generated and stored (Section 7.3.1.2), reducing the memory requirements and construction times. However, due to the reduced portability of conservative rasterization, downscaling can be optionally omitted $(m = 0)$.

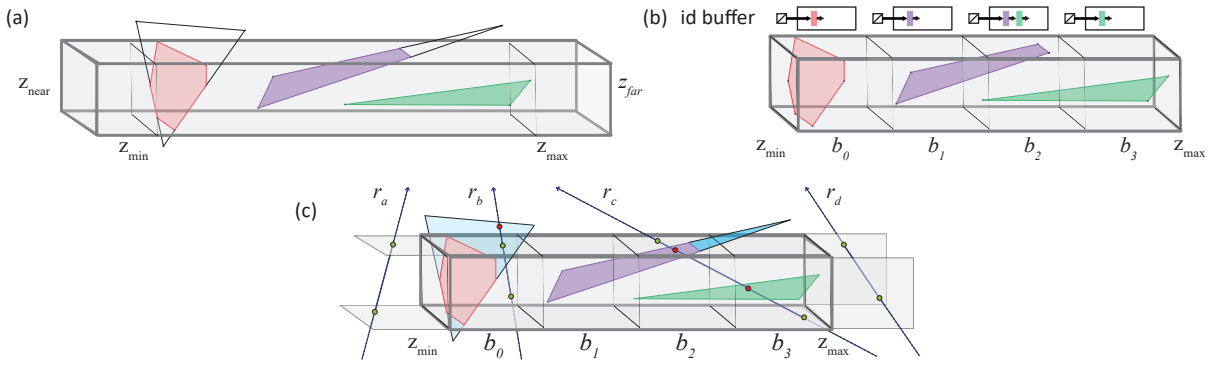


Figure 7.6: Handling primitives in screen-space. Per-pixel primitive clipping is exploited for correct ADS building including (a) depth range computation and (b) multi-bucket placement. (c) Empty-space skipping is performed when either rays pass outside depth boundaries (r_a, r_d) or a valid hit is found in a bucket where traversal order is defined by the ray's direction (r_c). Note that intersections detected outside the pixel frustum are discarded (r_b).

7.3.1.1 Depth Bounds Passes

Fill Depth

The first geometry pass is responsible for generating a depth bounds texture, which stores the depth extents $z_{min}(p_l),\ z_{max}(p_l)$ of all primitives spanning each pixel p_l of every view. Since the ADS stores primitive ids instead of depth-ordered samples, correct calculation of depth extents is achieved by clipping primitives against each pixel's boundaries (see Figure 7.6(a)). This operation is required for two reasons. First, uniform depth subdivision (bucketing) can be exploited, where primitives are assigned to multiple linked-lists spanning equally sized depth intervals, during the Fill Primitives Pass (Section 7.3.1.2). A primitive is assigned to all buckets its depth bounds overlap with (see Figure 7.6(b)). Second, Hi-Z can be performed after constructing a mipmapped version of the depth bounds buffer (Section 7.3.2.1).

From an implementation point of view, a geometry shader outputs each primitive pr_k to the fragment shader, which clips it against the pixel's frustum planes, resulting in a new primitive pr_l. Then, the new primitive's depth extents update the respective pixel's depth bounds via atomic min/max operations. The geometry shader is required for emitting primitives to multiple views and is not required in a single-view implementation.

Mipmap Depth

The depth bounds texture is downscaled and filtered, independently for both the min and max values, into multiple levels d by performing a fast full-screen quad rendering pass. The modest increase in texture memory usage ($\simeq 1.5\times$) provides a significant ray tracing speedup.

7.3.1.2 Fill Primitives Pass

Once the mipmapped depth bounds texture is computed, a second geometry pass is employed to construct the core structure of our ADS: the vertex buffer, a linear array that contains the three per-vertex attribute structures $\mathbf{v}_i(id_k), i = 0, 1, 2$ of each incoming primitive pr_k with unique identifier id_k; as well as the id buffer, an unsorted multiple linked-list structure that contains the unique identifiers for all primitives rasterized in every pixel p_l of every view v_j. Specifically, at the geometry shader invocation, the incoming vertex information (such as positions, normals, texture coordinates, etc.) are stored sequentially in the vertex-buffer. The value of id_k can be based either on the unique primitive identifier provided by the corresponding API for single draw call operations or on a global atomic counter, when multiple draw calls are required. In the following fragment shader invocation, each pixel's p_l depth range $z_{range}(p_l) =$

$z_{max}(p_l) - z_{min}(p_l)$ is split into B uniform sub-intervals $b_0, \ldots, b_{B-1}$. Then, each incoming rasterized primitive is clipped against pixel's p_l frustum boundaries and the resulting (clipped) primitive pr_l is atomically added to all buckets in the range $[b_{min}, b_{max}]$ overlapping its depth extents (see Figure 7.6(b)) :

$$b_{min}(p_l, pr_l) = \lfloor B \cdot (z_{min}(pr_l) - z_{min}(p_l))/z_{range}(p_l) \rfloor, \tag{7.1a}$$
$$b_{max}(p_l, pr_l) = \lfloor B \cdot (z_{max}(pr_l) - z_{min}(p_l))/z_{range}(p_l) \rfloor. \tag{7.1b}$$

7.3.1.3 Direct Visibility Pass

When simple perspective primary rays are generated, a quick rendering pass is performed first, storing the material properties of the tracing starting point for each pixel p_h in the camera view shading buffer G_1. The primary ray hits are created via direct rendering and stored in the shading buffer. In our implementation, this buffer write operation exploits a semaphore-based spin-lock mechanism since the shading buffer contains structures and is therefore not a regular frame buffer, where conventional fragment depth testing operations can be performed. After this step, we also initiate a Shade pass to quickly compute the direct illumination and then continue iteratively the Traversal stage as discussed in the following section.

7.3.2 Traversal Stage

Overview

This stage is executed in an iterative manner, computing the illumination contribution of each scattering event per iteration $k \geq 1$. Briefly, it consists of three different passes:

- Rays are traced in screen space by traversing the downscaled ADS until the closest ray-primitive intersection is analytically found and captured in the hit buffer. The hit buffer stores intersection data for *all rays* intersecting a particular pixel during iteration k. This information includes the primitive ID, intersection position and barycentric coordinates. Since the number of registered hits is unknown a priori, a per-pixel linked-list structure was preferred as a storage container.

- The shading properties for each identified hit are fetched and stored at the shading buffer by performing a geometry rendering pass. Conceptually, the shading buffer's contents correspond to three points for each pixel: G_{k-1}, holding the previous point's position, G_k, pointing to the current event's shading attributes (essentially the starting ray location); and finally, G_{k+1}, storing the new intersected point's fetched material properties. A pixel-rejection mechanism is also exploited to prevent unnecessary invocations of non-intersected pixels.

- A full-screen quad rendering pass is finally employed to compute the shading of the current event by evaluating the three-point light transport formulation at the shading buffer. To support a complete path tracing implementation, an additional *operators* texture is employed during the Trace and Shade passes, storing the updated probability of path segment k as well as the cumulative transport operators respectively.

7.3.2.1 Trace Pass

Screen-space Ray Tracing

This step starts with the generation of a new ray r_h for each pixel p_h, based on the contents of shading buffer G_1. G_1 stores either the camera position or the attributes stored in the direct visibility pass during the Build stage, depending on whether direct rendering is used for the primary rays or not. Rays are clipped against the viewing frustum and subsequently traced via ray marching up to the screen-space projection of their clipped endpoint until the closest intersection is found in the id buffer. If no valid hit is found within the current view, the process is repeated for each subsequent view the ray intersects. Readers

are referred to [MM14; VVP16] for further details on screen-space ray tracing in one or more views respectively. Generally, the image-space ray tracing performance relies heavily on how well the empty space regions of the scene can be avoided, by exploiting the available min-max depth maps. Efficient empty space skipping is achieved by moving in hierarchical steps on the image plane and by avoiding uniform sub-intervals in the depth domain.

Hi-Z Tracing

Single-layer hierarchical screen-space tracing is commonly carried out by switching between tiles of different sizes, belonging to different mip levels $d \in [0, d_h]$ of the high-resolution (R_h) Z-buffer [Ulu14]. We revise accordingly this operation with respect to our compact primitive-based pipeline with two modifications: (i) reaching the lowest mip level does not define an intersection, but initiates a lookup procedure in the id buffer and (ii) the lowest mip level d_l depends on the resolution of the ADS buffer: $d_l \in (0, d_h)$. The last modification essentially, trades traversal overhead in the image-space with larger lists in the depth domain. Specifically, starting at mip level $d = d_h$, the ray's eye-space Z coordinates are compared against the tile's depth extents. For each successful intersection, the mip level is decreased and iteratively refined until reaching the lowest value d_l, where primitive traversal in the id buffer is initiated (see Alg. 2, Ln $1 - 6$). Otherwise, the current tile is skipped and the mip level is increased for the next tracing iteration (see rays r_a, r_d at Figure 7.6(c)).

Bucket Tracing

Once the process enters the primitive traversal phase, analytic ray-primitive intersection tests are performed between the ray and the primitives stored in each bucket of the id buffer. To ensure the closest valid hit, the following operations take place. First, we identify the range of pixel bucket IDs $b_{range}(p_l, r_h) = [b_{min}(p_l, r_h), b_{max}(p_l, r_h)]$ intersected by the ray using Equations 7.1a and 7.1b. Rays that move towards the near plane are checked in back-to-front bucket order, while rays towards the far plane are checked in the opposite order. Note that while this operation is similar to Vardis et al. [VVP16], it is applied here in order to skip any remaining buckets during the traversal operation rather than to ensure correct multi-hit behavior of ray-fragment collisions. Second, we linearly iterate through all primitives of each intersected bucket in order to find the closest ray-primitive collision. For each successful intersection test between a ray r_h and a primitive pr_k, we keep the hit if both of the following conditions are met:

- The hit point lies within the candidate pixel tile $p_l^{d_l}$. We check this by projecting the hit location in the image space of the tile's associated view (see ray r_b at Figure 7.6(c)).
- The intersection distance t_k is the shortest one acquired up to this point ($t_k < t_{hit}$).

For any accepted hit, we maintain the primitive ID id_k, the barycentric coordinates br_k and intersection position pos_k. If at least one valid hit is found in one bucket, the remaining buckets are omitted (see ray r_c at Figure 7.6(c)). The details of this method are shown in Algorithm 2 (Ln $7 - 22$).

Final Storage

After testing all primitives in the intersected bucket for ray-triangle collisions, the closest hit location pos_{hit} is projected to the high-resolution hit buffer location p_{hit}, where a new hit record is atomically inserted in the linked list at that location. Each record contains the hit data information acquired during tracing, the view v_j in which the intersection occurred and the shading buffer pixel location p_h. The latter is the pixel location in which the tracing started ($G_k[p_h]$) and also the location where the fetched data will be stored at the next pass ($G_{k+1}[p_h]$). The view information is required since lighting calculations are performed in eye space. To reduce the overhead of unnecessary fragment invocations of non-intersected pixels, a pixel-rejection scheme is employed, by flagging the pixel p_{hit} as occupied, through the use of a mask texture. In terms of implementation this can be either a depth or a stencil texture, essentially used as an early culling testing mechanism.

Algorithm 2 int *analytic_ssrt* (Pixel p, Ray r, Mip-level d)

▷ (1) Hi-Z tracing
if $z_{min}(r^d) > z_{max}(p^d)$ **or** $z_{max}(r^d) < z_{min}(p^d)$ **then**
    return *no_hit*; ▷ skip complete pixel tile
else if $d > d_l$ **then**
    return $invalid\_level$; ▷ large mip-level, decrease & retry
end if
▷ (2) Bucket tracing
$b_{range} = [b_{min}(p, r), b_{max}(p, r)]$;
for ***each bucket*** $b_i \in b_{range}$ **do**
    for ***each primitive*** $pr_k \in b_i$ **with id** id_k **do**
        $\{t_k, br_k\} \leftarrow hit(r, pr_k)$; ▷ ray-primitive intersection test
        $pos_k \leftarrow orig(r) + t_k \cdot dir(r)$;
        $p_k \leftarrow project(pos_k)$;
        if $t_k \in (0, t_{hit})$ **and** $p_k = p^d$ **then** ▷ new best hit found
            $\{t_{hit}, id_{hit}, br_{hit}, pos_{hit}\} \leftarrow \{t_k, id_k, br_k, pos_k\}$;
        end if
    end for
    if *hit found* **then**
        break; ▷ skip remaining buckets
    end if
end for
return $\{pos_{hit}, br_{hit}, id_{hit}\}$; ▷ return closest hit info

7.3.2.2 Fetch Pass

A full geometry rendering pass is responsible for retrieving the shading information for all intersected primitives identified in the previous pass. For each incoming rasterized primitive, we search the linked-list at that hit buffer location for a match between the stored primitive ID and the incoming one. For every successful comparison during each iteration, we compute the vertex attributes using the stored barycentric coordinates, thus avoiding the attribute extrapolation problem of conservative rasterization. The primitive's shading data are then retrieved through common texture fetching operations. Finally, the primitive's shading data are stored in the shading buffer, at G_{k+1}.

7.3.2.3 Shade Pass

The last step of the iteration initiates a quick full-screen quad rendering pass, which computes the lighting contribution of the current scattering event k. Using the shading buffer contents and the operators texture, the three-point light transport is evaluated and stored in the frame buffer. Afterwards, a left shift operation on the shading-buffer is performed, essentially setting the current intersection point as the starting ray position for the next iteration: $G_{k-1} = G_k, G_k = G_{k+1}$.

7.3.2.4 Notes on Conservative Rasterization

If conservative rasterization is not available, the entire pipeline is slightly modified: (i) the comparison during the Fetch pass requires both primitive and view information and (ii) barycentric coordinates are not required since attribute interpolation is based entirely on the rasterizer.

7.4 Experimental Study

We have implemented DIRT solely on the OpenGL rasterization pipeline, which is compatible with common graphics engines. Due to the design philosophy of the algorithm, a full path tracing solution was implemented and evaluated. Our experiments setup, as well as the provided source code, are described in Sections 7.4.1 and 7.4.2, respectively. We offer a detailed evaluation in terms of quality (Section 7.4.3), performance (Section 7.4.4) and memory (Section 7.4.5) against: (i) the most recent fragment-based deep G-buffer (DGB) ray tracing method by Vardis et al. [VVP16] since it offers the best quality to date but also suffers from the highest memory consumption and over-fetching issues and (ii) two fast and highly optimized spatial-based data structures, HLBVH [PL10; GPM11] and TRBVH [KA13] on a GPGPU path tracer implemented with NVIDIA Optix [Par+10a].

7.4.1 Experiments Setup

We have run our tests on an NVIDIA GTX980Ti GPU and all images were rendered at a resolution of $R_h = 1024 \times 512$, $R_l = R_h$ unless specified otherwise. The node size in bytes for each of our buffers was $size(node_{id}) = 8$ for the ADS and $size(node_{hit}) = size(node_{sb}) = 32$ for the hit and shading buffers respectively. Note that the size of the shading buffer is specific to our path tracing implementation, containing packed material attributes such as diffuse and specular reflectivity, opacity and emissive parameters. All quality results were rendered at ≈ 1000spp, where each sample corresponds to a complete tracing path, including both the direct and indirect illumination events. The Dirt, Candy, Hangar and Cloth scenes are presented in Figure 7.1 while the Bunny scene is shown in Figures 7.7 and 7.9.

7.4.2 Supplemental Material

The shader source code for the entire pipeline of DIRT, i.e., the construction as well as the tracings stages, is provided in the supplemental material of this dissertation.

7.4.3 Quality Evaluation

Fragment-based Ray Tracing.

Tracing methods, which approximate the environment based on discrete samples (fragments), are able to produce plausible results in most typical scenarios. However, they are susceptible to view dependencies due to rays passing through sparsely sampled geometry. While this can be mitigated by assuming each sample is a frustum-shaped voxel of non-zero *thickness*, inaccurate misses cannot be totally avoided. Even worse, the thickness parameter is view-dependent and requires manual adjustment. As a result, these methods are approximate and their error is mostly visible when high frequency phenomena are present. Figure 7.7 demonstrates this on a scene, where perfect reflections are dominant. Rays which would intersect with the seat of the cyan chair end up passing through (a). Increasing the thickness eventually resolves this issue but results in extruding the reflected objects (b, c), which is especially noticeable when compared with the correct result captured by our method (d - see for instance the power cord).

GPGPU Ray Tracing

Figure 7.8 (top) provides a quality comparison of our method against two spatial-based data structures, based on NVIDIA OptiX. We show results with primary and shadow rays (left), primary and secondary rays without shadow rays (middle) and primary and secondary rays with shadow rays (right). We are able to achieve identical results with the reference images, demonstrating that our method can also be employed for quality renderings, a fact that is also validated through all experiments shown in this chapter.

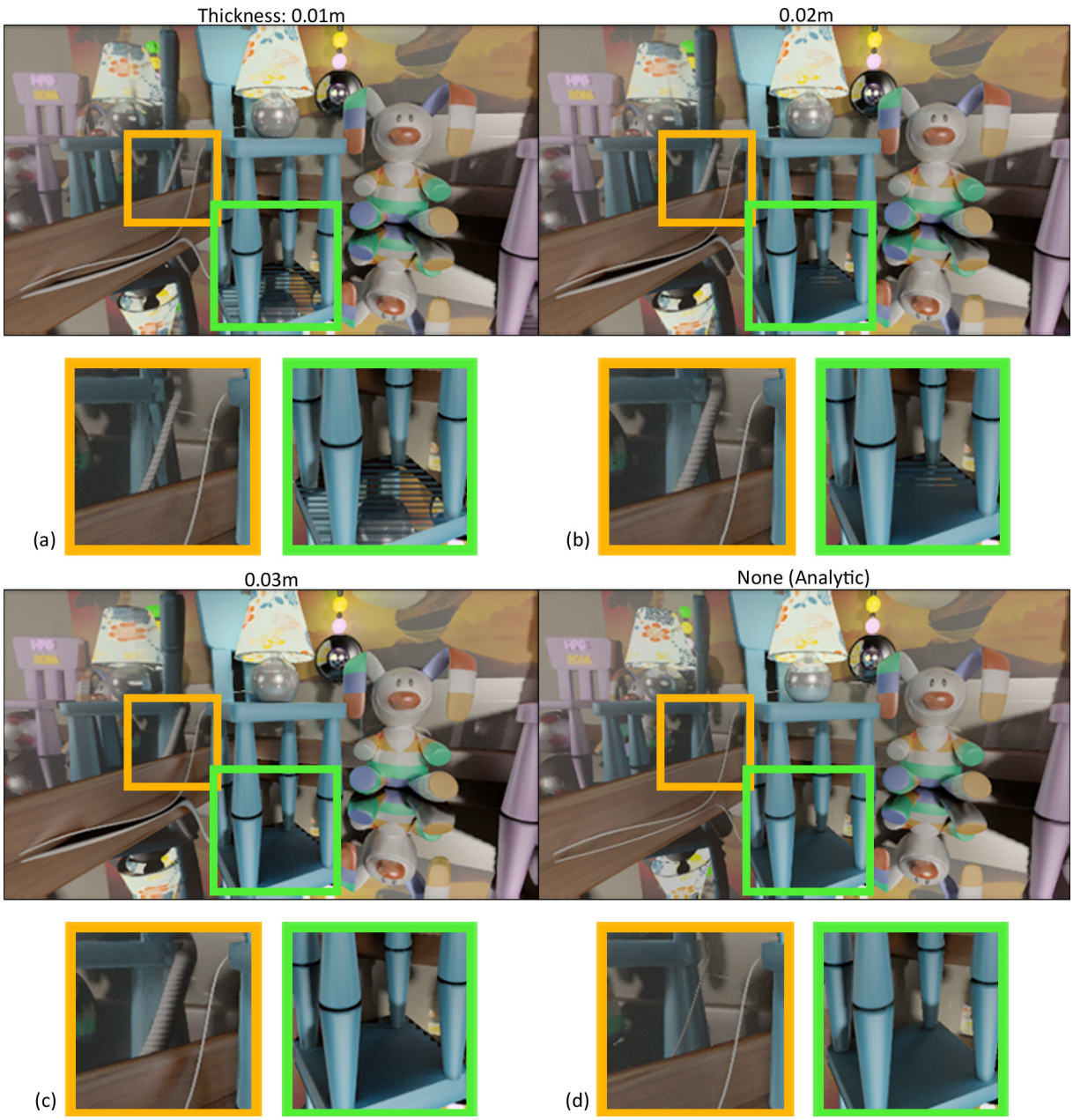


Figure 7.7: Quality comparison (2-indirect bounce lighting) between fragment-based collisions (a,b,c) and our analytic approach (d). High frequency phenomena cannot be accurately captured by manually adjusting the view-dependent thickness parameter. This is mostly visible in the reflections of the lamp power cord (orange inset) and the cyan chair (green inset) at the floor and walls, which appear stripped or extruded.

7.4.4 Performance Evaluation

Figure 7.10 (bottom) presents performance results under different depth subdivision settings and tile sizes as well as the corresponding cost of the Fetch and Shade passes for three test environments. Since these two depend only on the resolution R_h, they are also reported separately (right). Note that we do not provide measurements for different path lengths ($k > 1$) as our tests showed that the increase is linear in closed environments and, as expected, sub-linear in open environments, due to excessive ray misses.

With respect to the number of buckets, we observed an exponential benefit in traversal times while construction is increased linearly by a slope of $\approx 2\%$ due to primitives overlapping more buckets. Regarding changes in the tile size, the performance benefit depends on the number of generated primitives

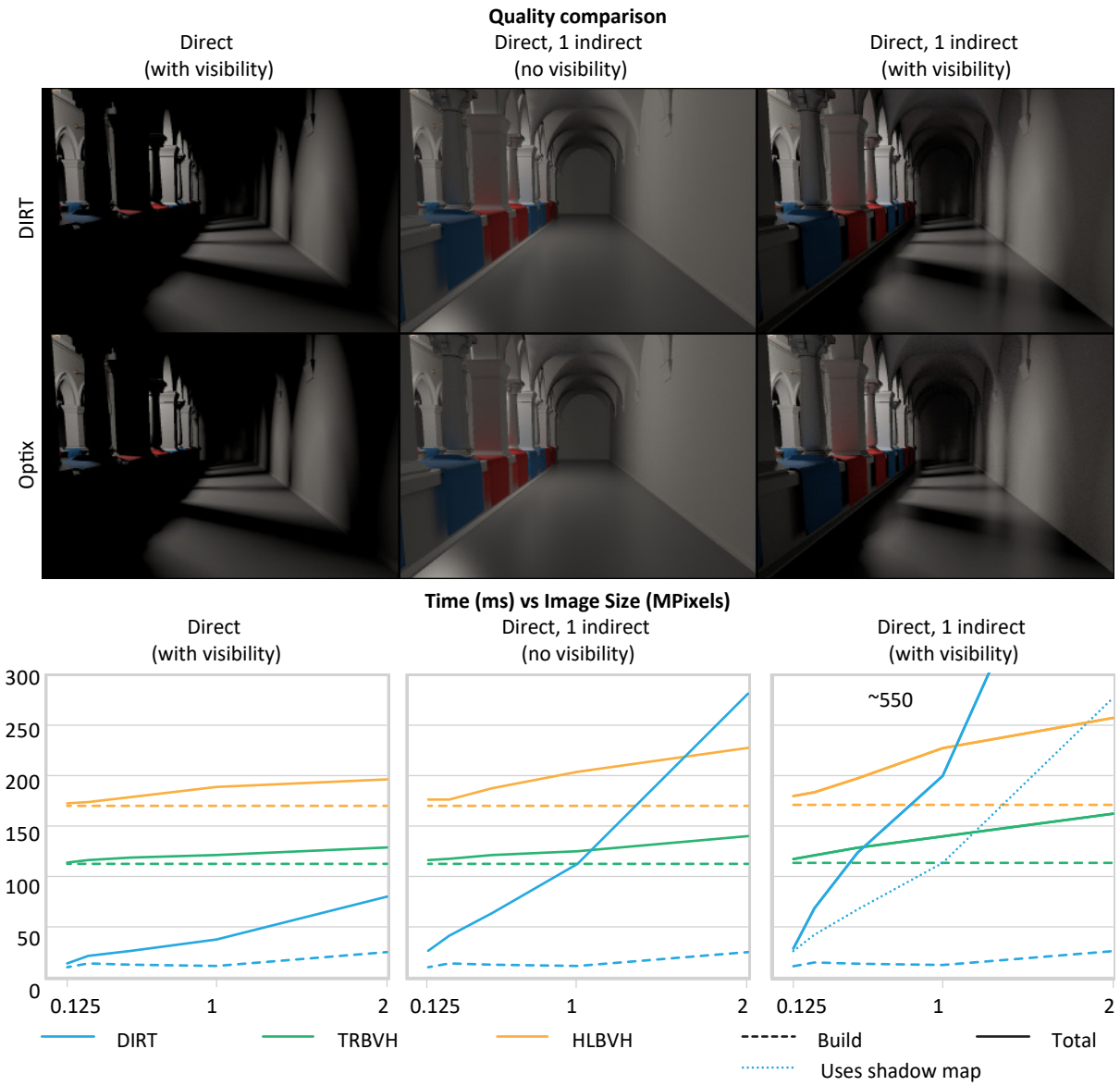


Figure 7.8: Comparison with two spatial-based data structures in the Sponza Atrium, where construction occurs in every frame.

during construction and on the tile *density* of the generated primitives during traversal. Therefore, an increase in the tile size provides notable traversal improvements in scenes where the geometry tessellation is low or medium. Environments containing many finely tessellated objects, however, such as the foliage in the Dirt scene, can benefit from a larger tile size mainly on high resolutions.

Impact of Ray Coherence

Table 7.1 presents the performance on four scenes with different types of rays, such as visibility, perfect reflection, highly glossy and pure diffuse. These measurements were taken in practical scenarios by changing the material properties of the objects in each scene and rendered with our prototype path tracing implementation by spawning rays. Note that we performed our tests on the first path event since the cost of primary rays is minimal due to rasterization. Visibility rays were spawned towards a predetermined location outside the scene's extents and terminated at the first intersection, specular rays towards the reflection direction, glossy rays through BSDF importance sampling and finally, diffuse rays using cosine hemisphere sampling. The performance drop observed when moving from specular to glossy and, finally,

diffuse rays is justified by the increasing incoherence caused by rough surfaces. The latter is also an indication of the improvement we can achieve with further research in the area.

Table 7.1: Mrays/sec of DIRT under different ray types for the first path event.

Scene/Ray	Primitives	Visibility	Specular	Glossy	Diffuse
Cloth	93k	65.5	26.7	26.6	15.2
Bunny	156k	40.3	24.9	14.58	11.6
Hangar	266k	39.9	14.7	9.6	9.6
Candy	420k	24.9	14.9	12.0	9.6

Fragment-based Ray Tracing

Table 7.2 presents a detailed comparison against fragment-based ray tracing in terms of both performance and memory consumption, where the thickness parameter was adjusted to produce the optimal quality result for each scene. For a fair performance comparison against DIRT, conservative rasterization was disabled ($R_l = R_h$) and the number of buckets B was the same as in the fragment-based approach in order to produce a similar number of fragments and clipped primitives. We observed that: (i) the memory was reduced by $7 - 21\%$ while the generated fragments (n_p) was increased by $\approx 5\%$ due to our multi-bucket placement, (ii) the construction times dropped by 15% since no fragment sorting occurs and finally (iii) tracing times dropped by an average of 15% in scenes where the average distances between objects were larger than the thickness parameter. The latter did not occur in the Bunny scene where many rays terminated prematurely, due to the thickness extrusion (see also Figure 7.7). Enabling conservative rasterization, which is required by our method, doubles in most cases the tracing times due to the significantly larger number of fragments generated and, consequently, results in an irregular increase of the primitive lists in cases of oblique geometry.

Table 7.2: Performance and memory comparison of a DGB-based method and DIRT.

	[VVP16] / DIRT			
Scene	Fragments (M)	Memory (MB)	Build (ms)	Traversal (ms)
Bunny	3/3.1	161/150	8/6.5	28/53
Hangar	4.5/4.7	201/184	13.7/11	57/47.9
Candy	8.9/10.2	317/250	18/16.8	75/62

GPGPU Ray Tracing

Figure 7.8 (bottom) provides a performance comparison against two spatial-based data structures, TRBVH and HLBVH, provided by the NVIDIA Optix. We report times on the rendered stills shown at the top, with one primary and one shadow ray (left), a primary and a secondary ray (middle) and one primary, one secondary and two shadow rays (right). Note that the tile size is increased gradually with the resolution from 1x1 to 8x8 pixels. Our results show that our method can maintain much lower construction times (dashed lines) even at high resolutions. In cases of incoherent rays, here the secondary camera path segments and their shadow rays, we observe a steeper increase in traversal times, which diminishes the benefit of low construction as the resolution increases.

Single-view Ray Tracing

Our approach can be also used in a single-view configuration, where image accuracy is traded for speed by skipping out-of-view geometry and respective view traversals. Performance can be further improved by replacing visibility calculations with shadow maps. The latter is demonstrated in Figure 7.9.

Figure 7.9: A single-view setup in conjunction with shadow maps can provide view-dependent, high quality rendering at higher frame rates. Construction/Traversal times (per spp): 5ms/37ms (left) and 9ms/59ms (right) respectively at 1MPixel, $R_l = 512 \times 512$.

7.4.5 Memory Evaluation

Table 7.3 presents the total memory requirements for all structures used in our pipeline, where $c \in [0, B]$ is the average intersected buckets per primitive and n_p is the number of clipped primitive samples. We omit the storage cost of the vertex buffer since it is already required as part of the vertex buffer creation in a common rasterization pipeline. Also note that the mipmap generation of the depth bounds texture slightly increases the memory consumption. By following a deferred tracing strategy, our method reduces significantly the memory required by the large amount of fragment information generated, when the entire scene shading information is captured, instead. This is achieved by (i) bounding the tracing information ($node_{sb}$) and disassociating it from the scene's geometric complexity ($node_{id}$) and (ii) downscaling the ADS.

Table 7.3: Per-pixel memory formulation of the DIRT architecture.

Memory	Requirements (in bytes)	Unit
depth bounds texture	$11 \approx 8 \times 1.33$	$\forall p_l \in v_j$
id buffer	$4 \cdot B + c \cdot n_p \cdot size(node_{id})$	
hit buffer	$4 + size(node_{hit})$	$\forall p_h$
shading buffer	$3 \cdot size(node_{sb})$	
mask texture	2	
$size(node_{id}) = 8,\ size(node_{hit}) = size(node_{sb}) = 32$		

Impact of Downscaling

Figure 7.10 shows the memory gain when the tile size is increased from 1×1 pixels to a tile of 2×2 and 4×4 pixels respectively on three test cases of different fragment complexity. For a bucket size $B > 8$, which is a practical minimum for our approach to achieve relatively decent performance, we observed that the fragment generation is reduced initially by 56% when moving to a 2×2 tile and a further 37% to a tile size of 4×4 pixels, while the memory required was 50% and 32% respectively. Note that the fragment generation drop is justified by the reduction in the number of clipped primitive samples n_p, when larger tile sizes are used. On the other hand, increasing the number of buckets by 8 corresponds leads to a memory increase of 15%, 8% and 4% for each tile setting. The main observations here are: (i) an increase in the tile size provides similar memory benefits on all scenarios regardless of the fragment complexity, (ii) an increase in the number of depth subdivisions consequently increases the memory required, since more buckets are being intersected per primitive and more head pointers are required for the linked lists,

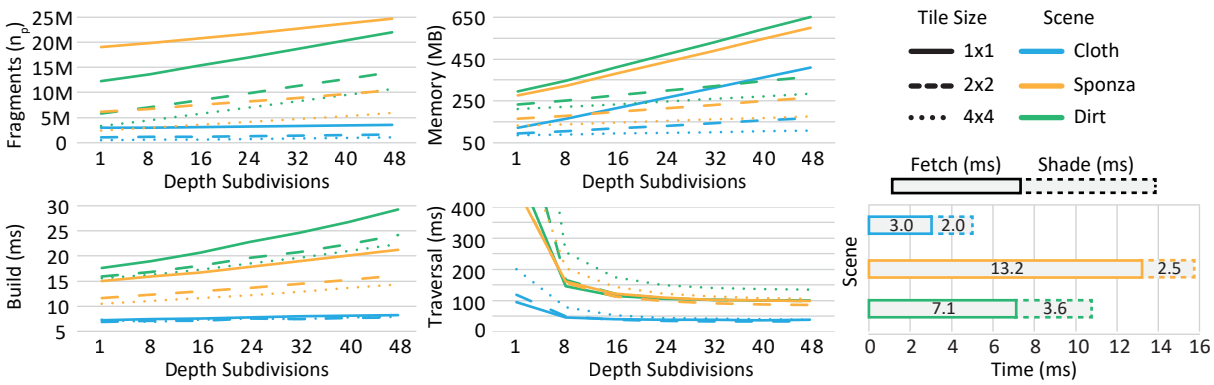


Figure 7.10: Detailed measurements regarding memory (top) and performance (bottom) with variable number of depth subdivisions (B) and tile sizes on scenes of different fragment complexity. Separate timings for the Fetch and Shade passes are also shown on the right.

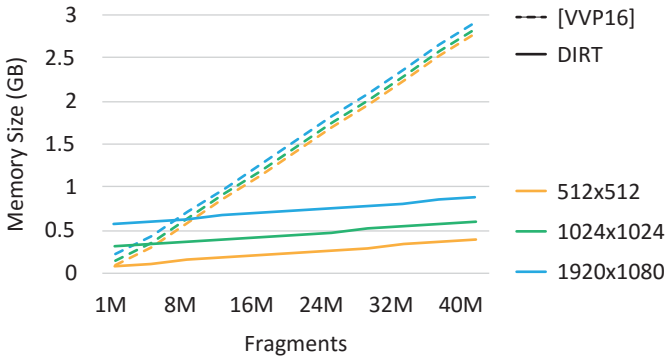


Figure 7.11: Correlation of ADS memory cost of [VVP16] and DIRT methods under increasing screen resolutions ($R_h = R_l$, $B = 1$).

and finally (iii) a combination of tile size 2×2 and depth subdivisions $B \in [20, 40]$ provides a good overall setting in terms of both performance and memory gain.

Impact of Decoupling

While the over-fetching issue is present in all variants based on DGB, memory problems are mainly evident in cases where the entire scene geometry is captured [Hu+14; VVP16]. In the following, we do not compare our method with variants where only a few layers are handled [MM14; Wid+15] since, while their memory overhead is small, the geometric information and resulting quality is very approximate. Figure 7.11 illustrates the memory allocation superiority of our method when compared to a recent fragment-based ADS [VVP16]. We were unable to provide a correct comparison against the method by Hu et al. [Hu+14] since it was not discussed by the authors. However, we assume that the total cost would be similar with the currently compared method due to the three orthographic A-buffers used for capturing the entire scene. Both methods allocate the id and shading data as part of the ADS construction resulting in a huge and wasteful memory demand. While DIRT has an initial overhead due to the fixed memory cost of the decoupled hit and shading buffers, it exhibits a near constant relation with the fragment generation as opposed to a linear one in DGB. This results in reduced memory requirements in most typical cases since the number of generated fragments is usually much higher than the number of pixels. This is even more noticeable when conservative rasterization is employed, since the number generated fragments increases significantly.

7.5 Conclusions and Future Research Directions

In this chapter, we have presented DIRT, a deferred rendering solution to image-based ray tracing. Our framework is based on a novel acceleration structure designed to simultaneously maintain three, commonly conflicting, criteria: (i) fast construction times via (conservative) rasterization in conjunction with (ii) analytic intersection tests using (clipped) primitives as storage nodes and (iii) reduced memory consumption. Our extensive evaluation study illustrates that our method advances and generalizes the field of image-space ray tracing as it addresses several highly important issues: (i) the sampling of oblique geometry and ray-fragment intersections that directly affects the quality of the generated images, and (ii) the over-fetching issues of fragment-based data structures. Furthermore, the wide spectrum of testing scenarios that has been presented showcases the low GPU resources consumed as well as the compelling graphics effects generated when DIRT is demonstrated under a physically-based path tracing platform.

It should be noted that while our deferred scheme is demonstrated on our primitive-based data structure, it can provide similar benefits on the fragment-based data structures discussed in Chapter 6. An additional observation regarding DIRT is that, while the multiview approach is clearly related to interactive rendering in today's hardware, a single-view configuration can maintain near real-time performance in various scenarios, making the method applicable to the realm of real-time rendering as well.

While our method is able to achieve high quality results, new issues have arised opening new problems and identifying promising research directions. The most notable improvement can be achieved in the traversal times, mainly in incoherent rays, when compared to spatial-based data structures. This is an expected outcome of our shallow acceleration structure (one-level deep regular subdivision), the overhead caused by the increase of our linked lists due to conservative rasterization and the fact that we do not explore any coherence strategies. Also, an overhead is caused in both construction times and memory consumption due to the duplicate primitive information currently present in multiple buckets of the same tile, which is currently a requirement for accurate traversal when depth subdivision is involved. It would be highly interesting to investigate these issues in the future by exploiting adaptive depth subdivision techniques, clustering algorithms and strategies for ray coherence [Mat+15].

Chapter 8

Conclusions and Future Work

In this thesis, we have investigated several illuminations problems, directly applicable to real-time and interactive rendering of arbitrarily complex and dynamic scenarios. Concluding, we offer an overview of our work and provide a general discussion regarding future research directions.

We started our investigation in Chapter 4, by focusing on the view-dependency problems of screen-space ambient occlusion techniques, in the domain of real-time rendering. Contrary to prior work, which was mainly focused on the improvement of the sampling quality and efficiency of the AO estimator, we were concerned with the general improvement of the algorithmic input of screen-space approaches. Any previous work on this area was employing either multiple depth-peeled layers or fixed view configurations, resulting in computational overhead and erroneous AO estimation as each layer/view contributed with equal importance. To address these issues, we proposed a generic single-pass weighting scheme that augments the screen-space AO estimation based on the contribution from other, arbitrary viewpoints, such as the readily available shadow maps. Instead of equally considering each view, our weighting functions balance the contribution of each viewpoint on a per-fragment basis, without overestimating occlusion or requiring a specializing view setup. To reduce the overhead of incorporating multiple views, we perform an adaptive sample reduction based on the importance of each view, while maintaining high quality. Our evaluation study demonstrates that our method is generic and thus, can be applied on various AO algorithms, improving the total convergence of the AO estimator in screen-space techniques. While our research was specifically directed to screen-space ambient occlusion, it posed some important questions regarding the efficiency of single-layered/view approaches, in general. Furthermore, it opened promising research directions related to the automated and optimized positioning of arbitrary viewpoints and the applicability of such approaches to simulate more advanced global illumination phenomena.

In our second approach, in Chapter 5, we focused on the improvement of illumination algorithms, based on a conjunction of volume- and image-space representations, still in the realm of real-time rendering. We identified several problems with respect to prior work, such as increased computational overhead in dynamic radiance caching schemes and view-dependency issues during indirect shadowing calculations. To this end, we proposed a very fast and stable diffuse global illumination method where high performance improvements are accomplished based on two generic strategies: (i) directional chrominance subsampling and, (ii) an optimized cache population scheme. Additionally, our method is also able to support view-independent indirect shadowing, at minimal cost. Our experimental study concluded that the presented strategies open some interesting future research directions, both with respect to directional compression methods as well as on positioning schemes able to incorporate more complex phenomena, such as participating media.

In the domain of interactive rendering, we began our investigation in Chapter 6, by exploring the ability of the rasterization pipeline to incorporate unified global illumination effects, under the consideration of fully dynamic environments. Our research resulted in several important contributions to the field of interactive rendering: First, we proposed a generic multiview/fragment configuration that was able, for the first time, to effectively capture in high detail the entire scene, which was not possible before using traditional volume- and image-space methods. Second, we presented a multi-fragment variant

specifically designed for image-space ray tracing, allowing fast construction times as well as efficient empty space skipping in both the image- and depth-domain. Third, we generalized image-space ray tracing for ray-fragment intersection queries based on a multi-layer/view pipeline. The aforementioned contributions allowed for the implementation of a full path-tracing solution in the rasterization pipeline, directly applicable to rendering of fully dynamic environments. Finally, we conducted a thorough performance/memory analysis of multi-fragment variants and their ability to perform ray-fragment collision queries against arbitrary pixel fragments in a rasterization-based ray tracing framework. Our evaluation study answered some important questions regarding the quality and memory issues present in screen-space illumination techniques and served as an inspiration for our next research method.

Our last work, in Chapter 7, concluded our research by presenting DIRT, a rasterization-based technique for global illumination of fully dynamic environments. The proposed method addresses the previously identified problems of fragment-based ray tracing by applying the deferred nature of the traditional ray-tracing pipeline in a rasterization-based framework. This modification allowed us to alter the acceleration data structure from a per-fragment to a per-primitive basis and support three, conflicting in prior work, objectives: (i) fast construction times, (ii) analytic intersection tests and (iii) reduced memory requirements. In order to maintain efficient traversal, we exploit several empty-space skipping optimizations both in image- and depth-space, such as screen-space hierarchical traversal, pixel frustum clipping, depth subdivision and lossless buffer downscaling. Our extensive evaluation showcases that our method further advances the field of image-space ray tracing under the constraints posed by arbitrarily complex, dynamic scenes. Furthermore, it strengthened the need for further investigation with respect to both the construction stage as well as the optimization of the traversal times, by exploring areas related to adaptive depth subdivision, clustering techniques and coherence strategies.

In general, we believe that the research conducted throughout this thesis has addressed several important issues in the fields of real-time and interactive rendering by focusing on the efficiency, quality and robustness of screen-space and volume-based techniques as well as on the overall capability of rasterization-based methods to accommodate full global illumination algorithms. As explained in the previous chapters, however, the efficient computation of dynamic global illumination is an inherently hard task. In real-time rendering, unfortunately, approximations are unavoidable due to the hard time constraints involved in the process. In this domain, the impressive, but not necessarily physically-correct results accomplished by recent rasterization-based methods can be unquestionably improved even further by employing more robust and efficient representations. In interactive rendering, the increased need to support complex dynamic environments in conjunction with the shared requirements regarding quality and interactivity constitute the investigation towards more efficient ray tracing algorithms of adamant importance. There, significant progress has been achieved from the research community by improving areas that have been, for a long time, applicable only to static environments. However, we have also showed that the same results can be obtained by elegantly exploiting the rasterization pipeline, proving that a strong continuum between these two, traditionally conflicting, approaches is actually possible.

From a generic point of view, it seems that any improvements in ray tracing and rasterization will eventually result in converging to the same outcome from different sides of the spectrum; efficient ray tracing algorithms will increase in interactivity, while advanced rasterization techniques will improve on quality. It is our opinion that, in the end, each one will influence the other in such aspects, where high quality interactive content will be a result of a hybrid form of rasterization and ray tracing, both algorithmically as well as in hardware.

To conclude, despite the tremendous progress on the landscape of both fields, we hope that the methodologies presented here have answered important questions and given rise to other, more interesting, problems in some key areas with increased research interest, where high potential for software and hardware improvements is feasible in the near future.

Content Credits

In this section, we provide information regarding the content that has been used throughout this thesis:

- The Sponza Atrium model (Crytek version), shown in Chapters 4 and 5, was downloaded from Crytek's website [Cry].
- The high resolution version of the decimated Lucy model, used in Chapter 5, was obtained from the Stanford University Computer Graphics Laboratory [Sta].
- The Cloth-ball simulation, in Chapter 6, was obtained from the model database of the GAMMA Research Group at the University of North Carolina [Unib].
- The Fairy Forest animation, used in Chapter 6, was downloaded from Ingo Wald's 3D Animation Repository at the University of Utah [Ing].
- The Smithy model, in Chapter 6, was downloaded from Unity Technologies and is part of the The Blacksmith Environments package [Unia].
- The Sponza Atrium model, shown in Chapters 6 and 7, was downloaded from Morgan McGuire's Computer Graphics Archive [Mor].
- The Marie Rose animation model shown as part of the Candy scene, in Chapter 7, was obtained from the Sketchfab's model database [Ske].
- Any remaining models, were created by the Computer Graphics Group at the Athens University of Economics and Business [Pap].

Bibliography

[AB91] Edward H. Adelson and James R. Bergen. *The Plenoptic Function and the Elements of Early Vision*. In: *Computational Models of Visual Processing*. MIT Press, 1991, pp. 3–20 (Cited on page 28).

[AC86] James Arvo and MA Chelmsford. *Backward ray tracing*. In: *Developments in Ray Tracing, Computer Graphics, Proc. of ACM SIGGRAPH 86 Course Notes*. 1986, pp. 259–263 (Cited on page 38).

[AMH08] Tomas Akenine-Möller, Tomas Moller, and Eric Haines. *Real-Time Rendering*. 3rd edition. Natick, MA, USA: A. K. Peters, Ltd., 2008. ISBN: 978-1-56881-424-7 (Cited on pages 3, 44, 50).

[And+16] Tobias Grønbeck Andersen, Viggo Falster, Jeppe Revall Frisvad, and Niels Jørgen Christensen. *Hybrid fur rendering: combining volumetric fur with explicit hair strands*. In: *The Visual Computer* 32.6 (2016), pp. 739–749. ISSN: 1432-2315. DOI: 10.1007/s00371-016-1252-x (Cited on page 45).

[And15] Johan Andresson. *The Rendering Pipeline - Challenges & Next Steps - Open Problems in Real-time Rendering*. In: *ACM SIGGRAPH 2015 Courses*. SIGGRAPH '15. Los Angeles, California, 2015 (Cited on page 4).

[App68] Arthur Appel. *Some Techniques for Shading Machine Renderings of Solids*. In: *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*. AFIPS '68 (Spring). Atlantic City, New Jersey: ACM, 1968, pp. 37–45. DOI: 10.1145/1468075.1468082 (Cited on pages 7, 38).

[AS00] Michael Ashikhmin and Peter Shirley. *An Anisotropic Phong BRDF Model*. In: *J. Graph. Tools* 5.2 (Feb. 2000), pp. 25–32. ISSN: 1086-7651. DOI: 10.1080/10867651.2000.10487522 (Cited on page 28).

[AWL13] Miika Aittala, Tim Weyrich, and Jaakko Lehtinen. *Practical SVBRDF Capture in the Frequency Domain*. In: *ACM Trans. Graph.* 32.4 (July 2013), 110:1–110:12. ISSN: 0730-0301. DOI: 10.1145/2461912.2461978 (Cited on page 28).

[Bav+07] Louis Bavoil, Steven P. Callahan, Aaron Lefohn, João L. D. Comba, and Cláudio T. Silva. *Multi-fragment Effects on the GPU Using the K-buffer*. In: *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*. I3D '07. Seattle, Washington: ACM, 2007, pp. 97–104. ISBN: 978-1-59593-628-8. DOI: 10.1145/1230100.1230117 (Cited on page 54).

[Ben75] Jon Louis Bentley. *Multidimensional Binary Search Trees Used for Associative Searching*. In: *Commun. ACM* 18.9 (Sept. 1975), pp. 509–517. ISSN: 0001-0782. DOI: 10.1145/361002.361007 (Cited on page 55).

[BKB13] Fabian Bauer, Martin Knuth, and Jan Bender. *Screen-space ambient occlusion using a-buffer techniques*. In: *Computer-Aided Design and Computer Graphics (CAD/Graphics), 2013 International Conference on*. IEEE. 2013, pp. 140–147 (Cited on page 58).

[Bli05] James F. Blinn. *What Is a Pixel?* In: *IEEE Comput. Graph. Appl.* 25.5 (Sept. 2005), pp. 82–87. ISSN: 0272-1716. DOI: 10.1109/MCG.2005.119 (Cited on page 25).

[Bli77] James F. Blinn. *Models of Light Reflection for Computer Synthesized Pictures*. In: *Proceedings of the 4th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '77. San Jose, California: ACM, 1977, pp. 192–198. DOI: 10.1145/563858.563893 (Cited on page 28).

[BM08a] Louis Bavoil and Kevin Myers. *Deferred rendering using a stencil routed k-buffer*. In: *ShaderX6: Advanced Rendering Techniques* (2008), pp. 189–198 (Cited on page 54).

[BM08b] Louis Bavoil and Kevin Myers. *Order Independent Transparency with Dual Depth Peeling*. Tech. rep. Nvidia Corp., 2008 (Cited on page 54).

[BM15] J. Bittner and D. Meister. *T-SAH: Animation Optimized Bounding Volume Hierarchies*. In: *Computer Graphics Forum* 34.2 (2015), pp. 527–536. ISSN: 1467-8659. DOI: 10.1111/cgf.12581 (Cited on page 55).

[Bou+13] Guillaume Bouchard, Jean-Claude Iehl, Victor Ostromoukhov, and Pierre Poulin. *Improving Robustness of Monte-Carlo Global Illumination with Directional Regularization*. In: *SIGGRAPH Asia 2013 Technical Briefs*. SA '13. Hong Kong, Hong Kong: ACM, 2013, 22:1–22:4. ISBN: 978-1-4503-2629-2. DOI: 10.1145/2542355.2542383 (Cited on page 38).

[BS09] Louis Bavoil and Miguel Sainz. *Multi-layer dual-resolution screen-space ambient occlusion*. In: *SIGGRAPH 2009: Talks*. SIGGRAPH '09. New Orleans, Louisiana: ACM, 2009, 45:1–45:1. ISBN: 978-1-60558-834-6. DOI: 10.1145/1597990.1598035 (Cited on page 58).

[BSD08] Louis Bavoil, Miguel Sainz, and Rouslan Dimitrov. *Image-space horizon-based ambient occlusion*. In: *ACM SIGGRAPH 2008 talks*. SIGGRAPH '08. Los Angeles, California: ACM, 2008, 22:1–22:1. ISBN: 978-1-60558-343-3. DOI: 10.1145/1401032.1401061 (Cited on page 57).

[Bun05] Michael Bunnell. *Dynamic Ambient Occlusion and Indirect Lighting*. In: *GPU Gems 2*. Ed. by Matt Pharr. Addison-Wesley, 2005, pp. 223–233 (Cited on pages 41, 56).

[Cam03] Wallace H Campbell. *Introduction to geomagnetic fields*. Cambridge University Press, 2003 (Cited on page 18).

[Car84] Loren Carpenter. *The A -buffer, an Antialiased Hidden Surface Method*. In: *SIGGRAPH Comput. Graph.* 18.3 (Jan. 1984), pp. 103–108. ISSN: 0097-8930. DOI: 10.1145/964965.808585 (Cited on pages 48, 53).

[Cat74] Edwin Earl Catmull. "A subdivision algorithm for computer display of curved surfaces." PhD thesis. 1974 (Cited on pages 3, 47, 53).

[CG12] Cyril Crassin and Simon Green. *Octree-Based Sparse Voxelization Using The GPU Hardware Rasterizer*. In: *OpenGL Insights*. Ed. by Patrick Cozzi and Christophe Riccio. CRC Press, July 2012, pp. 303–319. ISBN: 978-1439893760 (Cited on page 62).

[CG85] Michael F. Cohen and Donald P. Greenberg. *The Hemi-cube: A Radiosity Solution for Complex Environments*. In: *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '85. New York, NY, USA: ACM, 1985, pp. 31–40. ISBN: 0-89791-166-0. DOI: 10.1145/325334.325171 (Cited on pages 36, 37).

[CH65] Richard Courant and David Hilbert. *Methods of mathematical physics*. Vol. 1. CUP Archive, 1965 (Cited on page 18).

[Chr08] P Christensen. *Point-based approximate color bleeding*. In: *Pixar Technical Notes* 2.5 (2008), p. 6 (Cited on page 40).

[Chr10] Per H Christensen. *Point-based global illumination for movie production*. In: *ACM SIGGRAPH*. 2010 (Cited on page 40).

[CK10] Andrew Cox and Jan Kautz. *Dynamic Ambient Occlusion from Volumetric Proxies*. In: *ACM SIGGRAPH 2010 Posters*. SIGGRAPH '10. Los Angeles, California: ACM, 2010, 124:1–124:2. ISBN: 978-1-4503-0393-4. DOI: 10.1145/1836845.1836978 (Cited on page 57).

[CMS87] Brian Cabral, Nelson Max, and Rebecca Springmeyer. *Bidirectional Reflection Functions from Surface Bump Maps*. In: *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '87. New York, NY, USA: ACM, 1987, pp. 273–281. ISBN: 0-89791-227-6. DOI: 10.1145/37401.37434 (Cited on page 18).

[Coo84] Robert L. Cook. *Shade Trees*. In: *SIGGRAPH Comput. Graph.* 18.3 (Jan. 1984), pp. 223–231. ISSN: 0097-8930. DOI: 10.1145/964965.808602 (Cited on page 45).

[Cor] Cornell University. *Light Measurement Laboratory*. URL: http://www.graphics.cornell.edu/research/measure (Cited on page 28).

[Cor+09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. 3rd edition. The MIT Press, 2009. ISBN: 9780262033848 (Cited on page 54).

[CPC84] Robert L. Cook, Thomas Porter, and Loren Carpenter. *Distributed Ray Tracing*. In: *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '84. New York, NY, USA: ACM, 1984, pp. 137–145. ISBN: 0-89791-138-5. DOI: 10.1145/800031.808590 (Cited on page 38).

[Cra+11] Cyril Crassin, Fabrice Neyret, Miguel Sainz, Simon Green, and Elmar Eisemann. *Interactive Indirect Illumination Using Voxel Cone Tracing*. In: *Computer Graphics Forum (Proceedings of Pacific Graphics 2011)* 30.7 (Sept. 2011) (Cited on page 62).

[Cra10] Cyril Crassin. *Fast and accurate single-pass A-Buffer*. 2010 (Cited on page 53).

[Cry] Crytek. *CryEngine 3 Downloads*. URL: http://www.crytek.com/cryengine/cryengine3/downloads (Cited on page 131).

[CT82] R. L. Cook and K. E. Torrance. *A Reflectance Model for Computer Graphics*. In: *ACM Trans. Graph.* 1.1 (Jan. 1982), pp. 7–24. ISSN: 0730-0301. DOI: 10.1145/357290.357293 (Cited on pages 27, 28).

[CTE05] David Cline, Justin Talbot, and Parris Egbert. *Energy Redistribution Path Tracing*. In: *ACM SIGGRAPH 2005 Papers*. SIGGRAPH '05. Los Angeles, California: ACM, 2005, pp. 1186–1195. DOI: 10.1145/1186822.1073330 (Cited on page 38).

[Cuy+12] Tom Cuypers, Tom Haber, Philippe Bekaert, Se Baek Oh, and Ramesh Raskar. *Reflectance Model for Diffraction*. In: *ACM Trans. Graph.* 31.5 (Sept. 2012), 122:1–122:11. ISSN: 0730-0301. DOI: 10.1145/2231816.2231820 (Cited on page 22).

[CWH93] Michael F. Cohen, John Wallace, and Pat Hanrahan. *Radiosity and Realistic Image Synthesis*. San Diego, CA, USA: Academic Press Professional, Inc., 1993. ISBN: 0-12-178270-0 (Cited on page 36).

[Dan+99] Kristin J. Dana, Bram van Ginneken, Shree K. Nayar, and Jan J. Koenderink. *Reflectance and Texture of Real-world Surfaces*. In: *ACM Trans. Graph.* 18.1 (Jan. 1999), pp. 1–34. ISSN: 0730-0301. DOI: 10.1145/300776.300778 (Cited on page 28).

[Dav+14] Tomáš Davidovič, Jaroslav Křivánek, Miloš Hašan, and Philipp Slusallek. *Progressive Light Transport Simulation on the GPU: Survey and Improvements*. In: *ACM Trans. Graph.* 33.3 (June 2014), 29:1–29:19. ISSN: 0730-0301. DOI: 10.1145/2602144 (Cited on page 38).

[DBS08] Rouslan Dimitrov, Louis Bavoil, and Miguel Sainz. *Horizon-split ambient occlusion*. In: *Proceedings of the 2008 symposium on Interactive 3D graphics and games*. I3D '08. Redwood City, California: ACM, 2008, 5:1–5:1. ISBN: 978-1-59593-983-8. DOI: 10.1145/1342250.1357017 (Cited on page 68).

[DLW93] Philip Dutré, Eric P Lafortune, and Yves Willems. *Monte Carlo light tracing with direct computation of pixel intensities*. In: *3rd International Conference on Computational Graphics and Visualisation Techniques*. 1993, pp. 128–137 (Cited on page 38).

[Don+09a] Zhao Dong, Thorsten Grosch, Tobias Ritschel, Jan Kautz, and Hans-Peter Seidel. *Real-time Indirect Illumination with Clustered Visibility*. In: *Vision, Modeling, and Visualization Workshop*. 2009 (Cited on page 59).

[Don+09b] Craig Donner, Jason Lawrence, Ravi Ramamoorthi, Toshiya Hachisuka, Henrik Wann Jensen, and Shree Nayar. *An Empirical BSSRDF Model*. In: *ACM SIGGRAPH 2009 Papers*. SIGGRAPH '09. New Orleans, Louisiana: ACM, 2009, 30:1–30:10. ISBN: 978-1-60558-726-4. DOI: 10.1145/1576246.1531336 (Cited on page 28).

[Don+10] Yue Dong, Jiaping Wang, Xin Tong, John Snyder, Yanxiang Lan, Moshe Ben-Ezra, and Baining Guo. *Manifold Bootstrapping for SVBRDF Capture*. In: *ACM SIGGRAPH 2010 Papers*. SIGGRAPH '10. Los Angeles, California: ACM, 2010, 98:1–98:10. ISBN: 978-1-4503-0210-4. DOI: 10.1145/1833349.1778835 (Cited on page 28).

[DS05] Carsten Dachsbacher and Marc Stamminger. *Reflective shadow maps*. In: *I3D '05: Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*. Washington, District of Columbia: ACM, 2005, pp. 203–231. ISBN: 1-59593-013-2 (Cited on pages 49, 53, 58, 59).

[DS06] Carsten Dachsbacher and Marc Stamminger. *Splatting indirect illumination*. In: *I3D '06: Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games*. Redwood City, California: ACM, 2006, pp. 93–100. ISBN: 1-59593-295-X (Cited on page 59).

[Dut+06] Philip Dutre, Kavita Bala, Philippe Bekaert, and Peter Shirley. *Advanced Global Illumination*. AK Peters Ltd, 2006. ISBN: 1568813074 (Cited on pages 14, 15, 36, 37).

[ED06] Elmar Eisemann and Xavier Décoret. *Fast Scene Voxelization and Applications*. In: *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games*. I3D '06. Redwood City, California: ACM, 2006, pp. 71–78. ISBN: 1-59593-295-X. DOI: 10.1145/1111411.1111424 (Cited on page 88).

[Edm96] Alan Robert Edmonds. *Angular momentum in quantum mechanics*. Princeton University Press, 1996 (Cited on page 18).

[Edw+06] Dave Edwards, Solomon Boulos, Jared Johnson, Peter Shirley, Michael Ashikhmin, Michael Stark, and Chris Wyman. *The Halfway Vector Disk for BRDF Modeling*. In: *ACM Trans. Graph.* 25.1 (Jan. 2006), pp. 1–18. ISSN: 0730-0301. DOI: 10.1145/1122501.1122502 (Cited on page 27).

[Eve01] Cass Everitt. *Interactive Order-Independent Transparency*. Tech. rep. NVIDIA Corporation, 2001 (Cited on page 54).

[Fey88] Richard P. Feynman. *QED: The Strange Theory of Light and Matter*. Princeton University Press, Oct. 1988. ISBN: 0691024170 (Cited on page 22).

[FM08] Dominic Filion and Rob McNaughton. *Effects & techniques*. In: *ACM SIGGRAPH 2008 Games*. SIGGRAPH '08. Los Angeles, California: ACM, 2008, pp. 133–164. DOI: 10.1145/1404435.1404441 (Cited on page 58).

[Gar09] Kirill Garanzha. *The Use of Precomputed Triangle Clusters for Accelerated Ray Tracing in Dynamic Scenes*. In: *Proceedings of the Twentieth Eurographics Conference on Rendering*. EGSR'09. Girona, Spain: Eurographics Association, 2009, pp. 1199–1206. DOI: 10.1111/j.1467-8659.2009.01497.x (Cited on page 55).

[Gau+04] Pascal Gautron, Jaroslav Krivanek, Sumanta Pattanaik, and Kadi Bouatouch. *A Novel Hemispherical Basis for Accurate and Efficient Rendering*. In: *Proceedings of the Fifteenth Eurographics Conference on Rendering Techniques*. EGSR'04. Norrkö;ping, Sweden: Eurographics Association, 2004, pp. 321–330. ISBN: 3-905673-12-6. DOI: 10.2312/EGWR/EGSR04/321-330 (Cited on page 18).

[GCS94] Steven Gortler, Michael F. Cohen, and Philipp Slusallek. *Radiosity and Relaxation Methods*. In: *IEEE Comput. Graph. Appl.* 14.6 (Nov. 1994), pp. 48–58. ISSN: 0272-1716. DOI: 10.1109/38.329094 (Cited on page 37).

[GD15] Per Ganestam and Michael Doggett. *Real-time multiply recursive reflections and refractions using hybrid rendering*. English. In: *The Visual Computer* 31.10 (2015), pp. 1395–1403. ISSN: 0178-2789. DOI: 10.1007/s00371-014-1021-7 (Cited on pages 63, 64).

[Gee08] Kevin Gee. *Introduction to the Direct3D 11 graphics pipeline*. Presentation, XNA Developer Connection, Microsoft Corporation. 2008 (Cited on page 44).

[Geo+12] Iliyan Georgiev, Jaroslav Křivánek, Tomáš Davidovič, and Philipp Slusallek. *Light Transport Simulation with Vertex Connection and Merging*. In: *ACM Trans. Graph.* 31.6 (Nov. 2012), 192:1–192:10. ISSN: 0730-0301. DOI: 10.1145/2366145.2366211 (Cited on page 38).

[Gla84] A. S. Glassner. *Space subdivision for fast ray tracing*. In: vol. 4. 10. Oct. 1984, pp. 15–24. DOI: 10.1109/MCG.1984.6429331 (Cited on page 55).

[GN12] S. Guntury and P. J. Narayanan. *Raytracing Dynamic Scenes on the GPU Using Grids*. In: *IEEE Transactions on Visualization and Computer Graphics* 18.1 (Jan. 2012), pp. 5–16. ISSN: 1077-2626 (Cited on pages 55, 112, 113).

[Gor+84] Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, and Bennett Battaile. *Modeling the Interaction of Light Between Diffuse Surfaces*. In: *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '84. New York, NY, USA: ACM, 1984, pp. 213–222. ISBN: 0-89791-138-5. DOI: 10.1145/800031.808601 (Cited on page 36).

[Gor+93] Steven J. Gortler, Peter Schröder, Michael F. Cohen, and Pat Hanrahan. *Wavelet Radiosity*. In: *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '93. Anaheim, CA: ACM, 1993, pp. 221–230. ISBN: 0-89791-601-8. DOI: 10.1145/166117.166146 (Cited on pages 18, 37).

[GPM11] Kirill Garanzha, Jacopo Pantaleoni, and David McAllister. *Simpler and Faster HLBVH with Work Queues*. In: *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*. HPG '11. Vancouver, British Columbia, Canada: ACM, 2011, pp. 59–64. ISBN: 978-1-4503-0896-0. DOI: 10.1145/2018323.2018333 (Cited on pages 55, 121).

[Gre+98] Gene Greger, Peter Shirley, Philip M. Hubbard, and Donald P. Greenberg. *The Irradiance Volume*. In: *IEEE Computer Graphics and Applications* 18.2 (Mar. 1998), pp. 32–43. ISSN: 0272-1716. DOI: 10.1109/38.656788 (Cited on page 60).

[Gre03] Robin Green. *Spherical harmonic lighting: The gritty details*. In: *Archives of the Game Developers Conference*. Vol. 56. 2003 (Cited on page 18).

[Hac+12] Toshiya Hachisuka, Wojciech Jarosz, Guillaume Bouchard, Per Christensen, Jeppe Revall Frisvad, Wenzel Jakob, Henrik Wann Jensen, Michael Kaschalk, Claude Knaus, Andrew Selle, and Ben Spencer. *State of the Art in Photon Density Estimation*. In: *ACM SIGGRAPH 2012 Courses*. SIGGRAPH '12. Los Angeles, California: ACM, 2012, 6:1–6:469. ISBN: 978-1-4503-1678-1. DOI: 10.1145/2343483.2343489 (Cited on page 40).

[Hac05] Toshiya Hachisuka. *High-quality global illumination rendering using rasterization*. In: *GPU Gems* 2 (2005), pp. 615–633 (Cited on page 63).

[Hal99] Roy Hall. *Comparing Spectral Color Computation Methods*. In: *IEEE Comput. Graph. Appl.* 19.4 (July 1999), pp. 36–45. ISSN: 0272-1716. DOI: 10.1109/38.773962 (Cited on page 25).

[HAO05] Jon Hasselgren, Tomas Akenine-Möller, and Lennart Ohlsson. *Conservative rasterization*. In: *GPU Gems* 2 (2005), pp. 677–690 (Cited on page 50).

[Hav00] Vlastimil Havran. "Heuristic ray shooting algorithms". PhD thesis. Faculty of Electrical Engineering, Czech Technical University, Prague, 2000 (Cited on page 55).

[He+91] Xiao D. He, Kenneth E. Torrance, François X. Sillion, and Donald P. Greenberg. *A Comprehensive Physical Model for Light Reflection*. In: *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '91. New York, NY, USA: ACM, 1991, pp. 175–186. ISBN: 0-89791-436-8. DOI: 10.1145/122718.122738 (Cited on page 28).

[Hec90] Paul S. Heckbert. *Adaptive Radiosity Textures for Bidirectional Ray Tracing*. In: *Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '90. Dallas, TX, USA: ACM, 1990, pp. 145–154. ISBN: 0-89791-344-2. DOI: 10.1145/97879.97895 (Cited on page 32).

[Hei+16] Eric Heitz, Johannes Hanika, Eugene d'Eon, and Carsten Dachsbacher. *Multiple-scattering Microfacet BSDFs with the Smith Model*. In: *ACM Trans. Graph.* 35.4 (July 2016), 58:1–58:14. ISSN: 0730-0301. DOI: 10.1145/2897824.2925943 (Cited on page 27).

[Her+10] Jan Hermes, Niklas Henrich, Thorsten Grosch, and Stefan Mueller. *Global Illumination using Parallel Global Ray-Bundles*. In: *Vision, Modeling, and Visualization (2010)*. The Eurographics Association, 2010. ISBN: 978-3-905673-79-1. DOI: 10.2312/PE/VMV/VMV10/065-072 (Cited on page 63).

[Hil+15] Stephen Hill, Stephen McAuley, Brent Burley, Danny Chan, Luca Fascione, Naty Iwanicki Michałand Hoffman, Wenzel Jakob, David Neubelt, Angelo Pesce, and Matt Pettineo. *Physically Based Shading in Theory and Practice*. In: *ACM SIGGRAPH 2015 Courses*. SIGGRAPH '15. Los Angeles, California: ACM, 2015, 22:1–22:8. ISBN: 978-1-4503-3634-5. DOI: 10.1145/2776880.2787670 (Cited on page 42).

[HJ08] Jared Hoberock and Yuntao Jia. *High-Quality Ambient Occlusion*. In: *GPU Gems 3*. Ed. by Hubert Nguyen. Addison-Wesley, 2008, pp. 257–274 (Cited on page 57).

[HKH11] Michal Hapala, Ondřej Karlık, and Vlastimil Havran. *When it makes sense to use uniform grids for ray tracing*. In: *Proceedings of WSCG'2011, communication papers* (2011), pp. 193–200 (Cited on page 55).

[HLZ10] Michael Holroyd, Jason Lawrence, and Todd Zickler. *A Coaxial Optical Scanner for Synchronous Acquisition of 3D Geometry and Surface Reflectance*. In: *ACM SIGGRAPH 2010 Papers*. SIGGRAPH '10. Los Angeles, California: ACM, 2010, 99:1–99:12. ISBN: 978-1-4503-0210-4. DOI: 10.1145/1833349.1778836 (Cited on page 28).

[Hol+11] Matthias Holländer, Tobias Ritschel, Elmar Eisemann, and Tamy Boubekeur. *ManyLoDs: Parallel Many-view Level-of-detail Selection for Real-time Global Illumination*. In: *Proceedings of the Twenty-second Eurographics Conference on Rendering*. EGSR '11. Prague, Czech Republic: Eurographics Association, 2011, pp. 1233–1240. DOI: 10.1111/j.1467-8659.2011.01982.x (Cited on page 40).

[HP15] Nicolas Holzschuch and Romain Pacanowski. *A physically accurate reflectance model combining reflection and diffraction*. Research Report RR-8807. INRIA, Nov. 2015, p. 24. URL: https://hal.inria.fr/hal-01224702 (Cited on page 22).

[HPJ12] Toshiya Hachisuka, Jacopo Pantaleoni, and Henrik Wann Jensen. *A Path Space Extension for Robust Light Transport Simulation*. In: *ACM Trans. Graph.* 31.6 (Nov. 2012), 191:1–191:10. ISSN: 0730-0301. DOI: 10.1145/2366145.2366210 (Cited on page 38).

[HQ07] Wei Hu and Kaihuai Qin. *Interactive approximate rendering of reflections, refractions, and caustics*. In: *Visualization and Computer Graphics, IEEE Transactions on* 13.1 (2007), pp. 46–57 (Cited on page 59).

[HSA91] Pat Hanrahan, David Salzman, and Larry Aupperle. *A Rapid Hierarchical Radiosity Algorithm*. In: *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '91. New York, NY, USA: ACM, 1991, pp. 197–206. ISBN: 0-89791-436-8. DOI: 10.1145/122718.122740 (Cited on page 37).

[Hu+14] Wei Hu, Yangyu Huang, Fan Zhang, Guodong Yuan, and Wei Li. *Ray tracing via GPU rasterization*. English. In: *The Visual Computer* 30.6-8 (2014), pp. 697–706. ISSN: 0178-2789. DOI: 10.1007/s00371-014-0968-8 (Cited on pages 63, 126).

[Hul+10] Matthias B. Hullin, Johannes Hanika, Boris Ajdin, Hans-Peter Seidel, Jan Kautz, and Hendrik P. A. Lensch. *Acquisition and Analysis of Bispectral Bidirectional Reflectance and Reradiation Distribution Functions*. In: *ACM SIGGRAPH 2010 Papers*. SIGGRAPH '10. Los Angeles, California: ACM, 2010, 97:1–97:7. ISBN: 978-1-4503-0210-4. DOI: 10.1145/1833349.1778834 (Cited on page 28).

[HW91] Paul S. Heckbert and James M. Winget. *Finite Element Methods for Global Illumination*. Tech. rep. Berkeley, CA, USA: University of California at Berkeley, 1991 (Cited on page 36).

[ICG86] David S. Immel, Michael F. Cohen, and Donald P. Greenberg. *A Radiosity Method for Non-diffuse Environments*. In: *SIGGRAPH Comput. Graph.* 20.4 (Aug. 1986), pp. 133–142. ISSN: 0097-8930. DOI: 10.1145/15886.15901 (Cited on page 37).

[Inc] The Khronos Group Inc. *Rendering Pipeline Overview — OpenGL Wiki*. URL: https://www.khronos.org/opengl/wiki/Rendering_Pipeline_Overview (Cited on page 45).

[Ing] Ingo Wald. *3D Animation Repository*. URL: http://www.sci.utah.edu/~wald/animrep (Cited on page 131).

[Jen01] Henrik Wann Jensen. *Realistic Image Synthesis Using Photon Mapping*. Natick, MA, USA: A. K. Peters, Ltd., 2001. ISBN: 1-56881-147-0 (Cited on page 39).

[Jen96] Henrik Wann Jensen. *Global illumination using photon maps*. In: *Proceedings of the eurographics workshop on Rendering techniques '96*. Porto, Portugal: Springer-Verlag, 1996, pp. 21–30. ISBN: 3-211-82883-4 (Cited on page 39).

[JF99] Garrett M. Johnson and Mark D. Fairchild. *Full-Spectral Color Calculations in Realistic Image Synthesis*. In: *IEEE Comput. Graph. Appl.* 19.4 (July 1999), pp. 47–53. ISSN: 0272-1716. DOI: 10.1109/38.773963 (Cited on page 25).

[JM12] Wenzel Jakob and Steve Marschner. *Manifold Exploration: A Markov Chain Monte Carlo Technique for Rendering Scenes with Difficult Specular Transport*. In: *ACM Trans. Graph.* 31.4 (July 2012), 58:1–58:13. ISSN: 0730-0301. DOI: 10.1145/2185520.2185554 (Cited on page 38).

[KA13] Tero Karras and Timo Aila. *Fast Parallel Construction of High-quality Bounding Volume Hierarchies*. In: *Proceedings of the 5th High-Performance Graphics Conference*. HPG '13. Anaheim, California: ACM, 2013, pp. 89–99. ISBN: 978-1-4503-2135-8. DOI: 10.1145/2492045.2492055 (Cited on pages 55, 105, 121).

[Kaj86] James T. Kajiya. *The Rendering Equation*. In: *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '86. New York, NY, USA: ACM, 1986, pp. 143–150. ISBN: 0-89791-196-2. DOI: 10.1145/15922.15902 (Cited on pages 2, 7, 29, 38).

[Kap09] Anton Kaplanyan. *Light propagation volumes in cryengine 3*. In: *ACM SIGGRAPH Courses* 7 (2009), p. 2 (Cited on page 61).

[Kap85] Michael R Kaplan. *Space-tracing: A constant time ray-tracer*. In: *State of the Art in Image Synthesis, SIGGRAPH '85 course notes* (July 1985), pp. 149–158 (Cited on page 55).

[Kar12] Tero Karras. *Maximizing Parallelism in the Construction of BVHs, Octrees, and K-d Trees*. In: *Proceedings of the Fourth ACM SIGGRAPH / Eurographics Conference on High-Performance Graphics*. EGGH-HPG'12. Paris, France: Eurographics Association, 2012, pp. 33–37. ISBN: 978-3-905674-41-5. DOI: 10.2312/EGGH/HPG12/033-037 (Cited on page 55).

[KBS11] Javor Kalojanov, Markus Billeter, and Philipp Slusallek. *Two-Level Grids for Ray Tracing on GPUs*. In: *Computer Graphics Forum* 30.2 (2011), pp. 307–314. ISSN: 1467-8659. DOI: 10.1111/j.1467-8659.2011.01862.x (Cited on page 55).

[KD10] Anton Kaplanyan and Carsten Dachsbacher. *Cascaded light propagation volumes for real-time indirect illumination*. In: *I3D '10: Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*. Washington, D.C.: ACM, 2010, pp. 99–107. ISBN: 978-1-60558-939-8 (Cited on pages 18, 61).

[KD13] Anton S. Kaplanyan and Carsten Dachsbacher. *Path Space Regularization for Holistic and Robust Light Transport*. In: *Computer Graphics Forum (Proc. of Eurographics 2013)* 32.2 (2013), pp. 63–72 (Cited on page 38).

[KE09] Murat Kurt and Dave Edwards. *A Survey of BRDF Models for Computer Graphics*. In: *SIGGRAPH Comput. Graph.* 43.2 (May 2009), 4:1–4:7. ISSN: 0097-8930. DOI: 10.1145/1629216.1629222 (Cited on page 28).

[KED11] Anton Kaplanyan, Wolfgang Engel, and Carsten Dachsbacher. *Diffuse Global Illumination with Temporally Coherent Light Propagation Volumes*. In: *GPU Pro 2*. Ed. by Wolfgang Engel. A.K. Peters, Feb. 2011. ISBN: 978-1568817187 (Cited on pages 18, 61, 86).

[Kel97] Alexander Keller. *Instant radiosity*. In: *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1997, pp. 49–56. ISBN: 0-89791-896-7 (Cited on page 40).

[KLZ12] Pyarelal Knowles, Geoff Leach, and Fabio Zambetta. *Efficient Layered Fragment Buffer Techniques*. In: *OpenGL Insights*. Ed. by Patrick Cozzi and Christophe Riccio. CRC Press, 2012, pp. 279–292 (Cited on page 54).

[KLZ13] Pyarelal Knowles, Geoff Leach, and Fabio Zambetta. *Backwards Memory Allocation and Improved OIT*. In: *Pacific Graphics Short Papers*. Ed. by Bruno Levy, Xin Tong, and KangKang Yin. The Eurographics Association, 2013. ISBN: 978-3-905674-50-7. DOI: 10.2312/PE.PG.PG2013short.059-064 (Cited on page 54).

[KLZ14] Pyarelal Knowles, Geoff Leach, and Fabio Zambetta. *Fast Sorting for Exact OIT of Complex Scenes*. In: *Vis. Comput.* 30.6-8 (June 2014), pp. 603–613. ISSN: 0178-2789. DOI: 10.1007/s00371-014-0956-z (Cited on page 54).

[Kon+06] Janne Kontkanen, Emmanuel Turquin, Nicolas Holzschuch, and François X. Sillion. *Wavelet Radiance Transport for Interactive Indirect Lighting*. In: *Proceedings of the 17th Eurographics Conference on Rendering Techniques*. EGSR'06. Nicosia, Cyprus: Eurographics Association, 2006, pp. 161–171. ISBN: 3-905673-35-5. DOI: 10.2312/EGWR/EGSR06/161-171 (Cited on pages 18, 61).

[Kop+12] Daniel Kopta, Thiago Ize, Josef Spjut, Erik Brunvand, Al Davis, and Andrew Kensler. *Fast, effective BVH updates for animated scenes*. In: *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. ACM. 2012, pp. 197–204 (Cited on page 55).

[Kri+05] Jaroslav Krivanek, Pascal Gautron, Sumanta Pattanaik, and Kadi Bouatouch. *Radiance Caching for Efficient Global Illumination*. In: *IEEE Transactions on Visualization and Computer Graphics*. 2005, pp. 550–561 (Cited on pages 18, 60).

[Kři+13] Jaroslav Křivánek, Iliyan Georgiev, Anton Kaplanyan, and Juan Canada. *Recent Advances in Light Transport Simulation: Theory and Practice*. In: *ACM SIGGRAPH 2013 Courses*. SIGGRAPH '13. Los Angeles, California: ACM, 2013. ISBN: 978-1-4503-1678-1 (Cited on page 38).

[Kři+14] Jaroslav Křivánek, Alexander Keller, Iliyan Georgiev, Anton Kaplanyan, Marcos Fajardo, Mark Meyer, Jean-Daniel Nahmias, Ondřej Karlík, and Juan Canada. *Recent Advances in Light Transport Simulation: Some Theory and a Lot of Practice*. In: *ACM SIGGRAPH 2014 Courses*. SIGGRAPH '14. Vancouver, Canada: ACM, 2014, 17:1–17:6. ISBN: 978-1-4503-2962-0. DOI: 10.1145/2614028.2615438 (Cited on page 38).

[KS01] Csaba Kelemen and László Szirmay-Kalos. *Simple and Robust Mutation Strategy for Metropolis Light Transport Algorithm*. Tech. rep. TR-186-2-01-18. human contact: technical-report@cg.tuwien.ac.at. Favoritenstrasse 9-11/186, A-1040 Vienna, Austria: Institute of Computer Graphics and Algorithms, Vienna University of Technology, July 2001 (Cited on page 38).

[Laf+97] Eric P. F. Lafortune, Sing-Choong Foo, Kenneth E. Torrance, and Donald P. Greenberg. *Non-linear Approximation of Reflectance Functions*. In: *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '97. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1997, pp. 117–126. ISBN: 0-89791-896-7. DOI: 10.1145/258734.258801 (Cited on page 28).

[Lai+07a] Yu-Chi Lai, Shao Hua Fan, Stephen Chenney, and Charcle Dyer. *Photorealistic Image Rendering with Population Monte Carlo Energy Redistribution*. In: *Proceedings of the 18th Eurographics Conference on Rendering Techniques*. EGSR'07. Grenoble, France: Eurographics Association, 2007, pp. 287–295. ISBN: 978-3-905673-52-4. DOI: 10.2312/EGWR/EGSR07/287-295 (Cited on page 38).

[Lai+07b] Samuli Laine, Hannu Saransaari, Janne Kontkanen, Jaakko Lehtinen, and Timo Aila. *Incremental Instant Radiosity for Real-time Indirect Illumination*. In: *Proceedings of the 18th Eurographics Conference on Rendering Techniques*. EGSR'07. Grenoble, France: Eurographics Association, 2007, pp. 277–286. ISBN: 978-3-905673-52-4. DOI: 10.2312/EGWR/EGSR07/277-286 (Cited on page 59).

[Lam60] J. H. Lambert. *Photometria, sive, De mensura et gradibus luminis, colorum et umbrae*. Augsburg, Germany: V. E. Klett, 1760 (Cited on page 27).

[LCD06] Thomas Luft, Carsten Colditz, and Oliver Deussen. *Image enhancement by unsharp masking the depth buffer*. In: *ACM Trans. Graph.* 25.3 (July 2006), pp. 1206–1213. ISSN: 0730-0301. DOI: 10.1145/1141911.1142016 (Cited on page 57).

[Leh+13] Jaakko Lehtinen, Tero Karras, Samuli Laine, Miika Aittala, Frédo Durand, and Timo Aila. *Gradient-domain Metropolis Light Transport*. In: *ACM Trans. Graph.* 32.4 (July 2013), 95:1–95:12. ISSN: 0730-0301. DOI: 10.1145/2461912.2461943 (Cited on page 38).

[Len04] H.P.A. Lensch. *Efficient, Image-based Appearance Acquisition of Real-world Objects*. Cuvillier, 2004. ISBN: 9783898739962 (Cited on page 28).

[Leu+06] Chi-Sing Leung, Tien-Tsin Wong, Ping-Man Lam, and Kwok-Hung Choy. *An RBF-based Compression Method for Image-based Relighting*. In: *Trans. Img. Proc.* 15.4 (Apr. 2006), pp. 1031–1041. ISSN: 1057-7149. DOI: 10.1109/TIP.2005.863936 (Cited on pages 18, 61).

[Liu+09a] Fang Liu, Meng-Cheng Huang, Xue-Hui Liu, and En-Hua Wu. *Efficient depth peeling via bucket sort*. In: *Proceedings of the 2009 Conference on High Performance Graphics*. HPG '09. New Orleans, Louisiana: ACM, 2009, pp. 51–57. ISBN: 978-1-60558-603-8 (Cited on page 54).

[Liu+09b] Fang Liu, Meng-Cheng Huang, Xue-Hui Liu, and En-Hua Wu. *Single Pass Depth Peeling via CUDA Rasterizer*. In: *SIGGRAPH 2009: Talks*. SIGGRAPH '09. New Orleans, Louisiana: ACM, 2009, 79:1–79:1. ISBN: 978-1-60558-834-6. DOI: 10.1145/1597990.1598069 (Cited on page 54).

[Liu+10] Fang Liu, Meng-Cheng Huang, Xue-Hui Liu, and En-Hua Wu. *FreePipe: A Programmable Parallel Rendering Architecture for Efficient Multi-fragment Effects*. In: *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. I3D '10. Washington, D.C.: ACM, 2010, pp. 75–82. ISBN: 978-1-60558-939-8. DOI: 10.1145/1730804.1730817 (Cited on page 53).

[LK10] Samuli Laine and Tero Karras. *Efficient Sparse Voxel Octrees*. In: *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. I3D '10. Washington, D.C.: ACM, 2010, pp. 55–63. ISBN: 978-1-60558-939-8. DOI: 10.1145/1730804.1730814 (Cited on page 55).

[LS10] Bradford James Loos and Peter-Pike Sloan. *Volumetric obscurance*. In: *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*. I3D '10. Washington, D.C.: ACM, 2010, pp. 151–156. ISBN: 978-1-60558-939-8. DOI: 10.1145/1730804.1730829 (Cited on page 57).

[LTG92] Dani Lischinski, Filippo Tampieri, and Donald P. Greenberg. *Discontinuity Meshing for Accurate Radiosity*. In: *IEEE Comput. Graph. Appl.* 12.6 (Nov. 1992), pp. 25–39. ISSN: 0272-1716. DOI: 10.1109/38.163622 (Cited on page 37).

[LTG93] Dani Lischinski, Filippo Tampieri, and Donald P. Greenberg. *Combining Hierarchical Radiosity and Discontinuity Meshing*. In: *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '93. Anaheim, CA: ACM, 1993, pp. 199–208. ISBN: 0-89791-601-8. DOI: 10.1145/166117.166143 (Cited on page 37).

[LW93] Eric P Lafortune and Yves D Willems. *Bi-directional path tracing*. In: *Proceedings of CompuGraphics*. Vol. 93. 1993, pp. 145–153 (Cited on page 38).

[Mar+14] Michael Mara, Morgan McGuire, Derek Nowrouzezahrai, and David Luebke. *Fast Global Illumination Approximations on Deep G-Buffers*. Tech. rep. NVR-2014-001. NVIDIA Corporation, June 2014, p. 16. URL: http://graphics.cs.williams.edu/papers/DeepGBuffer14 (Cited on pages 49, 54, 60).

[Mat+03] Wojciech Matusik, Hanspeter Pfister, Matthew Brand, and Leonard McMillan. *Efficient Isotropic BRDF Measurement*. In: *Proceedings of the 14th Eurographics Workshop on Rendering*. EGRW '03. Leuven, Belgium: Eurographics Association, 2003, pp. 241–247. ISBN: 3-905673-03-7 (Cited on page 28).

[Mat+15] O. Mattausch, J. Bittner, A. Jaspe, E. Gobbetti, M. Wimmer, and R. Pajarola. *CHC+RT: Coherent Hierarchical Culling for Ray Tracing*. In: *Computer Graphics Forum* 34.2 (2015), pp. 537–548. ISSN: 1467-8659. DOI: 10.1111/cgf.12582 (Cited on page 127).

[Mau+12] Marilena Maule, Joao L. D. Comba, Rafael Torchelsen, and Rui Bastos. *Memory-Efficient Order-Independent Transparency with Dynamic Fragment Buffer*. In: *Proceedings of the 2012 25th SIBGRAPI Conference on Graphics, Patterns and Images*. SIBGRAPI '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 134–141. ISBN: 978-0-7695-4829-6. DOI: 10.1109/SIBGRAPI.2012.27 (Cited on page 53).

[Mav13] Pavlos Mavridis. "Efficient texture representation and sampling algorithms for real-time rendering". PhD thesis. 2013 (Cited on page 50).

[Max86] N L Max. *Shadows for bump-mapped surfaces*. In: *Proceedings of Computer Graphics Tokyo '86 on Advanced Computer Graphics*. Tokyo, Japan: Springer-Verlag New York, Inc., 1986, pp. 145–156. ISBN: 4-431-70011-0 (Cited on page 57).

[McA+13] Stephen McAuley, Stephen Hill, Adam Martinez, Ryusuke Villemin, Matt Pettineo, Dimitar Lazarov, David Neubelt, Brian Karis, Christophe Hery, Naty Hoffman, and Hakan Zap Andersson. *Physically Based Shading in Theory and Practice*. In: *ACM SIGGRAPH 2013 Courses*. SIGGRAPH '13. Anaheim, California: ACM, 2013, 22:1–22:8. ISBN: 978-1-4503-2339-0. DOI: 10.1145/2504435.2504457 (Cited on page 26).

[McG+11] Morgan McGuire, Brian Osman, Michael Bukowski, and Padraic Hennessy. *The Alchemy Screen-Space Ambient Obscurance Algorithm*. In: *High-Performance Graphics 2011*. Vancouver, BC, Canada, Aug. 2011 (Cited on pages 53, 57, 68, 74).

[MER] MERL. *Mitsubishi Electric Research Laboratories BRDF Database*. URL: http://www.merl.com/brdf (Cited on page 28).

[Mey16] J Meyer. *Rendering 'Hitman' with DirectX 12*. In: *Game Developers Conference 2016*. GDC '16. San Franscisco, California, 2016 (Cited on page 4).

[Mil94] Gavin Miller. *Efficient Algorithms for Local and Global Accessibility Shading*. In: *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '94. New York, NY, USA: ACM, 1994, pp. 319–326. ISBN: 0-89791-667-0. DOI: 10.1145/192161.192244 (Cited on page 56).

[Mit07] Martin Mittring. *Finding next gen: CryEngine 2*. In: *ACM SIGGRAPH 2007 courses*. SIGGRAPH '07. San Diego, California: ACM, 2007, pp. 97–121. DOI: 10.1145/1281500.1281671 (Cited on pages 57, 68).

[ML09] Morgan McGuire and David Luebke. *Hardware-accelerated global illumination by image space photon mapping*. In: *HPG '09: Proceedings of the Conference on High Performance Graphics 2009*. New Orleans, Louisiana: ACM, 2009, pp. 77–89. ISBN: 978-1-60558-603-8 (Cited on page 59).

[MLH02] David K. McAllister, Anselmo Lastra, and Wolfgang Heidrich. *Efficient Rendering of Spatial Bi-directional Reflectance Distribution Functions*. In: *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*. HWWS '02. Saarbrücken, Germany: Eurographics Association, 2002, pp. 79–88. ISBN: 1-58113-580-7 (Cited on page 28).

[MM14] Morgan McGuire and Michael Mara. *Efficient GPU Screen-Space Ray Tracing*. In: *Journal of Computer Graphics Techniques (JCGT)* 3.4 (Dec. 2014), pp. 73–85. ISSN: 2331-7418 (Cited on pages 64, 98, 101, 107, 113, 119, 126).

[MML12] Morgan McGuire, Michael Mara, and David Luebke. *Scalable Ambient Obscurance*. In: *High-Performance Graphics 2012*. Paris, France, June 2012 (Cited on pages 58, 69).

[Mor] Morgan McGuire. *Computer Graphics Archive*. URL: https://casual-effects.com/data (Cited on page 131).

[MP11] Pavlos Mavridis and Georgios Papaioannou. *Global illumination using imperfect volumes*. In: *GRAPP '11: Proceedings of the international conference on computer graphics theory and applications*. 2011 (Cited on pages 18, 62).

[MP12] Pavlos Mavridis and Georgios Papaioannou. *The Compact YCoCg Frame Buffer*. In: *Journal of Computer Graphics Techniques (JCGT)* 1.1 (Sept. 2012), pp. 19–35 (Cited on pages 43, 82).

[MS03] H Malvar and G Sullivan. *YCoCg-R: A Color Space with RGB Reversibility and Low Dynamic Range*. Tech. rep. JVTI014r3. 21. Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG, July 2003. URL: http://ftp3.itu.ch/av-arch/jvt-site/2003_09_SanDiego/JVT-I014r3.doc (Cited on page 43).

[MSW10] Oliver Mattausch, Daniel Scherzer, and Michael Wimmer. *High-Quality Screen-Space Ambient Occlusion using Temporal Coherence*. In: *Computer Graphics Forum* 29.8 (Dec. 2010), pp. 2492–2503. ISSN: 0167-7055 (Cited on page 80).

[MW11] David Maletz and Rui Wang. *Importance Point Projection for GPU-based Final Gathering*. In: *Proceedings of the Twenty-second Eurographics Conference on Rendering*. EGSR'11. Prague, Czech Republic: Eurographics Association, 2011, pp. 1327–1336. DOI: 10.1111/j.1467-8659.2011.01992.x (Cited on page 40).

[Nic+77] F. E. Nicodemus, J. C. Richmond, J. J. Hsia, I. W. Ginsberg, and T. Limperis. *Geometric Considerations and Nomenclature for Reflectance*. In: *National Bureau of Standards* (1977) (Cited on pages 26, 28).

[Nie92] Harald Niederreiter. *Random number generation and quasi-Monte Carlo methods*. Vol. 63. SIAM, 1992 (Cited on page 17).

[NN85] Tomoyuki Nishita and Eihachiro Nakamae. *Continuous Tone Representation of Three-dimensional Objects Taking Account of Shadows and Interreflection*. In: *SIGGRAPH Comput. Graph.* 19.3 (July 1985), pp. 23–30. ISSN: 0097-8930. DOI: 10.1145/325165.325169 (Cited on page 36).

[NPG05] Mangesh Nijasure, Sumanta Pattanaik, and Vineet Goel. *Real-Time Global Illumination on GPUs*. In: *Journal of Graphics, GPU, & Game Tools* 10.2 (2005), pp. 55–71 (Cited on pages 18, 61, 62, 83).

[NRH03] Ren Ng, Ravi Ramamoorthi, and Pat Hanrahan. *All-frequency Shadows Using Non-linear Wavelet Lighting Approximation*. In: *ACM SIGGRAPH 2003 Papers*. SIGGRAPH '03. San Diego, California: ACM, 2003, pp. 376–381. ISBN: 1-58113-709-5. DOI: 10.1145/1201775.882280 (Cited on pages 18, 61).

[NRH04] Ren Ng, Ravi Ramamoorthi, and Pat Hanrahan. *Triple Product Wavelet Integrals for All-frequency Relighting*. In: *ACM SIGGRAPH 2004 Papers*. SIGGRAPH '04. Los Angeles, California: ACM, 2004, pp. 477–487. DOI: 10.1145/1186562.1015749 (Cited on pages 18, 61).

[NRS14] Oliver Nalbach, Tobias Ritschel, and Hans-Peter Seidel. *Deep Screen Space*. In: *Proceedings of the 18th Meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. I3D '14. San Francisco, California: ACM, 2014, pp. 79–86. ISBN: 978-1-4503-2717-6. DOI: 10.1145/2556700.2556708 (Cited on pages 45, 60, 107).

[NW09] Greg Nichols and Chris Wyman. *Multiresolution Splatting for Indirect Illumination*. In: *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games*. I3D '09. Boston, Massachusetts: ACM, 2009, pp. 83–90. ISBN: 978-1-60558-429-4. DOI: 10.1145/1507149.1507162 (Cited on page 59).

[ON94] Michael Oren and Shree K. Nayar. *Generalization of Lambert's Reflectance Model*. In: *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '94. New York, NY, USA: ACM, 1994, pp. 239–246. ISBN: 0-89791-667-0. DOI: 10.1145/192161.192213 (Cited on page 28).

[Pap] Georgios Papaioannou. *Computer Graphics Group*. URL: http://graphics.cs.aueb.gr/graphics/downloads.html (Cited on page 131).

[Pap11] Georgios Papaioannou. *Real-time Diffuse Global Illumination Using Radiance Hints*. In: *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*. HPG '11. Vancouver, British Columbia, Canada: ACM, 2011, pp. 15–24. ISBN: 978-1-4503-0896-0. DOI: 10.1145/2018323.2018326 (Cited on pages 18, 62, 82, 83).

[Par+10a] Steven G. Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, and Martin Stich. *OptiX: A General Purpose Ray Tracing Engine*. In: *ACM SIGGRAPH 2010 Papers*. SIGGRAPH '10. Los Angeles, California: ACM, 2010, 66:1–66:13. ISBN: 978-1-4503-0210-4. DOI: 10.1145/1833349.1778803 (Cited on pages 105, 121).

[Par+10b] Steven G. Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, and Martin Stich. *OptiX: A General Purpose Ray Tracing Engine*. In: *ACM Trans. Graph.* 29.4 (July 2010), 66:1–66:13. ISSN: 0730-0301. DOI: 10.1145/1778765.1778803 (Cited on page 55).

[Pee93] Mark S. Peercy. *Linear Color Representations for Full Speed Spectral Rendering*. In: *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '93. Anaheim, CA: ACM, 1993, pp. 191–198. ISBN: 0-89791-601-8. DOI: 10.1145/166117.166142 (Cited on page 25).

[PF90] Pierre Poulin and Alain Fournier. *A Model for Anisotropic Reflection*. In: *Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '90. Dallas, TX, USA: ACM, 1990, pp. 273–282. ISBN: 0-89791-344-2. DOI: 10.1145/97879.97909 (Cited on page 28).

[PH04] Matt Pharr and Greg Humphreys. *Physically based rendering: From theory to implementation*. Morgan Kaufmann, 2004 (Cited on pages 14, 50).

[Pho75] Bui Tuong Phong. *Illumination for Computer Generated Pictures*. In: *Commun. ACM* 18.6 (June 1975), pp. 311–317. ISSN: 0001-0782. DOI: 10.1145/360825.360839 (Cited on page 28).

[Pin88] Juan Pineda. *A Parallel Algorithm for Polygon Rasterization*. In: *SIGGRAPH Comput. Graph.* 22.4 (June 1988), pp. 17–20. ISSN: 0097-8930. DOI: 10.1145/378456.378457 (Cited on page 46).

[PL10] J. Pantaleoni and D. Luebke. *HLBVH: Hierarchical LBVH Construction for Real-time Ray Tracing of Dynamic Geometry*. In: *Proceedings of the Conference on High Performance Graphics*. HPG '10. Saarbrücken, Germany: Eurographics Association, 2010, pp. 87–95 (Cited on pages 55, 121).

[PMP10] Georgios Papaioannou, Maria Lida Menexi, and Charilaos Papadopoulos. *Real-Time Volume-Based Ambient Occlusion*. In: *IEEE Transactions on Visualization and Computer Graphics* 16 (5 Sept. 2010), pp. 752–762. ISSN: 1077-2626. DOI: http://dx.doi.org/10.1109/TVCG.2010.18 (Cited on page 57).

[Pre+96] William H Press, Saul A Teukolsky, William T Vetterling, and Brian P Flannery. *Numerical recipes in C*. Vol. 2. Cambridge university press Cambridge, 1996 (Cited on page 19).

[Ram02] Ravi Ramamoorthi. "A Signal-processing Framework for Forward and Inverse Rendering". AAI3067921. PhD thesis. Stanford, CA, USA, 2002. ISBN: 0-493-87584-0 (Cited on page 18).

[Ren+06] Zhong Ren, Rui Wang, John Snyder, Kun Zhou, Xinguo Liu, Bo Sun, Peter-Pike Sloan, Hujun Bao, Qunsheng Peng, and Baining Guo. *Real-time Soft Shadows in Dynamic Scenes Using Spherical Harmonic Exponentiation*. In: *ACM SIGGRAPH 2006 Papers*. SIGGRAPH '06. Boston, Massachusetts: ACM, 2006, pp. 977–986. ISBN: 1-59593-364-6. DOI: 10.1145/1179352.1141982 (Cited on page 57).

[RGS09] Tobias Ritschel, Thorsten Grosch, and Hans-Peter Seidel. *Approximating Dynamic Global Illumination in Image Space*. In: *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games*. I3D '09. Boston, Massachusetts: ACM, 2009, pp. 75–82. ISBN: 978-1-60558-429-4. DOI: 10.1145/1507149.1507161 (Cited on pages 53, 58, 59, 68, 79).

[RH01] Ravi Ramamoorthi and Pat Hanrahan. *An efficient representation for irradiance environment maps*. In: *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM, 2001, pp. 497–500. ISBN: 1-58113-374-X (Cited on pages 18, 21, 35, 61, 85).

[Rit+08] T. Ritschel, T. Grosch, M. H. Kim, H.-P. Seidel, C. Dachsbacher, and J. Kautz. *Imperfect shadow maps for efficient computation of indirect illumination*. In: *ACM Trans. Graph.* 27.5 (2008), pp. 1–8. ISSN: 0730-0301 (Cited on pages 40, 59).

[Rit+09] T. Ritschel, T. Engelhardt, T. Grosch, H.-P. Seidel, J. Kautz, and C. Dachsbacher. *Micro-rendering for Scalable, Parallel Final Gathering*. In: *ACM SIGGRAPH Asia 2009 Papers*. SIGGRAPH Asia '09. Yokohama, Japan: ACM, 2009, 132:1–132:8. ISBN: 978-1-60558-858-2. DOI: 10.1145/1661412.1618478 (Cited on pages 40, 41).

[Rit+11] Tobias Ritschel, Elmar Eisemann, Inwoo Ha, James D.K. Kim, and Hans-Peter Seidel. *Making Imperfect Shadow Maps View-Adaptive: High-Quality Global Illumination in Large Dynamic Scenes*. In: *Computer Graphics Forum (presented at EGSR 2011)* (2011) (Cited on page 59).

[Rit+12] Tobias Ritschel, Carsten Dachsbacher, Thorsten Grosch, and Jan Kautz. *The State of the Art in Interactive Global Illumination*. In: *Comput. Graph. Forum* 31.1 (Feb. 2012), pp. 160–188. ISSN: 0167-7055. DOI: 10.1111/j.1467-8659.2012.02093.x (Cited on page 4).

[SA07] Perumaal Shanmugam and Okan Arikan. *Hardware Accelerated Ambient Occlusion Techniques on GPUs*. In: *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*. I3D '07. Seattle, Washington: ACM, 2007, pp. 73–80. ISBN: 978-1-59593-628-8. DOI: 10.1145/1230100.1230113 (Cited on page 57).

[Sch05] Volker Schönefeld. *Spherical harmonics*. In: *Computer Graphics and Multimedia Group, Technical Note. RWTH Aachen University, Germany* (2005) (Cited on page 18).

[Sch94] Christophe Schlick. *A Survey of Shading and Reflectance Models*. In: *Computer Graphics forum* 13 (1994), pp. 121–131 (Cited on page 28).

[SFD09] Martin Stich, Heiko Friedrich, and Andreas Dietrich. *Spatial Splits in Bounding Volume Hierarchies*. In: *Proceedings of the Conference on High Performance Graphics 2009*. HPG '09. New Orleans, Louisiana: ACM, 2009, pp. 7–13. ISBN: 978-1-60558-603-8. DOI: 10.1145/1572769.1572771 (Cited on page 105).

[Shi+97] P. Shirley, B. Smits, H. Hu, and E. Lafortune. *A Practitioners' Assessment of Light Reflection Models*. In: *Proceedings of the 5th Pacific Conference on Computer Graphics and Applications*. PG '97. Washington, DC, USA: IEEE Computer Society, 1997, pp. 40–. ISBN: 0-8186-8028-8 (Cited on page 28).

[Shi91] Peter S. Shirley. "Physically Based Lighting Calculations for Computer Graphics". UMI Order NO. GAX91-24487. PhD thesis. Champaign, IL, USA, 1991 (Cited on page 17).

[Sil+91] Françis X. Sillion, James R. Arvo, Stephen H. Westin, and Donald P. Greenberg. *A global illumination solution for general reflectance distributions*. In: *SIGGRAPH Comput. Graph.* 25.4 (1991), pp. 187–196. ISSN: 0097-8930 (Cited on pages 18, 61).

[Ske] Sketchfab. *Model Database*. URL: https://sketchfab.com (Cited on page 131).

[SKP07] Musawir A. Shah, Jaakko Konttinen, and Sumanta Pattanaik. *Caustics Mapping: An Image-Space Technique for Real-Time Caustics*. In: *IEEE Transactions on Visualization and Computer Graphics* 13.2 (Mar. 2007), pp. 272–280. ISSN: 1077-2626. DOI: 10.1109/TVCG.2007.32 (Cited on page 59).

[SKS02] Peter-Pike Sloan, Jan Kautz, and John Snyder. *Precomputed Radiance Transfer for Real-time Rendering in Dynamic, Low-frequency Lighting Environments*. In: *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '02. San Antonio, Texas: ACM, 2002, pp. 527–536. ISBN: 1-58113-521-1. DOI: 10.1145/566570.566612 (Cited on page 18).

[SKS11] Tiago Sousa, N Kasyan, and N Schulz. *Secrets of CryEngine 3 graphics technology*. In: *ACM SIGGRAPH Talks*. 2011 (Cited on page 63).

[Slo08] Peter-Pike Sloan. *Stupid spherical harmonics (sh) tricks*. In: *Game developers conference*. Vol. 9. 2008 (Cited on page 18).

[Smi99] Brian Smits. *An RGB-to-spectrum Conversion for Reflectances*. In: *J. Graph. Tools* 4.4 (Dec. 1999), pp. 11–22. ISSN: 1086-7651. DOI: 10.1080/10867651.1999.10487511 (Cited on page 25).

[Sob94] Ilya M Sobol. *A primer for the Monte Carlo method*. CRC press, 1994 (Cited on pages 13, 38).

[Sol] Solid Angle. *Arnold Renderer*. URL: https://support.solidangle.com/display/AFMUG/Ambient+Occlusion (Cited on page 31).

[SP89] F. Sillion and C. Puech. *A General Two-pass Method Integrating Specular and Diffuse Reflection*. In: *SIGGRAPH Comput. Graph.* 23.3 (July 1989), pp. 335–344. ISSN: 0097-8930. DOI: 10.1145/74334.74368 (Cited on page 37).

[SP94] Francois X. Sillion and Claude Puech. *Radiosity and Global Illumination*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1994. ISBN: 1558602771 (Cited on page 36).

[SP98] László Szirmay-Kalos and Werner Purgathofer. "Global Ray-bundle Tracing with Hardware Acceleration". In: *Rendering Techniques '98: Proceedings of the Eurographics Workshop in Vienna, Austria, June 29—July 1, 1998*. Ed. by George Drettakis and Nelson Max. Vienna: Springer Vienna, 1998, pp. 247–258. ISBN: 978-3-7091-6453-2. DOI: 10.1007/978-3-7091-6453-2_23. URL: http://dx.doi.org/10.1007/978-3-7091-6453-2_23 (Cited on page 63).

[SRS14] Masamichi Sugihara, Randall Rauwendaal, and Marco Salvi. *Layered Reflective Shadow Maps for Voxel-based Indirect Illumination*. In: *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*. HPG '14. Lyon, France: ACM, 2014 (Cited on page 62).

[SS95] Peter Schröder and Wim Sweldens. *Spherical Wavelets: Efficiently Representing Functions on the Sphere*. In: *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '95. New York, NY, USA: ACM, 1995, pp. 161–172. ISBN: 0-89791-701-4. DOI: 10.1145/218380.218439 (Cited on page 18).

[SS96] Mateu Sbert and Xavier Pueyo i Sàndez. "The Use of global random directions to compute radiosity: global Montecarlo techniques". PhD thesis. 1996 (Cited on pages 37, 63).

[ST90] Takafumi Saito and Tokiichiro Takahashi. *Comprehensible Rendering of 3-D Shapes*. In: *Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '90. Dallas, TX, USA: ACM, 1990, pp. 197–206. ISBN: 0-89791-344-2. DOI: 10.1145/97879.97901 (Cited on page 49).

[Sta] Stanford University. *Computer Graphics Laboratory*. URL: http://graphics.stanford.edu/data/3Dscanrep (Cited on page 131).

[Sto05] M. C. Stone. *Representing colors as three numbers [color graphics]*. In: *IEEE Computer Graphics and Applications* 25.4 (July 2005), pp. 78–85. ISSN: 0272-1716. DOI: 10.1109/MCG.2005.84 (Cited on page 42).

[Str90] Paul S. Strauss. *A Realistic Lighting Model for Computer Animators*. In: *IEEE Comput. Graph. Appl.* 10.6 (Nov. 1990), pp. 56–64. ISSN: 0272-1716. DOI: 10.1109/38.62696 (Cited on page 28).

[The+08] Theoharis Theoharis, Georgios Papaioannou, Nikolaos Platis, and Nicholas. M. Patrikalakis. *Graphics and Visualization: Principles & Algorithms*. Ak Peters Series. Taylor & Francis, 2008. ISBN: 9781568812748 (Cited on pages 22, 39).

[Thi+11] Sinje Thiedemann, Niklas Henrich, Thorsten Grosch, and Stefan Müller. *Voxel-based global illumination*. In: *I3D '11: Proceedings of the symposium on interactive 3D graphics and games 2011*. San Francisco, CA, USA: ACM, 2011 (Cited on page 62).

[Tim13] Ville Timonen. *Screen-space Far-field Ambient Obscurance*. In: *Proceedings of the 5th High-Performance Graphics Conference*. HPG '13. Anaheim, California: ACM, 2013, pp. 33–43. ISBN: 978-1-4503-2135-8. DOI: 10.1145/2492045.2492049 (Cited on page 58).

[TL04] Eric Tabellion and Arnauld Lamorlette. *An Approximate Global Illumination System for Computer Generated Films*. In: *ACM Trans. Graph.* 23.3 (Aug. 2004), pp. 469–476. ISSN: 0730-0301. DOI: 10.1145/1015706.1015748 (Cited on page 31).

[TO12] Yusuke Tokuyoshi and Shinji Ogaki. *Real-time Bidirectional Path Tracing via Rasterization*. In: *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. I3D '12. Costa Mesa, California: ACM, 2012, pp. 183–190. ISBN: 978-1-4503-1194-6. DOI: 10.1145/2159616.2159647 (Cited on page 63).

[Tob] Wikipedia Tobias R Metoc. *Wikipedia: Caustics*. URL: http://commons.wikimedia.org/wiki/File:Glas-1000-enery.jpg (Cited on page 35).

[Tok+13] Yusuke Tokuyoshi, Takashi Sekine, Tiago da Silva, and Takashi Kanai. *Adaptive Ray-bundle Tracing with Memory Usage Prediction: Efficient Global Illumination in Large Scenes*. In: *Computer Graphics Forum* 32.7 (2013), pp. 315–324. ISSN: 1467-8659 (Cited on page 63).

[TS67] Kenneth Torrance and Edward Sparrow. *Theory for Off-Specular Reflection from Roughened Surfaces*. In: *Journal of Optical Society of America* 57.9 (Sept. 1967), pp. 1105–1114 (Cited on page 28).

[Ulu14] Yasin Uludag. *Hi-Z Screen-Space Cone-Traced Reflections*. In: *GPU Pro 5*. Ed. by Wolfgang Engel. CRC Press, 2014, pp. 149–192 (Cited on pages 64, 102, 107, 113, 115, 119).

[Unia] Unity Technologies. *The Blacksmith Environments*. URL: https://www.assetstore.unity3d.com/en/#!/content/39948 (Cited on page 131).

[Unib] University of North Carolina. *GAMMA Research Group - Dynamic Scene Benchmarks*. URL: http://gamma.cs.unc.edu/DYNAMICB (Cited on page 131).

[VC07] JMP Van Waveren and Ignacio Castaño. *Real-time YCoCg-DXT compression*. Tech. rep. NVIDIA Corporation, Sept. 2007, p. 16. URL: http://www.nvidia.com/object/real-time-ycocg-dxt-compression.html (Cited on page 43).

[Vea98] Eric Veach. "Robust Monte Carlo Methods for Light Transport Simulation". AAI9837162. PhD thesis. Stanford, CA, USA, 1998. ISBN: 0-591-90780-1 (Cited on pages 7, 17, 22, 38).

[VF12] Andreas A. Vasilakis and Ioannis Fudos. *S-buffer: Sparsity-aware multi-fragment rendering*. In: *Proceedings of Eurographics 2012 Short Papers*. EG '12. Cagliari, Sardinia, Italy, 2012, pp. 101–104 (Cited on pages 53, 108).

[VF13] Andreas-A. Vasilakis and Ioannis Fudos. *Depth-Fighting Aware Methods for Multifragment Rendering*. In: *IEEE Transactions on Visualization and Computer Graphics* 19.6 (2013), pp. 967–977. ISSN: 1077-2626 (Cited on pages 54, 99, 101, 109, 115).

[VF14] Andreas A. Vasilakis and Ioannis Fudos. *k^+-buffer: Fragment Synchronized K-buffer*. In: *Proceedings of the 18th Meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. I3D '14. San Francisco, California: ACM, 2014, pp. 143–150. ISBN: 978-1-4503-2717-6 (Cited on page 54).

[VH74] W.H. Venable and J.J Hsia. *Optical radiation measurements : describing spectrophotometric measurements*. NBS technical note. U.S. Dept. of Commerce, National Bureau of Standards: for sale by the Supt. of Docs., U.S. Govt. Print. Off., 1974 (Cited on page 28).

[VHB15] Marek Vinkler, Vlastimil Havran, and Jiri Bittner. *Performance Comparison of Bounding Volume Hierarchies and Kd-Trees for GPU Ray Tracing*. In: *Computer Graphics Forum* (2015), n/a–n/a. ISSN: 1467-8659. DOI: 10.1111/cgf.12776 (Cited on page 54).

[VP] Kostas Vardis and Georgios Papaioannou. *XEngine*. URL: http://graphics.cs.aueb.gr/graphics/downloads_tutorials.html (Cited on pages 12, 14, 33, 36, 50, 51, 61).

[VPF15] A. A. Vasilakis, G. Papaioannou, and I. Fudos. *k^+-buffer: An Efficient, Memory-Friendly and Dynamic k-buffer Framework*. In: *IEEE Transactions on Visualization and Computer Graphics* 21.6 (June 2015), pp. 688–700. ISSN: 1077-2626. DOI: 10.1109/TVCG.2015.2417581 (Cited on page 54).

[VVP16] Kostas Vardis, Andreas A. Vasilakis, and Georgios Papaioannou. *A Multiview and Multilayer Approach for Interactive Ray Tracing*. In: *Proceedings of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. I3D '16. Redmond, Washington: ACM, 2016, pp. 171–178. ISBN: 978-1-4503-4043-4. DOI: 10.1145/2856400.2856401 (Cited on pages 112, 113, 116, 119, 121, 124, 126).

[Wal] Walt Disney Animation Studios. *BRDF Explorer - Development and analysis of bidirectional reflectance distribution functions (BRDFs)*. URL: https://www.disneyanimation.com/technology/brdf.html (Cited on page 27).

[Wal+07] Bruce Walter, Stephen R. Marschner, Hongsong Li, and Kenneth E. Torrance. *Microfacet Models for Refraction Through Rough Surfaces*. In: *Proceedings of the 18th Eurographics Conference on Rendering Techniques*. EGSR'07. Grenoble, France: Eurographics Association, 2007, pp. 195–206. ISBN: 978-3-905673-52-4. DOI: 10.2312/EGWR/EGSR07/195-206 (Cited on page 28).

[Wal+09] Ingo Wald, William R. Mark, Johannes Gunther, Solomon Boulos, Thiago Ize, Warren Hunt, Steven G. Parker, and Peter Shirley. *State of the Art in Ray Tracing Animated Scenes*. In: *Computer Graphics Forum* 28.6 (2009), pp. 1691–1722. ISSN: 1467-8659. DOI: 10.1111/j.1467-8659.2008.01313.x (Cited on page 54).

[War92] Gregory J. Ward. *Measuring and Modeling Anisotropic Reflection*. In: *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '92. New York, NY, USA: ACM, 1992, pp. 265–272. ISBN: 0-89791-479-1. DOI: 10.1145/133994.134078 (Cited on pages 28, 60).

[War94] Gregory J. Ward. *The RADIANCE Lighting Simulation and Rendering System*. In: *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '94. New York, NY, USA: ACM, 1994, pp. 459–472. ISBN: 0-89791-667-0. DOI: 10.1145/192161.192286 (Cited on page 60).

[WAT92] Stephen H. Westin, James R. Arvo, and Kenneth E. Torrance. *Predicting Reflectance Functions from Complex Surfaces*. In: *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '92. New York, NY, USA: ACM, 1992, pp. 255–264. ISBN: 0-89791-479-1. DOI: 10.1145/133994.134075 (Cited on page 18).

[WBS07] Ingo Wald, Solomon Boulos, and Peter Shirley. *Ray Tracing Deformable Scenes Using Dynamic Bounding Volume Hierarchies*. In: *ACM Trans. Graph.* 26.1 (Jan. 2007). ISSN: 0730-0301. DOI: 10.1145/1189762.1206075 (Cited on page 55).

[WD06] Chris Wyman and Scott Davis. *Interactive Image-space Techniques for Approximating Caustics*. In: *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games*. I3D '06. Redwood City, California: ACM, 2006, pp. 153–160. ISBN: 1-59593-295-X. DOI: 10.1145/1111411.1111439 (Cited on page 59).

[WD08] Chris Wyman and Carsten Dachsbacher. *Reducing noise in image-space caustics with variable-sized splatting*. In: *Journal of Graphics, GPU, and Game Tools* 13.1 (2008), pp. 1–17 (Cited on page 59).

[WE02] Greg Ward and Elena Eydelberg-Vileshin. *Picture Perfect RGB Rendering Using Spectral Prefiltering and Sharp Color Primaries*. In: *Proceedings of the 13th Eurographics Workshop on Rendering*. EGRW '02. Pisa, Italy: Eurographics Association, 2002, pp. 117–124. ISBN: 1-58113-534-3 (Cited on page 25).

[Wey+08] Tim Weyrich, Jason Lawrence, Hendrik Lensch, Szymon Rusinkiewicz, and Todd Zickler. *Principles of Appearance Acquisition and Representation*. In: *ACM SIGGRAPH 2008 Classes*. SIGGRAPH '08. Los Angeles, California: ACM, 2008, 80:1–80:119. DOI: 10.1145/1401132.1401234 (Cited on page 28).

[WH92] Gregory J Ward and Paul Heckbert. *Irradiance gradients*. In: *Third Eurographics Workshop on Rendering*. Vol. 8598. 1992 (Cited on page 39).

[WHG84] Hank Weghorst, Gary Hooper, and Donald P. Greenberg. *Improved Computational Methods for Ray Tracing*. In: *ACM Trans. Graph.* 3.1 (Jan. 1984), pp. 52–69. ISSN: 0730-0301. DOI: 10.1145/357332.357335 (Cited on page 55).

[Whi80] Turner Whitted. *An Improved Illumination Model for Shaded Display*. In: *Commun. ACM* 23.6 (June 1980), pp. 343–349. ISSN: 0001-0782. DOI: 10.1145/358876.358882 (Cited on pages 7, 38).

[Wid+15] S. Widmer, D. Pajak, A. Schulz, K. Pulli, J. Kautz, M. Goesele, and D. Luebke. *An Adaptive Acceleration Structure for Screen-space Ray Tracing*. In: *Proceedings of the 7th Conference on High-Performance Graphics*. HPG '15. Los Angeles, California: ACM, 2015, pp. 67–76. ISBN: 978-1-4503-3707-6. DOI: 10.1145/2790060.2790069 (Cited on pages 63, 126).

[Wie66] J.A. Wiebelt. *Engineering radiation heat transfer*. Holt, Rinehart and Winston, 1966 (Cited on page 36).

[Wika] Wikipedia. *Ray Tracing — Wikipedia, The Free Encyclopedia*. URL: %5Curl%7Bhttps://en.wikipedia.org/wiki/Ray_tracing_(graphics)%7D (Cited on page 7).

[Wikb] Wikipedia. *YCbCr — Wikipedia, The Free Encyclopedia*. URL: https://en.wikipedia.org/wiki/YCbCr (Cited on page 43).

[Wil78] Lance Williams. *Casting Curved Shadows on Curved Surfaces*. In: *Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '78. New York, NY, USA: ACM, 1978, pp. 270–274. DOI: 10.1145/800248.807402 (Cited on pages 49, 53).

[WK90] Lawrence B. Wolff and David J. Kurlander. *Ray Tracing with Polarization Parameters*. In: *IEEE Comput. Graph. Appl.* 10.6 (Nov. 1990), pp. 44–55. ISSN: 0272-1716. DOI: 10.1109/38.62695 (Cited on page 22).

[WN09] Chris Wyman and Greg Nichols. *Adaptive Caustic Maps Using Deferred Shading*. In: *Computer Graphics Forum* 28.2 (2009), pp. 309–318. ISSN: 1467-8659. DOI: 10.1111/j.1467-8659.2009.01370.x (Cited on page 59).

[WRC88] Gregory J. Ward, Francis M. Rubinstein, and Robert D. Clear. *A ray tracing solution for diffuse interreflection*. In: *SIGGRAPH Comput. Graph.* 22 (4 June 1988), pp. 85–92. ISSN: 0097-8930 (Cited on pages 39, 60).

[WTP01] Alexander Wilkie, Robert F. Tobler, and Werner Purgathofer. *Combined Rendering of Polarization and Fluorescence Effects*. In: *Proceedings of the 12th Eurographics Conference on Rendering*. EGWR'01. London, UK: Eurographics Association, 2001, pp. 197–204. ISBN: 3-211-83709-4. DOI: 10.2312/EGWR/EGWR01/197-204 (Cited on page 22).

[WW08] Andrea Weidlich and Alexander Wilkie. *Realistic Rendering of Birefringency in Uniaxial Crystals*. In: *ACM Trans. Graph.* 27.1 (Mar. 2008), 6:1–6:12. ISSN: 0730-0301. DOI: 10.1145/1330511.1330517 (Cited on page 22).

[WW12] Alexander Wilkie and Andrea Weidlich. *Polarised Light in Computer Graphics*. In: *SIGGRAPH Asia 2012 Courses*. SA '12. Singapore, Singapore: ACM, 2012, 8:1–8:87. ISBN: 978-1-4503-1913-3. DOI: 10.1145/2407783.2407791 (Cited on page 22).

[Wym08] Chris Wyman. *Hierarchical Caustic Maps*. In: *Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games*. I3D '08. Redwood City, California: ACM, 2008, pp. 163–171. ISBN: 978-1-59593-983-8. DOI: 10.1145/1342250.1342276 (Cited on page 59).

[Yan+10] Jason C. Yang, Justin Hensley, Holger Grün, and Nicolas Thibieroz. *Real-time Concurrent Linked List Construction on the GPU*. In: *Proceedings of the 21st Eurographics Conference on Rendering*. EGSR'10. Saarbrücken, Germany: Eurographics Association, 2010, pp. 1297–1304. DOI: 10.1111/j.1467-8659.2010.01725.x (Cited on pages 48, 53, 109).

[YCM07] Sung-Eui Yoon, Sean Curtis, and Dinesh Manocha. *Ray tracing dynamic scenes using selective restructuring*. In: *Proceedings of the 18th Eurographics conference on Rendering Techniques*. Eurographics Association. 2007, pp. 73–84 (Cited on page 55).

[YN04] Yangli Hector Yee and Anna Newman. *A Perceptual Metric for Production Testing*. In: *ACM SIGGRAPH 2004 Sketches*. SIGGRAPH '04. Los Angeles, California: ACM, 2004, pp. 121–. ISBN: 1-58113-896-2. DOI: 10.1145/1186223.1186374 (Cited on page 91).

[YT10] Cem Yuksel and Sarah Tariq. *Advanced Techniques in Real-time Hair Rendering and Simulation*. In: *ACM SIGGRAPH 2010 Courses*. SIGGRAPH '10. Los Angeles, California: ACM, 2010, 1:1–1:168. ISBN: 978-1-4503-0395-8. DOI: 10.1145/1837101.1837102 (Cited on page 45).

[Yu+09] Insu Yu, Andrew Cox, Min H. Kim, Tobias Ritschel, Thorsten Grosch, Carsten Dachsbacher, and Jan Kautz. *Perceptual Influence of Approximate Visibility in Indirect Illumination*. In: *ACM Trans. Appl. Percept.* 6.4 (Oct. 2009), 24:1–24:14. ISSN: 1544-3558. DOI: 10.1145/1609967.1609971 (Cited on page 90).

[Yu+12] Xuan Yu, Jason C. Yang, Justin Hensley, Takahiro Harada, and Jingyi Yu. *A framework for rendering complex scattering effects on hair*. In: *Proceedings of the 2012 symposium on Interactive 3D Graphics and Games*. I3D '12. Costa Mesa, California: ACM, 2012, pp. 111–118. ISBN: 978-1-4503-1194-6 (Cited on page 54).

[Zho+08] Kun Zhou, Qiming Hou, Rui Wang, and Baining Guo. *Real-time KD-tree Construction on Graphics Hardware*. In: *ACM Trans. Graph.* 27.5 (Dec. 2008), 126:1–126:11. ISSN: 0730-0301. DOI: 10.1145/1409060.1409079 (Cited on page 55).

[ZIK98] Sergej Zhukov, Andrej Inoes, and Grigorij Kronin. *An Ambient Light Illumination Model*. In: *Rendering Techniques '98*. Ed. by George Drettakis and Nelson Max. Eurographics. Springer-Verlag Wien New York, 1998, pp. 45–56 (Cited on pages 33, 56).

[ZRD14] Tobias Zirr, Hauke Rehfeld, and Carsten Dachsbacher. *Object-Order Ray Tracing for Fully Dynamic Scenes*. In: *GPU Pro 5*. Ed. by Wolfgang Engel. CRC Press, 2014, pp. 419–438 (Cited on page 63).